

# Software Sentiment Analysis using Deep-learning Approach with Word-Embedding Techniques

Venkata Krishna Chandra Mula<sup>1</sup>

Dept. CSIS  
Andhra University College of Engineering  
krishnacmula@gmail.com

Lalita Bhanu Murthy<sup>3</sup>

Department of Computer Science & Information Systems  
BITS Pilani Hyderabad Campus  
bhanu@hyderabad.bits-pilani.ac.in

Lov Kumar<sup>2</sup>

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

Prof. Aneesh Krishna<sup>4</sup>

Curtin University  
A.Krishna@curtin.edu.au

**Abstract**—Sentiment Analysis in the Software Engineering community aims to make the development and maintenance of software a better experience by helping provide code and library suggestions, defect-related comments for source code, etc. The manual finding of sentiment-based comments may be an inaccurate prediction and a time-consuming process. Automating the sentiment analysis process by leveraging Machine Learning models can benefit software professionals by giving them insights into other developers and feelings about software products, libraries, development, and maintenance tasks at a glance. This study aims to develop software sentiment prediction models based on comments by (1) identifying the best embedding techniques to represent the word of the comments, not just as a number but as a vector in n-dimensional space (2) finding the best sets of vectors using different features selection techniques (3) finding the best methods to handle the class imbalance nature of the data, and (4) finding the best architecture of deep-learning for the training of models. The developed models are validated using 5-fold cross-validation with four different performance parameters: accuracy, AUC, recall, and precision on three different datasets. The experimental finding shows that the models developed using the word embeddings with feature selection using Deep Learning classifiers on balanced data can significantly predict the underlying sentiments of textual comments.

**Keywords**—Sentiment Analysis, Software Engineering Tasks, Word Embedding, Feature Selection, Data Imbalance, SMOTE, Deep Learning classifiers

## I. INTRODUCTION

**S**ENTIMENT Analysis can be used to gather the opinions and feelings of the consumers regarding social and political opinions, brand loyalty, etc. Sentiment Analysis utilizes natural language processing and machine learning algorithms to draw out textual data's mood, opinions, and feelings. The texts can be product reviews, posts on social media, messages on chat boards, answers to questions on a question-answering website, commit messages by developers, etc [1]. Software developers can utilize the benefits of sentiment analysis to assist them in their development and maintenance activities. They can be well informed about whether a particular software, technology, or tutorial is appropriate for their purposes. Sentiment Analysis can distinguish feedback as being either positive or negative, which helps in development decisions. The application of

sentiment analysis is to find Software professionals' sentiments and mindsets, and it can be approached in two different ways: The first and most straightforward way is to sit down face to face with the software developers, glean insights into their mindset, and evaluate their mood and feelings. Unfortunately, this is a very time-intensive and tedious process to incorporate. So, the other approach is preferred. The other approach uses sentiment analysis to discern the mood and feelings of software developers, from the product reviews, feedback forms, commit messages, etc. An effort is made to identify the positive and negative sentiments of developers. So, we have worked to create a predictive model leveraging natural language processing techniques and machine learning algorithms, to predict and detect the exhibited sentiments, moods, and feelings effectively and efficiently. The datasets are obtained from user's App Reviews, and issues tracked and managed using JIRA, and user's comments and messages on the Stack Overflow platform [1]. For the machine to better understand and analyze the text to improve the predictive model's performance, we must represent words as vectors [2][3].

Word embedding techniques do this by representing words as vectors in an n-dimensional space. This provides a numeric representation to the words, which allows them to be used as input to Machine Learning Models, and it also preserves its syntactic and semantic integrity so that words that are used similarly have similar vector representations. In this study, we use six Word Embedding Techniques to vectorize the textual data, which are Term Frequency and Inverse Document Frequency (TF-IDF), Skip-Gram (SKG), Continuous Bag of Words (CBOW), Global Vectors for Word Representations (GLOVE), Fast Text (FST) and Google News Word to Vector (GW2V) [3]. After applying the word embeddings, we obtain a multitude of features for the data, many of which will be ineffective in the predictive model. To obtain the subset of important features that are to be used as input to the model, we apply six Feature Selection Techniques, namely Principal Components Analysis, Gain Ratio Attribute Eval, Classifier Attribute Eval, Info Gain Attribute Eval, OneR Attribute Eval, and Analysis of Variance (ANOVA) [4]. Analyzing the data after applying the Feature Selection Techniques, it is clear that the data suffer from the class imbalance problem, which

occurs when the number of samples in each class is not the same. If not corrected, this can negatively affect the predictive performance of the model. So, the Synthetic Minority Oversampling Technique (SMOTE) and the Borderline Synthetic Minority Oversampling Technique (Borderline-SMOTE) are applied to balance the data. After we balance the data, we need to compare and evaluate the performance of the different techniques we have applied. To achieve this, we use eight different deep learning classifiers, which are applied by varying the number of Hidden Layers and Dropout Layers. The application of Deep Learning Classifiers to the models developed using different Word Embedding Techniques can help determine the models that can accurately and effectively predict the underlying sentiment in textual data, which can be convenient for a broad scope of Software Development and Maintenance activities. This study also aims to find the Word Embeddings, Feature Selection, and Data Sampling Techniques that provide the most optimal results.

The remainder of the paper is laid out as follows: Section 2 presents a literature review on software sentiment analysis and various word embedding approaches. Section 3 describes the experimental dataset collection as well as the various machine learning algorithms used. The research methodology is described in Section 4 using an architecture framework. In Section 5, the results of the experiments, along with their analysis, are presented. Section 6 shows a comparison of models created using various word-embedding approaches, sets of features, and machine learning models. Finally, Section 7 summarizes the information provided and offers directions for further research.

## II. RELATED WORK

There are many methods to acquire features from textual data. Term Frequency and Inverse Document Frequency (TF-IDF) have been used by Rajni Jindal et al. to obtain features from defect descriptions. They've used a Radial Basis function of the Neural Network to classify the defect reports. Based on tangible evidence, they've established that the model predicted high severity defects with significant accuracy and efficiency [5]. Sari and Siahaan have also leveraged Term Frequency and Inverse Document Frequency (TF-IDF) to extract features from defect descriptions. They've applied the InfoGain Feature Selection technique to obtain the set of relevant features. They've built severities prediction models with the assistance of Support Vector Machine to predict severity levels of defects [6]. Sentiment Analysis of Software Engineering Tasks has tremendous potential, but pre-trained models don't accurately predict sentiments in Software Engineering Tasks. Bin Lin et al. applied Deep Learning techniques to an enormous dataset consisting of 40k manually labeled sentences, which were sourced from Stack Overflow. Despite determining all the text's sentiments, it resulted in low accuracy levels and, ultimately, poor results. In a comparison between Stanford CoreNLP and Stanford CoreNLP SO, it was determined that the Stanford CoreNLP SO was a better influence on Sentiment Analysis tools than the Stanford CoreNLP. Another conclusion reached by Bin Lin et al. was that Sentiment Analysis tools that are not specifically trained for Software Engineering data yield disappointing results on Software Engineering datasets [1].

Biswas et al. used software domain-specific Word Embed-

ding learned from Stack Overflow in an attempt to improve the performance of the predictive model. The impact on the performance of Sentiment Analysis tools using Domain-specific Word Embedding and Generic Word Embedding, trained using Google news, were compared. The conclusion reached was that the Generic Word Embedding was better than the Domain-specific Word embedding. Biswas et al. also found that oversampling or a combination of oversampling and undersampling achieves a jump in performance in the handling of compact Software Engineering datasets[2]. R Malhotra et al. have attempted to develop Software Bug Classification (SBC) models that can identify "low", "moderate," and "high" impact levels on Software Bugs. The levels were indicated based on Maintenance Effort (ME), Change Impact (CI), or a product of both. The data is sourced from the changelogs in Google's GIT Repository. Data preprocessing is performed, and the SBC models are developed using six classification techniques. The study assessed three predictors, which were obtained from text mining. After evaluation, it was found that the performance of the combined SBC model showed higher accuracy than the ME or CI SBC models. They also found that the accuracy of the "high" category was superior to that of the other categories [7].

