

Software Sentiment Analysis Using Machine Learning with Different Word-Embedding

Venkata Krishna Chandra Mula[1], Sanidhya Vijayvargiya[2(✉)], Lov Kumar[2], Surender Singh Samant[3], and Lalita Bhanu Murthy[2]

[1] Andhra University College of Engineering, Visakhapatnam, India
[2] BITS-Pilani Hyderabad, Hyderabad, India
{f20202056,lovkumar,bhanu}@hyderabad.bits-pilani.ac.in
[3] Graphic Era, Dehradun, India

Abstract. Software sentiment analysis has applications in numerous software engineering tasks ranging from code suggestions to evaluating app reviews which help to save the development team valuable time and increase productivity. In recent years, sentiment analysis has been used to study the emotional state of developers through sources like commit messages. State-of-the-art sentiment analysis techniques have been employed to accomplish these tasks with varying results. The goal of this paper is to provide a comparison between the performance of various models for possible applications of sentiment analysis in software engineering. We have used three different datasets to account for the possible applications: JIRA, AppReviews, and StackOverflow. In this work, six word embedding techniques have been applied on above datasets to represent the text as n-dimensional vectors. To handle the skewed distribution of classes present in the data, we have employed two class balancing techniques in the form of SMOTE and Borderline-SMOTE. The resulting data is subjected to six feature selection techniques, and finally, the sentiment of the text is classified using 14 different classifiers. The experimental results suggest that some models are very successful in accurately classifying the sentiment of the text, whereas choosing the wrong combination of ML techniques can lead to disappointing performance.

Keywords: Sentiment analysis · Word embedding · SMOTE

1 Introduction

Sentiment Analysis Tools and Techniques have witnessed an increase of usage in the Software Engineering World. Usually, Sentiment Analysis is used to identify the mood and feelings behind textual data and reviews, by leveraging Machine Learning Algorithms and Natural Language techniques. It is usually used by corporates to identify the sentiment behind user reviews, to garner feedback, be it positive, negative, or neutral, and to improve their products. In the area of

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13381, pp. 396–410, 2022.
https://doi.org/10.1007/978-3-031-10548-7_29

Software Engineering and its tasks, Sentiment Analysis can be used in various ways, such as to determine the polarity of user App Reviews, to evaluate the mood and mindset of software developers through their commit messages. Sentiment Analysis for Software Engineering Tasks aims to develop models which can accurately and efficiently predict the mood and feelings behind Software Engineering Tasks. The first thing to do to develop the predictive model is to perform word embedding techniques on the datasets, to convert textual data into vectorized data that is easier to analyze and process.

The experimental datasets were sourced from user's App Reviews, Stack Overflow comments and messages, JIRA issues tracked. All unnecessary spaces, symbols and stopwords were cleaned and Word embedding techniques were used on these datasets to express textual data as vectors. Six word embedding techniques have been applied in total to vectorize the data and preserve their semantic integrity. The application of the word embedding techniques now result in an abundance of features, of which not all are entirely useful to build predictive models. As a matter of fact, too many features can cause the accuracy of predictive models to plummet. We make use of six different Feature Selection Techniques to extract the most important and relevant features to improve the performance of the predictive model. The predictive ability of these techniques applied needs to be evaluated to make an informed decision on which word embedding technique is to be used to obtain the maximum performance and efficiency. The performance is evaluated by means of fourteen different classifiers. The application of classifiers on data that has been plagued with the Class Imbalance Problem will result in incorrect and unusable results. To rectify this, the classifiers are applied on the resulting datasets which have been rid of the Class Imbalance Problem, by the application of Synthetic Minority Oversampling Technique and Borderline Synthetic Minority Oversampling Technique. The challenges faced during the development of the model were:

- Word Embeddings: The dataset of user App Reviews, messages and comments on StackOverflow were represented as numerical features retaining their semantics in an n-dimensional space using six different word embedding techniques. The techniques used were 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).
- High Volume of Features: The application of the word embedding techniques have resulted in a very high volume of features data. Using all of the features data could affect the performance of the model adversely. So important features must be optimized. In this study, we have used six different Feature Selection Techniques to extract the relevant features to optimize the development of the model.
- Imbalanced Data: The data used to build the model is found to be imbalanced, which affects the performance of the model. A balanced dataset is one in which each class should have the same number of samples. Synthetic Minority Oversampling Technique (SMOTE) and Borderline Synthetic Minority

Oversampling Technique (Borderline-SMOTE) have been applied to resolve the class imbalance problem of the datasets.

This study aims to evaluate the performance of models developed using different Word Embedding and Feature Selection Techniques with the help of fourteen different classifiers, so that the model can accurately and effectively predict the sentiment behind Software Engineering Tasks.

2 Related Work

Different methods have been used to extract relevant features from textual content in the past. Rajni Jindal et al. have used the Term Frequency and Inverse Document Frequency (TF-IDF) to extract features from defect descriptions [3]. The classification of the defect reports has been done by the Radial Basis function of the Neural Network. The results have concluded that the model has performed very well in predicting high severity defects. Sari and Siahaan have also used Term Frequency and Inverse Document Frequency (TF-IDF) to tokenize the defect description. They have applied InfoGain Technique to extract the relevant features from the tokenized data. They have used a Support Vector Machine to develop Severities Prediction Models to predict the severity levels of defects [7].