R Malhotra et al. have worked to find out if resampling methods applied to software defect data improve performance. They have used datasets sourced from the Defect Collection and Reporting tool (DCRS) and performed data preprocessing and applied three different resampling methods, and evaluated their performance. The performance of the developed models is evaluated using seven performance measures, accuracy, precision, sensitivity, specificity, G-Mean, and AUC. They have concluded that the application of resampling methods to the maintainability prediction models can accurately predict the minority class [5]. R Malhotra et al. have also attempted to find the effects on the performance of Software Defect Prediction Models after applying resampling techniques. They have applied six oversampling and four undersampling methods to rectify the class imbalance problem. Examining the evaluators, which are: Sensitivity, GMean, Balance, and AUC values, it was found that there was an evident improvement in the values of the evaluators when data resampling methods were applied to the Software Defect Prediction models [8].

Dr. Lov Kumar et al. have worked to automate the process of determining the severity level of a defect in the software. Defect descriptions in the form of text have been tokenized using seven different word embedding techniques. The obtained features are further pruned to achieve an optimal set of relevant features using three different Feature Selection techniques. These features, plagued by the Class Imbalance Problem, have been rid of it by using the Synthetic Minority Oversampling Technique (SMOTE). The performance of the Word Embedding is evaluated using eleven different classifiers. Dr. Lov Kumar et al. have successfully used Word Embeddings, Feature Selection, and Synthetic Minority Oversampling Technique (SMOTE) to assemble a predictive model capable of assigning a severity level to defect descriptions [9]. SentiStrength-SE, which was proposed by Islam et al., achieved 73.85% precision and 85% recall. They call attention to issues commonly faced by Sentiment Analysis Tools, some of which are: Domain-specific meaning of words, Context-sensitive variations in meanings of words, Difficulties in dealing with negation, Sentimental words in copy-pasted content,

Difficulty in dealing with irony and sarcasm, and Wrong detection of proper nouns [3].

### III. STUDY DESIGN

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

#### A. Experimental Dataset

The study uses three different experiential datasets to validate our proposed framework. These datasets are used by many software researchers for sentiment analysis [1][2]. The primary objective is to explore different types of embedding, feature selection, data sampling, and different variants of deep-learning on these datasets to predict the sentiments of software engineers. Figure 2 shows the number of positive and negative sentiments for the considered datasets. From Figure 2, we observed that the number of positive sentiments for stack overflow is much higher than negative sentiments. The unequal distribution of data leads to a class imbalance problem.

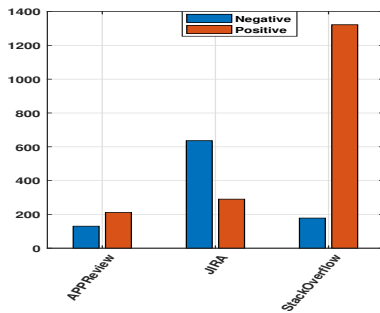


Fig. 2: Data-Sets

#### B. Training of Models from Imbalanced Data Set:

After analysis of the data, it becomes quite evident that the data is suffering from a class imbalance problem, i.e., the number of samples in each class is not the same. So, the balancing of data is required to improve the predictive ability of the developed Sentiment Analysis Models. We have performed Synthetic Minority Oversampling Technique (SMOTE) and Borderline Synthetic Minority Oversampling Technique (Borderline-SMOTE) on each dataset to balance the data [10].

#### C. Word Embedding:

The textual data of the dataset is to be expressed as vectors in relation to each other. Six different word embedding techniques including Term Frequency and Inverse Document Frequency (TF-IDF), Continuous Bag of Words (CBOW), Skip-Gram (SKG), Global Vectors for Word Representation (GLOVE), Google news Word to Vector (GW2V), fasttext (FST) have been applied on the dataset. These techniques were used to represent the textual data as a vector in an n-dimensional space. We have also removed any and all stopwords, bad symbols, and spaces before applying the word embedding techniques. These will now be used to develop models to determine the sentiment of Software Engineering Tasks [9][2].

#### D. Feature Selection Techniques

The features vectors extracted from word-embedding are used as an input, so, the performance also depends upon the optimization of important features. To extract the important features from the existing set of vectors, we have used six different Feature Selection Techniques such as: Analysis of Variance (ANOVA) is used to find feature having capability to differentiate positive and negative sentiment, correlation attribute evaluation (CORR\_ATR) is used to remove highly correlated features, Principal Components Analysis (PCA) is used to find new value of uncorrelated features, Gain ratio, information gain, and OneR are used to rank the features and select best features for sentiment analysis [4][11].

#### E. Classification Technique:

In this study, we have used eight deep learning models, which use K-Fold Cross-Validation with a k value of 5. We have separated the data into training and testing data subsets. An input layer with a number of neurons equal in quantity to the number of features of the input data is present in every single deep learning model. The models are all constituted of Dense and Dropout layers. The Dense layer's neurons receive inputs from all the neurons present in the previous layer. The Dropout layer's neurons are randomly selected. The dropout value used in this study is 0.2. The output layer has a single neuron that corresponds to the binary classification of either functional or non-functional requirements, and it uses a sigmoid activation function, unlike the other layers, which use the

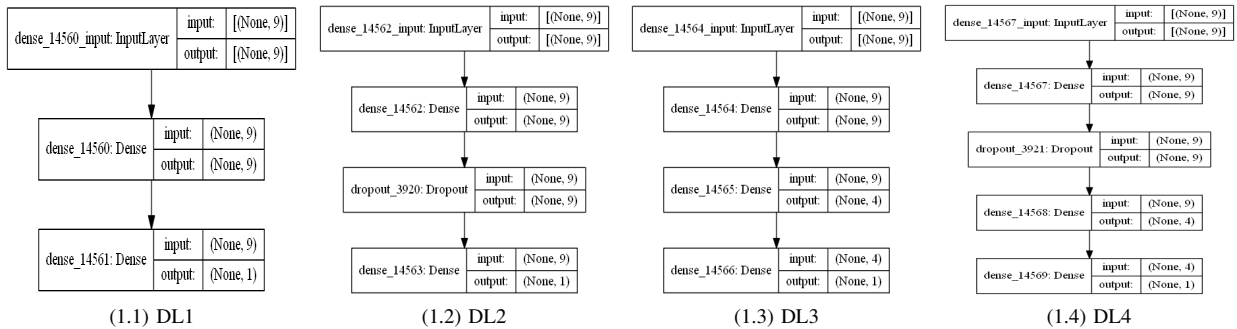


Fig. 1: Deep Learning Architecture

Rectified Linear Activation function (ReLU) as the activation function. Adam is the optimizer used to train the models, with the loss function being the Binary Cross entropy. The number of hidden layers is increased for the other four models. Figure 1 demonstrates the architectures of the models (DL1, DL2, DL3, and DL4). The parameters used to validate the models are: batch size = 30, Dropout = 0.2, and epochs = 100.

#### IV. RESEARCH METHODOLOGY

The pictorial representation of the proposed framework is provided in Figure 3. We first extract the textual documents from three different SE repositories, i.e., issue trackers, i.e., JIRA issue comments, Stack Overflow discussions contain questions and answers and user reviews on mobile apps using app stores so that their corresponding data may be analyzed. After finding these textual documents, six different types of word-embedding techniques have been used to represent text documents as numerical vectors. Each embedding uses a different way to represent words for text documents with a real-valued vector. The values of these vectors are closer in the vector space for similar words. Next, we have used two different types of sampling techniques, such as: SMOTE and BLSMOTE to handle the class imbalanced nature of datasets. In the next step, we have applied different feature selection techniques to select the best combination of relevant features. The ANOVA test is used to remove insignificant features, PCA is used to remove high correlation between features and find new values of features, gain ratio, information gain, and OneR is used to rank features and select the top best features, and finally, correlation analysis is used to remove highly correlated features. After finding the right sets of features, we have used different variants of deep-learning techniques to train software sentiment, and analysis models. The trained models are validated with a 5-fold cross-validation method, and the performance of these models is compared with the help of four different performance parameters: accuracy, precision, recall, and AUC.

#### V. EMPIRICAL RESULTS AND ANALYSIS

The primary objective of this work is to analyze the performance of the developed software sentiment analysis models using different variants of deep-learning, word-embedding techniques, features selection techniques, and data sampling techniques with the purpose of investigating how different contexts can impact their effectiveness. The proposed models are validated with three software-related datasets, namely mobile app reviews, Stack Overflow discussions, and JIRA issue comments. Finally, the predictive ability of these models is evaluated using different performance parameters such as accuracy, AUC, precision, and recall. AUC is considered the primary parameter for the model's performance because of its capability to provide good findings in case of imbalanced nature of data. Tables I and II show the performance of models in terms of Precision, Recall, accuracy, and AUC for the AppReview dataset using different variants of deep-learning and feature selection techniques with original data and sampled data. The results for other combinations are similar. The high value of AUC ( $\geq 0.7$ ) in Tables I and II suggested that the proposed models have the capability to predict the current state of sentiment analysis for software engineering. In the majority of cases, the precision, recall, and AUC values are higher than 0.8. Also, the information present in Tables I and II suggest that the models trained on sampled data have a better ability to predict sentiment as compared to original data. Another finding from Tables I and II is that the models trained on selected sets of features have a high value of precision, recall, and AUC as compared to all features.

#### VI. COMPARATIVE ANALYSIS

In this section, we have compared the performance of different word embedding techniques, class balancing approaches, feature selection strategies, and deep-learning techniques, which are used for developing sentiment prediction models using Box-plot diagrams and descriptive statistics. We have also performed the Friedman test to find statistical significance differences between different techniques. The hypothesis used to achieve our objective is mentioned below:

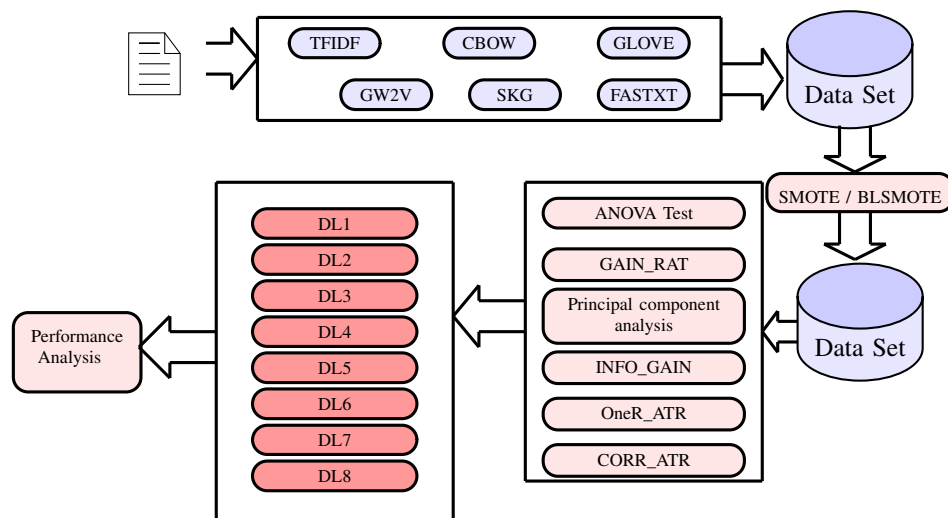


Fig. 3: Framework of proposed work

TABLE I: AppReviews :Precision and Recall

	Precision								Recall							
	DL1	DL2	DL3	DL4	DL5	DL6	DL7	DL8	DL1	DL2	DL3	DL4	DL5	DL6	DL7	DL8
<b>ORGDATA(OD)</b>																
TFIDF	0.85	0.84	0.84	0.84	0.85	0.83	0.83	0.84	0.90	0.87	0.85	0.87	0.87	0.87	0.88	0.88
CBOW	0.64	0.72	0.76	0.68	0.67	0.76	0.64	0.64	0.95	0.86	0.63	0.80	0.90	0.81	0.92	0.98
SKG	0.81	0.82	0.83	0.8	0.83	0.85	0.82	0.81	0.7	0.86	0.81	0.86	0.87	0.75	0.77	0.83
GLOVE	0.86	0.86	0.85	0.85	0.88	0.84	0.85	0.87	0.88	0.85	0.86	0.88	0.87	0.84	0.88	0.84
GW2V	0.84	0.83	0.85	0.85	0.87	0.85	0.86	0.85	0.87	0.88	0.87	0.89	0.79	0.86	0.88	0.85
FASTXT	0.74	0.77	0.77	0.76	0.73	0.73	0.73	0.76	0.73	0.68	0.73	0.69	0.82	0.72	0.66	0.69
<b>ORGDATA(ANOVA)</b>																
TFIDF	0.87	0.87	0.86	0.87	0.87	0.84	0.87	0.84	0.89	0.91	0.91	0.91	0.88	0.92	0.89	0.91
CBOW	0.62	0.62	0.71	0.68	0.66	0.66	0.62	0.64	1.00	1.00	0.88	0.89	0.96	0.90	1.00	0.98
SKG	0.81	0.83	0.8	0.86	0.85	0.85	0.82	0.83	0.82	0.83	0.82	0.68	0.83	0.80	0.83	0.83
GLOVE	0.87	0.85	0.84	0.87	0.84	0.88	0.88	0.88	0.84	0.84	0.86	0.83	0.85	0.83	0.8	0.83
GW2V	0.89	0.86	0.87	0.86	0.89	0.87	0.84	0.87	0.86	0.89	0.88	0.85	0.88	0.87	0.87	0.88
FASTXT	0.74	0.76	0.76	0.76	0.76	0.75	0.76	0.75	0.84	0.86	0.77	0.82	0.80	0.80	0.75	0.72
<b>ORGDATA(OneR_ATR)</b>																
TFIDF	0.82	0.82	0.83	0.83	0.78	0.73	0.72	0.62	0.93	0.93	0.93	0.94	0.97	0.97	0.97	1.00
CBOW	0.62	0.62	0.64	0.66	0.64	0.70	0.62	0.62	1.00	1.00	0.99	0.96	0.98	0.91	1.00	1.00
SKG	0.71	0.71	0.71	0.70	0.73	0.73	0.62	0.68	0.84	0.91	0.87	0.87	0.82	0.84	1.00	0.90
GLOVE	0.84	0.82	0.83	0.82	0.83	0.74	0.80	0.64	0.85	0.85	0.82	0.82	0.85	0.89	0.87	0.95
GW2V	0.85	0.86	0.85	0.83	0.83	0.7	0.82	0.79	0.83	0.84	0.84	0.86	0.87	0.91	0.84	0.87
FASTXT	0.63	0.69	0.71	0.69	0.68	0.67	0.63	0.64	0.88	0.82	0.81	0.78	0.77	0.83	0.99	0.81
<b>SMOTE(OD)</b>																
TFIDF	0.87	0.92	0.90	0.92	0.92	0.90	0.92	0.91	0.94	0.94	0.93	0.94	0.94	0.94	0.93	0.94
CBOW	0.66	0.68	0.78	0.71	0.68	0.81	0.67	0.73	0.8	0.98	0.92	0.97	0.98	0.90	1.00	0.96
SKG	0.81	0.86	0.89	0.84	0.84	0.91	0.86	0.84	0.66	0.94	0.90	0.91	0.91	0.85	0.91	0.91
GLOVE	0.88	0.92	0.93	0.93	0.93	0.93	0.92	0.93	0.87	0.94	0.90	0.91	0.94	0.93	0.90	0.92
GW2V	0.94	0.95	0.93	0.92	0.93	0.94	0.93	0.94	0.96	0.94	0.95	0.92	0.95	0.95	0.95	0.95
FASTXT	0.79	0.77	0.81	0.75	0.77	0.78	0.73	0.77	0.87	0.87	0.85	0.90	0.89	0.90	0.92	0.85
<b>SMOTE(ANOVA)</b>																
TFIDF	0.88	0.92	0.89	0.91	0.90	0.89	0.92	0.89	0.97	0.96	0.97	0.96	0.97	0.97	0.97	0.96
CBOW	0.67	0.67	0.68	0.67	0.67	0.67	0.67	0.68	1.00	1.00	0.94	1.00	1.00	0.99	1.00	0.95
SKG	0.78	0.85	0.90	0.85	0.85	0.88	0.82	0.87	0.95	0.90	0.88	0.92	0.91	0.90	0.92	0.90
GLOVE	0.87	0.93	0.93	0.93	0.92	0.93	0.92	0.94	0.94	0.94	0.93	0.92	0.92	0.92	0.95	0.94
GW2V	0.92	0.92	0.93	0.93	0.92	0.94	0.93	0.94	0.96	0.95	0.96	0.95	0.95	0.95	0.93	0.96
FASTXT	0.79	0.78	0.81	0.78	0.80	0.83	0.83	0.84	0.84	0.92	0.87	0.92	0.91	0.88	0.89	0.87
<b>SMOTE(OneR_ATR)</b>																
TFIDF	0.83	0.84	0.85	0.83	0.83	0.73	0.77	0.72	0.94	0.96	0.95	0.94	0.94	0.97	0.96	0.97
CBOW	0.67	0.67	0.68	0.69	0.73	0.67	0.67	0.68	1.00	0.99	0.95	0.97	0.95	1.00	1.00	0.96
SKG	0.70	0.70	0.71	0.70	0.68	0.69	0.67	0.68	0.97	0.98	0.95	0.97	0.98	0.98	1.00	0.93
GLOVE	0.80	0.79	0.83	0.83	0.79	0.71	0.7	0.72	0.89	0.90	0.86	0.89	0.90	0.95	0.99	0.97
GW2V	0.78	0.79	0.79	0.80	0.80	0.83	0.79	0.74	0.88	0.88	0.86	0.90	0.90	0.88	0.88	0.90
FASTXT	0.68	0.71	0.74	0.72	0.69	0.75	0.67	0.73	0.98	0.97	0.87	0.95	0.98	0.87	0.99	0.90
<b>BLSMOTE(OD)</b>																
TFIDF	0.89	0.92	0.92	0.94	0.93	0.92	0.93	0.92	0.92	0.92	0.91	0.93	0.93	0.9	0.94	0.92
CBOW	0.66	0.67	0.80	0.73	0.67	0.76	0.67	0.76	0.8	0.99	0.92	0.96	1.00	0.93	1.00	0.95
SKG	0.79	0.80	0.87	0.83	0.87	0.88	0.82	0.86	0.75	0.94	0.92	0.91	0.84	0.90	0.84	0.90
GLOVE	0.9	0.93	0.92	0.93	0.93	0.93	0.91	0.92	0.93	0.93	0.92	0.94	0.92	0.93	0.94	0.93
GW2V	0.95	0.95	0.95	0.89	0.96	0.95	0.87	0.95	0.95	0.94	0.95	0.95	0.94	0.94	0.95	0.94
FASTXT	0.77	0.74	0.76	0.72	0.73	0.75	0.69	0.73	0.87	0.90	0.85	0.84	0.83	0.84	0.91	0.87
<b>BLSMOTE(ANOVA)</b>																
TFIDF	0.87	0.92	0.89	0.91	0.91	0.89	0.91	0.90	0.94	0.95	0.95	0.95	0.95	0.95	0.95	0.96
CBOW	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.69	1.00	1.00	0.95	1.00	1.00	0.98	1.00	0.98
SKG	0.74	0.83	0.89	0.83	0.79	0.85	0.83	0.90	0.97	0.90	0.90	0.89	0.92	0.91	0.88	0.87
GLOVE	0.87	0.93	0.95	0.95	0.93	0.95	0.92	0.92	0.94	0.94	0.92	0.93	0.92	0.91	0.94	0.93
GW2V	0.94	0.95	0.95	0.94	0.94	0.96	0.93	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.96	0.95
FASTXT	0.74	0.74	0.83	0.78	0.77	0.86	0.73	0.81	0.90	0.92	0.83	0.85	0.91	0.78	0.94	0.86
<b>BLSMOTE(OneR_ATR)</b>																
TFIDF	0.80	0.81	0.79	0.81	0.81	0.75	0.76	0.73	0.94	0.95	0.95	0.95	0.95	0.97	0.98	0.98
CBOW	0.67	0.67	0.73	0.70	0.67	0.68	0.67	0.67	1.00	0.99	0.94	0.97	1.00	0.99	1.00	0.99
SKG	0.70	0.69	0.69	0.68	0.69	0.69	0.67	0.68	0.95	0.97	0.95	0.97	0.95	0.97	1.00	0.98
GLOVE	0.77	0.75	0.80	0.76	0.74	0.70	0.69	0.75	0.92	0.96	0.9	0.95	0.97	0.95	0.98	0.91
GW2V	0.74	0.76	0.75	0.78	0.76	0.78	0.73	0.77	0.89	0.88	0.89	0.90	0.90	0.90	0.94	0.87
FASTXT	0.67	0.67	0.70	0.70	0.67	0.69	0.67	0.68	0.98	0.98	0.95	0.94	0.99	0.96	0.99	0.98

- Word-Embedding Techniques: Null Hypothesis:** There is no significant difference between the models trained by using features extracted by different embedding techniques.
- Feature Selection Techniques: Null Hypothesis:** There is no significant difference between the models trained by using selected sets of features using different feature selection techniques and all features.
- Data Sampling Techniques: Null Hypothesis:** There is no significant difference between the models trained on sampled data and original data.
- Deep-Learning Techniques: Null Hypothesis:** There is no significant difference between the models trained using different variants of deep-learning techniques.

TABLE II: AppReviews: Accuracy and AUC

	Accuracy								AUC							
	DL1	DL2	DL3	DL4	DL5	DL6	DL7	DL8	DL1	DL2	DL3	DL4	DL5	DL6	DL7	DL8
<b>ORGDATA(OD)</b>																
TFIDF	83.87	81.52	80.65	82.11	82.11	80.94	81.82	82.4	0.89	0.85	0.85	0.85	0.84	0.83	0.84	0.82
CBOW	64.22	70.67	64.81	64.52	65.69	72.43	62.46	64.52	0.63	0.7	0.73	0.69	0.71	0.78	0.64	0.67
SKG	71.55	79.77	77.42	77.71	80.65	76.54	75.07	77.13	0.81	0.87	0.85	0.85	0.88	0.86	0.83	0.86
GLOVE	83.58	81.82	82.11	82.7	84.16	80.65	83.28	82.4	0.91	0.9	0.89	0.9	0.91	0.89	0.9	0.89
GW2V	81.82	81.52	82.7	83.58	79.77	82.4	83.28	81.52	0.91	0.91	0.89	0.91	0.9	0.87	0.91	0.87
FASTXT	67.45	67.45	69.21	67.16	69.5	66.28	63.64	66.86	0.76	0.77	0.75	0.75	0.74	0.74	0.71	0.77
<b>ORGDATA(ANOVA)</b>																
TFIDF	84.46	86.22	85.63	85.92	84.75	84.75	85.04	84.16	0.91	0.9	0.87	0.89	0.88	0.85	0.88	0.85
CBOW	61.88	62.46	70.38	67.74	67.45	64.52	62.17	64.52	0.59	0.75	0.73	0.74	0.74	0.65	0.69	0.52
SKG	77.13	78.59	75.95	73.61	80.35	78.59	78.01	78.59	0.84	0.86	0.84	0.85	0.85	0.86	0.84	0.85
GLOVE	82.4	81.23	81.52	81.82	80.35	82.7	80.65	82.4	0.89	0.88	0.88	0.88	0.88	0.89	0.88	0.88
GW2V	84.46	84.16	84.16	81.82	85.92	84.16	81.52	84.46	0.92	0.92	0.9	0.91	0.92	0.9	0.9	0.89
FASTXT	71.85	75.07	70.97	72.73	71.85	71.55	70.09	67.74	0.73	0.82	0.78	0.79	0.79	0.77	0.78	0.75
<b>ORGDATA(OneR_ATR)</b>																
TFIDF	83.58	83.58	84.16	84.75	80.94	76.25	75.07	61.88	0.85	0.84	0.83	0.84	0.81	0.77	0.76	0.47
CBOW	61.88	61.58	64.81	67.45	64.22	70.38	61.88	61.88	0.68	0.79	0.71	0.78	0.77	0.78	0.75	0.48
SKG	68.92	71.26	69.5	68.62	70.38	70.97	61.88	67.45	0.74	0.75	0.72	0.71	0.73	0.76	0.6	0.67
GLOVE	80.65	79.47	78.59	78.01	79.77	73.61	78.3	64.52	0.88	0.88	0.86	0.88	0.85	0.77	0.85	0.55
GW2V	80.65	81.52	80.65	80.65	80.65	70.67	78.3	77.71	0.88	0.88	0.86	0.87	0.87	0.75	0.85	0.85
FASTXT	61	65.98	68.04	64.52	63.64	63.93	63.64	59.53	0.64	0.71	0.71	0.69	0.69	0.64	0.69	0.54
<b>SMOTE(OD)</b>																
TFIDF	86.2	90.14	88.17	90.14	90.86	89.25	90.14	89.43	0.92	0.92	0.89	0.92	0.91	0.9	0.92	0.9
CBOW	58.96	68.28	77.6	71.68	67.74	79.21	66.67	73.66	0.62	0.74	0.79	0.82	0.74	0.81	0.66	0.81
SKG	66.67	85.3	85.66	82.62	82.62	84.41	83.69	82.26	0.77	0.9	0.91	0.87	0.88	0.91	0.89	0.86
GLOVE	83.51	90.68	89.25	89.43	91.4	90.86	87.81	90.14	0.9	0.94	0.95	0.94	0.96	0.93	0.94	0.93
GW2V	92.83	92.47	91.94	89.96	91.58	92.47	92.11	92.29	0.97	0.97	0.95	0.95	0.94	0.94	0.94	0.94
FASTXT	75.81	74.01	76.52	73.12	74.55	76.34	72.4	73.3	0.81	0.77	0.82	0.76	0.77	0.79	0.73	0.78
<b>SMOTE(ANOVA)</b>																
TFIDF	88.53	91.22	90.14	91.04	91.04	90.5	92.11	88.89	0.92	0.94	0.91	0.94	0.94	0.91	0.94	0.9
CBOW	66.67	66.67	67.03	66.67	66.67	66.31	66.67	67.38	0.57	0.65	0.68	0.68	0.7	0.57	0.67	0.59
SKG	78.49	82.8	85.13	83.69	82.97	85.66	81.36	84.77	0.84	0.88	0.92	0.88	0.88	0.9	0.86	0.9
GLOVE	86.38	91.4	90.86	90.14	89.07	90.68	91.22	92.11	0.92	0.95	0.95	0.94	0.94	0.95	0.96	0.95
GW2V	91.94	91.58	92.65	92.29	91.76	92.83	90.86	93.01	0.97	0.97	0.96	0.97	0.96	0.97	0.96	0.94
FASTXT	73.84	77.42	77.96	77.06	78.32	79.39	80.29	80.11	0.73	0.79	0.83	0.81	0.81	0.81	0.82	0.83
<b>SMOTE(OneR_ATR)</b>																
TFIDF	82.8	84.95	84.95	83.69	83.15	74.01	78.67	72.76	0.83	0.82	0.84	0.83	0.82	0.7	0.79	0.69
CBOW	66.67	66.67	67.03	68.28	72.94	66.67	66.67	66.85	0.49	0.63	0.75	0.78	0.78	0.49	0.4	0.5
SKG	70.07	70.43	70.79	69.89	67.74	69.89	66.67	65.77	0.69	0.71	0.7	0.71	0.71	0.61	0.69	0.55
GLOVE	77.42	77.6	78.49	80.29	77.42	70.43	70.43	73.48	0.83	0.83	0.84	0.85	0.83	0.62	0.59	0.69
GW2V	75.63	76.16	75.81	78.49	78.67	79.57	76.52	72.4	0.81	0.83	0.81	0.83	0.83	0.84	0.82	0.74
FASTXT	68.1	70.97	71.15	72.58	69.71	72.22	67.2	70.97	0.64	0.7	0.71	0.72	0.67	0.72	0.69	0.69
<b>BLSMOTE(OD)</b>																
TFIDF	87.28	89.78	88.17	91.22	90.86	88.53	91.22	89.25	0.92	0.92	0.91	0.93	0.93	0.91	0.92	0.91
CBOW	58.6	67.56	79.21	73.12	67.03	75.45	66.67	76.52	0.6	0.72	0.82	0.77	0.71	0.77	0.54	0.8
SKG	70.07	80.65	85.3	81.9	81.54	84.95	76.7	83.69	0.78	0.9	0.91	0.88	0.88	0.9	0.86	0.89
GLOVE	87.99	90.68	89.25	91.22	90.14	90.5	89.78	89.96	0.93	0.94	0.93	0.94	0.95	0.94	0.92	0.93
GW2V	93.55	92.83	93.55	89.25	93.01	92.65	87.28	92.83	0.97	0.97	0.97	0.94	0.97	0.96	0.92	0.95
FASTXT	73.84	71.51	72.4	67.74	68.64	70.79	67.03	70.25	0.77	0.74	0.75	0.73	0.71	0.74	0.69	0.74
<b>BLSMOTE(ANOVA)</b>																
TFIDF	86.92	91.22	89.25	90.68	90.86	89.25	91.04	90.14	0.91	0.93	0.9	0.93	0.92	0.89	0.92	0.91
CBOW	66.67	66.49	66.31	66.85	66.67	67.03	66.67	69	0.58	0.67	0.67	0.68	0.64	0.56	0.65	0.66
SKG	75.63	81.36	86.02	80.65	78.49	83.87	79.93	84.59	0.84	0.88	0.91	0.88	0.87	0.89	0.85	0.91
GLOVE	86.92	90.86	91.04	92.47	90.5	90.68	90.86	89.78	0.93	0.95	0.94	0.95	0.94	0.93	0.94	0.94
GW2V	92.47	93.19	93.73	92.47	92.65	93.91	92.83	93.55	0.97	0.97	0.97	0.96	0.96	0.97	0.97	0.95
FASTXT	72.22	73.3	77.6	74.55	76.16	76.88	73.12	77.06	0.73	0.8	0.83	0.81	0.79	0.82	0.8	0.81
<b>BLSMOTE(OneR_ATR)</b>																
TFIDF	80.65	82.26	79.75	82.08	81.72	76.7	78.32	74.37	0.75	0.75	0.75	0.76	0.76	0.72	0.76	0.63
CBOW	66.67	66.31	73.12	70.25	66.67	68.82	66.67	67.38	0.53	0.64	0.71	0.75	0.72	0.53	0.58	0.54
SKG	69.18	69	68.64	67.74	68.82	68.82	66.67	67.56	0.64	0.64	0.65	0.62	0.71	0.63	0.56	0.55
GLOVE	76.7	75.63	78.67	76.88	75.27	69	69.35	73.84	0.79	0.79	0.8	0.82	0.8	0.6	0.74	0.74
GW2V	71.86	74.01	72.94	75.99	74.73	76.34	73.12	74.19	0.78	0.79	0.78	0.8	0.8	0.8	0.77	0.77
FASTXT	67.2	66.67	70.07	68.64	66.49	68.28	66.49	68.28	0.58	0.66	0.67	0.66	0.67	0.57	0.58	0.51

### A. Word-Embedding

In this work, six different types of word embedding approaches such as TFIDF, CBOW, GLOVE, GW2V, SKG, and FASTXT have been used to find the numerical vectors of software text comments. To find the best embedding approach, we exploited performance evaluators- Accuracy, Precision, AUC, and Recall, which are computed for models trained

by taking the above embedding techniques as input and trained using different variants of deep-learning with 5-fold cross-validation techniques on sampled as well as original data. Figure 4 visually depicts the model's ability to predict sentiments using different word-embedding techniques, and Table III depicts descriptive statistics of different embedding in terms of accuracy, AUC, precision, and Recall. From Figure 4, it is visible that the models trained by taking numerical

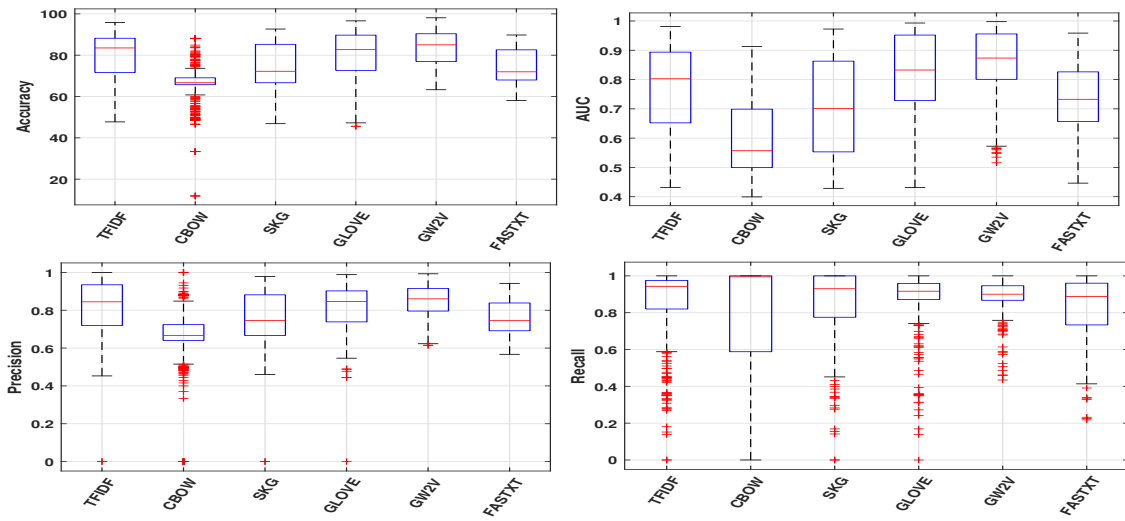


Fig. 4: Performance Box-Plot Diagram: Performance of Different Word Embedding

vectors computed using google word to vector (GW2V) have better capability to predict sentiment as compared to other embeddings. The models trained using GW2V achieved 0.86 average AUC, 0.89 average Recall, 0.85 average precision, and 84.03 average accuracy. Similarly, From Table III, we observed that the models trained using CBOW have the worst performance with 0.60 average AUC, 0.78 average Recall, 0.66 average precision, and 67.49 average accuracy.

TABLE III: Descriptive Statistics: Different Word Embedding

	Min	Max	Mean	Q2	Q1	Q3
<b>Accuracy</b>						
TFIDF	47.72	95.83	80.39	83.51	71.58	88.20
CBOW	11.87	88.13	67.49	66.67	65.73	69.05
SKG	46.86	92.66	75.30	72.22	66.67	85.26
GLOVE	45.60	96.62	81.46	82.70	72.63	89.71
GW2V	63.29	98.11	84.03	85.02	76.92	90.38
FASTXT	58.06	89.78	74.66	71.96	68.00	82.59
<b>Precision</b>						
TFIDF	0.00	1.00	0.82	0.84	0.72	0.93
CBOW	0.00	1.00	0.66	0.67	0.64	0.72
SKG	0.00	0.98	0.77	0.75	0.67	0.88
GLOVE	0.00	0.99	0.82	0.85	0.74	0.90
GW2V	0.61	0.99	0.85	0.86	0.80	0.91
FASTXT	0.57	0.94	0.76	0.74	0.69	0.84
<b>Recall</b>						
TFIDF	0.00	1.00	0.87	0.94	0.82	0.98
CBOW	0.00	1.00	0.78	0.99	0.59	1.00
SKG	0.00	1.00	0.87	0.93	0.78	1.00
GLOVE	0.00	1.00	0.89	0.92	0.87	0.96
GW2V	0.43	1.00	0.89	0.90	0.87	0.95
FASTXT	0.22	1.00	0.84	0.89	0.73	0.96
<b>AUC</b>						
TFIDF	0.43	0.98	0.78	0.80	0.65	0.89
CBOW	0.40	0.91	0.60	0.56	0.50	0.70
SKG	0.43	0.97	0.70	0.70	0.55	0.86
GLOVE	0.43	0.99	0.82	0.83	0.73	0.95
GW2V	0.52	1.00	0.86	0.87	0.80	0.96
FASTXT	0.45	0.96	0.74	0.73	0.66	0.83

Table IV shows the mean ranks using the Freidman test for the various word embedding techniques. We have evaluated

the considered null hypothesis at 0.05 with five degrees of freedom on four different performance parameters such as accuracy, recall, precision, and AUC. The lower value of mean rank represents the best word-embedding techniques for sentiment analysis of software engineering comments. According to information present in Table IV, the models trained using different embedding techniques are significantly different. Similarly, according to information present in Table IV, the models trained using GW2V have a lower mean rank, i.e., 1.91 for accuracy, 1.92 precision, 3.61 recall, and 1.37 AUC representing that the developed models have better prediction capability as compared to other embedding techniques.

*B. Feature Selection*

In this work, six different types of features selection techniques: significant features calculation using ANOVA test, un-correlated sets of features using PCA, best sets of features using the gain ratio(GAIN\_RAT), information gain(INFO\_GAIN), oneR attribute evaluation (OneR\_ATR), correlation attribute selection (CORR\_ATR) have been used to find the best combination of relevant features for software engineering sentiment analysis. We exploited performance evaluators- Accuracy, Precision, AUC, and Recall to find the best feature selection technique that gives us the best sets of features for models trained using different variants of deep-learning with 5-fold cross-validation techniques on sampled as well as original data. Figure 5 visually depicts the model’s ability to predict sentiments using different feature selection techniques and Table V depicts descriptive statistics of different feature selection techniques in terms of accuracy, AUC, precision, and Recall. From Figure 4, it is quite evident that the models trained by taking significant sets of features using the ANOVA test have better capability to predict sentiment as compared to other feature selection techniques. The models trained using ANOVA features achieved 0.85 average AUC, 0.88 average recall, 0.84 average precision, and 83.21 average accuracy. Similarly, From Table V, we observed that the models trained using features selection from OneR\_ATR have

TABLE IV: Friedman test : Mean Rank

	Accuracy	AUC	Precision	Recall
<b>DL</b>				
DL1	4.69	4.48	4.60	4.80
DL2	3.83	3.20	4.03	4.27
DL3	3.49	3.15	3.39	5.17
DL4	3.46	3.07	3.68	4.69
DL5	4.12	3.70	4.32	4.04
DL6	5.01	5.62	4.79	4.78
DL7	5.62	6.03	5.67	3.65
DL8	5.78	6.75	5.51	4.60
	$P < 0.05$	$P < 0.05$	$P < 0.05$	$P < 0.05$
<b>Word-Embedding</b>				
TFIDF	2.74	3.25	2.43	3.52
CBOW	5.27	5.58	5.39	3.07
SKG	4.17	4.49	4.16	3.15
GLOVE	2.60	2.23	2.74	3.46
GW2V	1.91	1.37	1.92	3.61
FASTXT	4.32	4.07	4.37	4.19
	$P < 0.05$	$P < 0.05$	$P < 0.05$	$P < 0.05$
<b>Feature Sets</b>				
OD	2.44	2.09	2.39	4.02
ANOVA	2.27	1.72	2.21	3.77
PCA	4.49	4.72	4.74	3.59
GAIN_RAT	4.83	4.93	4.80	4.10
INFO_GAIN	4.56	4.46	4.48	4.15
OneR_ATR	4.91	5.35	4.90	4.10
CORR_ATR	4.50	4.73	4.48	4.28
	$P < 0.05$	$P < 0.05$	$P < 0.05$	$P < 0.05$
<b>OD and SMOTE</b>				
ORGDATA	1.69	2.28	1.83	2.34
SMOTE	2.02	1.73	1.88	1.87
BLSMOTE	2.29	1.98	2.29	1.79
	$P < 0.05$	$P < 0.05$	$P < 0.05$	$P < 0.05$

the worst performance with 0.70 average AUC, 0.84 average Recall, 0.75 average precision, and 74.39 average accuracy.

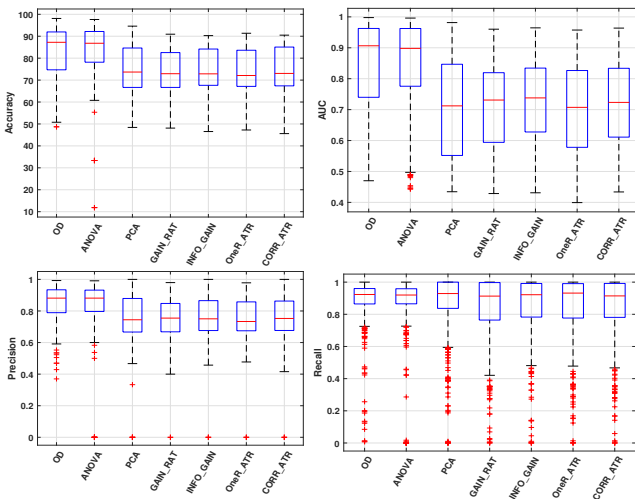


Fig. 5: Performance Box-Plot Diagram: Performance of Different Sets of Features

Table IV shows the mean ranks using the Friedman test for the various feature selection techniques. We have evaluated the considered null hypothesis at 0.05 with six degrees of freedom on four different performance parameters such as accuracy, recall, precision, and AUC. According to information present in Table IV, the models trained using different feature selection techniques are significantly different. Similarly, information

TABLE V: Descriptive Statistics: Different Sets of Features

	Min	Max	Mean	Q2	Q1	Q3
<b>Accuracy</b>						
OD	48.66	98.11	83.40	87.24	74.69	91.96
ANOVA	11.87	97.64	83.21	86.80	78.16	92.17
PCA	48.43	94.60	74.83	73.67	66.67	84.63
GAIN_RAT	48.11	90.93	74.51	72.91	66.67	82.54
INFO_GAIN	46.54	90.28	75.08	72.85	67.62	84.16
OneR_ATR	47.25	91.36	74.39	72.10	67.14	83.61
CORR_ATR	45.60	90.50	75.12	73.05	67.37	85.06
<b>Precision</b>						
OD	0.37	0.99	0.85	0.88	0.79	0.93
ANOVA	0.00	0.99	0.84	0.88	0.80	0.93
PCA	0.00	1.00	0.75	0.74	0.67	0.88
GAIN_RAT	0.00	0.98	0.76	0.75	0.67	0.85
INFO_GAIN	0.00	1.00	0.76	0.75	0.68	0.87
OneR_ATR	0.00	0.98	0.75	0.73	0.67	0.86
CORR_ATR	0.00	1.00	0.76	0.75	0.68	0.86
<b>Recall</b>						
OD	0.01	1.00	0.88	0.92	0.86	0.96
ANOVA	0.00	1.00	0.88	0.92	0.87	0.96
PCA	0.00	1.00	0.86	0.93	0.84	1.00
GAIN_RAT	0.00	1.00	0.84	0.91	0.76	1.00
INFO_GAIN	0.00	1.00	0.85	0.92	0.78	0.99
OneR_ATR	0.00	1.00	0.84	0.93	0.78	0.99
CORR_ATR	0.00	1.00	0.84	0.91	0.78	0.99
<b>AUC</b>						
OD	0.47	1.00	0.83	0.91	0.74	0.96
ANOVA	0.44	1.00	0.85	0.90	0.78	0.96
PCA	0.43	0.98	0.71	0.71	0.55	0.85
GAIN_RAT	0.43	0.96	0.71	0.73	0.59	0.82
INFO_GAIN	0.43	0.96	0.73	0.74	0.63	0.83
OneR_ATR	0.40	0.96	0.70	0.71	0.58	0.83
CORR_ATR	0.43	0.96	0.72	0.72	0.61	0.83

present in Table IV shows that the models trained by taking selected significant features using ANOVA as an input have a lower mean rank, i.e., 2.27 for accuracy, 2.21 precision, 3.77 recall, and 1.72 AUC, represent that the developed models have better prediction capability as compared to other features selection techniques.

### C. Classification Techniques

The sentiment prediction models for software engineering comments are trained using different variants of deep-learning techniques with a 5-fold cross-validation approach. These trained models' capability is compared using performance parameters such as accuracy, precision, recall, and AUC. Figure 6 depicts the box-plot diagrams of different performance parameters for the models trained using different variants of deep learning. Table VI shows the descriptive statistics in terms of Mean, Median, Min, Max, Q1, and Q3 for different deep-learning techniques. It can be seen from Figure 6 and Table VI that the models trained using DL2, DL3, DL4, and DL5 have similar average values of AUC, i.e., 0.78. Similarly, the DL8 classifier produces models with a minimum average AUC of 0.67.

In this paper, we have also compared the effectiveness of different variants of deep learning using the Friedman test with a significance level of 0.05 and six degrees of freedom on four different performance parameters such as accuracy, recall, precision, and AUC. Table IV shows the mean ranks using the Friedman test for different variants of deep learning. According to the results of the Friedman test, we observed



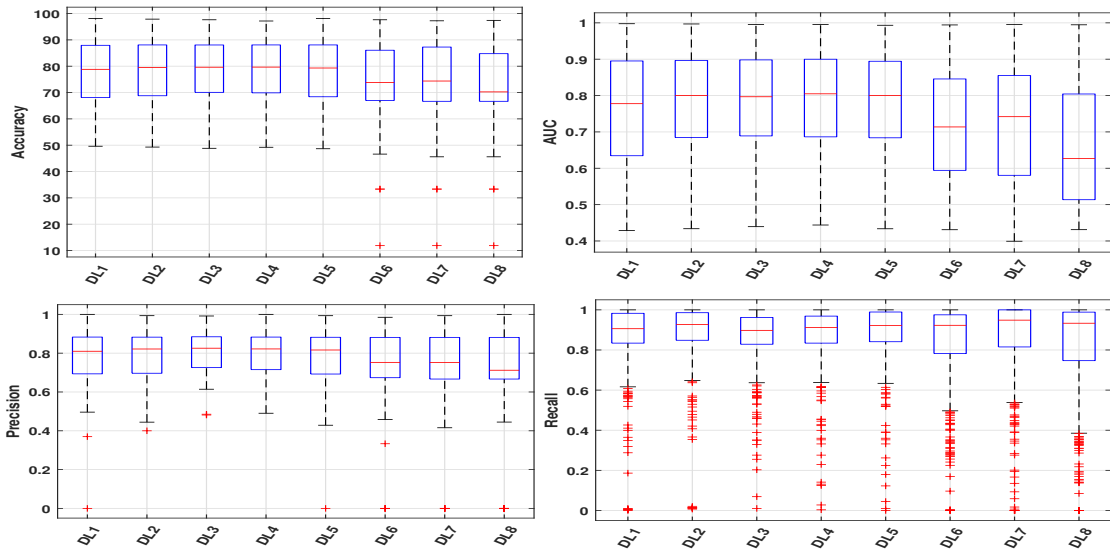


Fig. 6: Performance Box-Plot Diagram: Different Variants of Deep Learning

TABLE VI: Descriptive Statistics: Different Variants of DL

	Min	Max	Mean	Q2	Q1	Q3
<b>Accuracy</b>						
DL1	49.61	98.11	77.97	78.81	68.09	87.93
DL2	49.29	97.88	78.96	79.53	68.79	88.09
DL3	48.82	97.64	79.19	79.62	70.07	88.07
DL4	49.21	97.17	79.07	79.66	69.89	88.09
DL5	48.66	98.11	78.69	79.32	68.40	88.09
DL6	11.87	97.64	75.30	73.82	67.00	86.07
DL7	11.87	97.25	75.41	74.37	66.67	87.28
DL8	11.87	97.41	73.15	70.25	66.67	84.77
<b>Precision</b>						
DL1	0.00	1.00	0.79	0.81	0.69	0.88
DL2	0.40	0.99	0.80	0.82	0.70	0.88
DL3	0.48	0.99	0.81	0.83	0.73	0.88
DL4	0.49	1.00	0.81	0.82	0.72	0.88
DL5	0.00	0.99	0.80	0.82	0.69	0.88
DL6	0.00	0.98	0.76	0.75	0.67	0.88
DL7	0.00	0.99	0.76	0.75	0.67	0.88
DL8	0.00	1.00	0.73	0.71	0.67	0.88
<b>Recall</b>						
DL1	0.00	1.00	0.86	0.91	0.83	0.98
DL2	0.01	1.00	0.88	0.93	0.85	0.99
DL3	0.01	1.00	0.86	0.90	0.83	0.96
DL4	0.00	1.00	0.87	0.91	0.83	0.97
DL5	0.00	1.00	0.88	0.92	0.84	0.99
DL6	0.00	1.00	0.83	0.92	0.78	0.98
DL7	0.00	1.00	0.85	0.95	0.82	1.00
DL8	0.00	1.00	0.81	0.93	0.75	0.99
<b>AUC</b>						
DL1	0.43	1.00	0.76	0.78	0.63	0.90
DL2	0.43	1.00	0.78	0.80	0.68	0.90
DL3	0.44	1.00	0.78	0.80	0.69	0.90
DL4	0.44	1.00	0.78	0.80	0.69	0.90
DL5	0.43	0.99	0.78	0.80	0.68	0.89
DL6	0.43	0.99	0.72	0.71	0.59	0.85
DL7	0.40	1.00	0.73	0.74	0.58	0.86
DL8	0.43	0.99	0.67	0.63	0.51	0.80

that the performance of software sentiment prediction models significantly depends on the architecture of the deep-learning

models. Similarly, the models trained using DL4 have better capability to predict sentiment as compared to other deep-learning techniques.

#### D. SMOTE

In this study, we have used two different variants of SMOTE techniques to handle the class imbalance nature of data. We have used box-plot and descriptive statistics of performance parameters to find the impact of data sampling techniques on sentiment analysis for software engineering comments. Figure 7 presents a visual representation of the predictive capability of models trained on a balanced dataset versus models learned on an imbalanced dataset. Table VII shows the descriptive statistics in terms of min, max, Mean, Median, Q1, and Q3 for models trained on sampled data and original data. From Figure 7, and Table VI, it can be seen that the models trained on sampled data have better performance as compared to the original data. The trained prediction models on sampled data have 0.76 average AUC, 0.88 average recall, 0.78 average precision, and 76 average accuracies. While, models trained on original data have 0.73 average AUC, 0.81 average recall, 0.82 average precision, and 81.51 average accuracy.

In this paper, the Friedman test with a significance level of 0.05 and two degrees of freedom has been considered to find the significant impact of sampling techniques on model performance. Table IV shows the mean ranks using the Friedman test for SMOTE, BLSMOTE, and original data. The smaller value of P represents that the models trained on sampled data have significant improvement in performance as compared to the original data. Similarly, the models trained on SMOTE sampled data have better capability to predict sentiments as compared to BLSMOTE and original data.

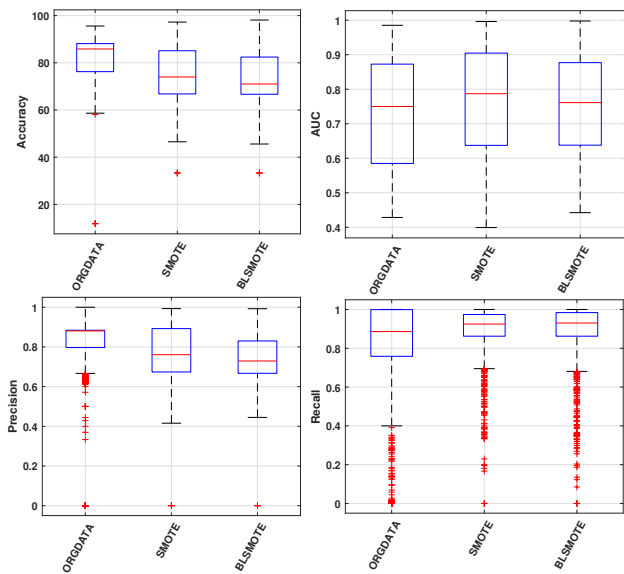


Fig. 7: Performance Box-Plot Diagram: Performance of Original Data and SMOTE

TABLE VII: Descriptive Statistics: OD and SMOTE

	Min	Max	Mean	Q2	Q1	Q3
<b>Accuracy</b>						
ORGDATA	11.87	95.57	81.51	85.86	76.24	88.13
SMOTE	33.33	97.25	76.00	73.99	66.81	85.09
BLSMOTE	33.33	98.11	74.16	71.01	66.67	82.43
<b>Precision</b>						
ORGDATA	0.00	1.00	0.82	0.88	0.80	0.88
SMOTE	0.00	0.99	0.78	0.76	0.67	0.89
BLSMOTE	0.00	0.99	0.75	0.73	0.67	0.83
<b>Recall</b>						
ORGDATA	0.00	1.00	0.81	0.89	0.76	1.00
SMOTE	0.00	1.00	0.88	0.92	0.86	0.97
BLSMOTE	0.00	1.00	0.87	0.93	0.86	0.98
<b>AUC</b>						
ORGDATA	0.43	0.99	0.73	0.75	0.59	0.87
SMOTE	0.40	1.00	0.76	0.79	0.64	0.90
BLSMOTE	0.44	1.00	0.75	0.76	0.64	0.88

## VII. CONCLUSION

Sentiment analysis prediction models for software engineers help in various engineering tasks like analyzing developers' sentiments, evaluating app reviews, users' sentiments of software products, etc. The work presented in this paper is a successful effort in the direction of development of software sentiment models by using different variants of embedding techniques, different methods to find important features, different methods to handle the imbalanced nature of the dataset, and finally, different variants of deep-learning for model development. The performance of the developed models is computed and compared using accuracy, precision, AUC, and recall. We have also applied the Friedman test to statistically examine the performance of models developed using a different combination of features. The major findings

are summarized as follows:

- The high value of AUC for the trained models confirms the capability of the models to predict sentiment based on text comments.
- The use of sampling techniques like SMOTE and BLSMOTE significantly helps in improving the performance of software sentiment prediction models.
- The models trained by using selected sets of features using ANOVA achieved better performance as compared to other techniques.
- The deep learning with one dropout layer and two hidden layers achieved better performance as compared to other combinations.

## VIII. ACKNOWLEDGEMENTS

This research is funded by TestAIng Solutions Pvt. Ltd.

## REFERENCES

- [1] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 94–104.
- [2] E. Biswas, K. Vijay-Shanker, and L. Pollock, "Exploring word embedding techniques to improve sentiment analysis of software engineering texts," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 68–78.
- [3] M. R. Islam and M. F. Zibran, "Leveraging automated sentiment analysis in software engineering," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 203–214.
- [4] L. Kumar, S. Misra, and S. K. Rath, "An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes," *Computer Standards & Interfaces*, vol. 53, pp. 1–32, 2017.
- [5] R. Jindal, R. Malhotra, and A. Jain, "Software defect prediction using neural networks," in *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization*. IEEE, 2014, pp. 1–6.
- [6] G. I. P. Sari and D. O. Siahaan, "An attribute selection for severity level determination according to the support vector machine classification result," in *Proceedings of the 1st international conference on information systems for business competitiveness (ICISBC)*, 2011.
- [7] R. Malhotra and M. Khanna, "A text mining framework for analyzing change impact and maintenance effort of software bug reports," *International Journal of Information Retrieval Research (IJIRR)*, vol. 12, no. 1, pp. 1–18, 2022.
- [8] R. Malhotra and J. Jain, "Predicting defects in imbalanced data using resampling methods: an empirical investigation," *PeerJ Computer Science*, vol. 8, p. e573, 2022.
- [9] L. Kumar, M. Kumar, L. B. Murthy, S. Misra, V. Kocher, and S. Padmanabhuni, "An empirical study on application of word embedding techniques for prediction of software defect severity level," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*. IEEE, 2021, pp. 477–484.
- [10] R. Panigrahi, L. Kumar, and S. K. Kuanar, "An empirical study to investigate different smote data sampling techniques for improving software refactoring prediction," in *International Conference on Neural Information Processing*. Springer, 2020, pp. 23–31.
- [11] L. Kumar, S. K. Sripada, A. Sureka, and S. K. Rath, "Effective fault prediction model developed using least square support vector machine (lssvm)," *Journal of Systems and Software*, vol. 137, pp. 686–712, 2018.