Lov Kumar et al. have worked on applying word embedding techniques to predict software defect severity level [5]. They have also used seven different word embedding techniques to tokenize defect descriptions. Three Feature Selection Techniques have been used to extract the relevant features. The effectiveness of the word embeddings are evaluated by eleven different Classification Techniques with Synthetic Minority Oversampling Technique (SMOTE) to overcome the class imbalance problem. The results conclude that the word embedding, feature selection techniques and SMOTE can predict the severity level of a defect in a software.

Software sentiment analysis is a tool that has various applications but cannot yet produce satisfactory accuracy. Lin et al. attempted to solve this problem by retraining a state-of-the-art sentiment analysis tool exploiting deep learning on a set of 40k manually labelled sentences from StackOverflow [6]. They assigned a sentiment score to the 40k sentences, yet their effort yielded negative results with low accuracy levels. They conducted a comparison between Stanford CoreNLP SO and Stanford CoreNLP. They concluded that Stanford CoreNLP SO has a positive impact on the performance of sentiment analysis tools over Stanford CoreNLP. Another conclusion reached by the authors in their research was that the accuracy of sentiment analysis tools is, in general, poor on SE datasets.

Biswas et al. took a different approach by attempting to improve performance using software domain-specific word embeddings learned from StackOverflow [1]. They compared the impact on the performance of sentiment analysis tools using the domain-specific word embeddings with those that used generic word embeddings learned from Google News. Their results showed that the generic word

embeddings performed at par or better than the domain-specific word embeddings. The domain-specific word embeddings led to a 10–20% decrease in precision compared to the generic word embeddings. Biswas et al. also studied the impact of oversampling and undersampling techniques for handling small SE datasets with a skewed distribution. They arrived at the conclusion that is oversampling alone, or a combination of undersampling and oversampling together, in the training of sentiment analysis tools resulted in improved performance. They achieved recall values of 56% for negative sentences and 46% for positive sentences, a major improvement on the work done by Lin et al.

Islam et al. proposed a tool for sentiment analysis, SentiStrength-SE, which achieved 73.85% precision and 85% recall [2]. In their work, they highlight some of the difficulties faced by sentiment analysis tools which include:

- Domain-specific meaning of words
- Context-sensitive variations in meanings of words
- Sentimental words in copy-pasted content (e.g., code)
- Difficulties in dealing with negation
- Wrong detection of proper nouns
- Difficulty in dealing with irony and sarcasm

These difficulties are faced by most state-of-the-art tools and highlight why achieving high accuracies in software sentiment analysis is still a challenging task.

3 Study Design

3.1 Experimental Dataset

In this experiment, three different datasets have been used, which are AppReviews, JIRA and StackOverflow. These datasets are collections of reviews, comments, etc. relating to Software Engineering and its tasks. Each dataset contains 'Sentence's' and their corresponding 'Oracles', by which the sentiment can be determined.

3.2 Word Embeddings

The textual data of the dataset is to be expressed as vectors in relation with 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.

3.3 Feature Selection Techniques

As we use the vectors as inputs to develop the model, the performance depends upon the optimization of the important features. To extract the important features from the existing set of vectors. We use six different Feature Selection Techniques, i.e. Principal Components Analysis, Analysis of Variance (ANOVA), GainRatioAttributeEval, InfoGainAttributeEval, ClassifierAttributeEval, OneRAttributeEval to remove irrelevant features and provide us with the set of relevant and important features.

3.4 Overcoming the Class Imbalance Problem

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.

3.5 Classification Techniques

The predictive ability of the different techniques used are evaluated using fourteen different classifiers such as Multinomial Naive Bayes (MNB), Bernoulli's Naive Bayes (BNB), Gaussian Naive Bayes (GNB), Complement Naïve Bayes (CNB), Decision Tree Classifier, KNeighbours Classifier, SVM with Linear Kernel (SVML), SVM with Polynomial Kernel (SVMP), SVM with RBF Kernel (SVMR), MinmaxScaler, BaggingClassifier, RandomForestClassifier, ExtraTreesClassifier, AdaBoostClassififer, GradientBoostingClassifier.

4 Research Methodology

In this work, we have applied five different word embedding methods to extract features from the software requirements in natural language, balanced the classes using SMOTE, selected features from both the balanced and imbalanced datasets, and used them as input to develop models for classification. These models are trained with eight different classifiers, with one of them being a soft voting ensemble classifier. The proposed methodology entails extracting features from text data using word embedding techniques, class balancing, deleting irrelevant and redundant features, and lastly, creating prediction models utilizing eight distinct classifiers. In this experiment, we employed the Rank-Sum test to identify features that may distinguish between functional and non-functional needs, as well as Principal component analysis to reduce the variance of the numerical features produced. Finally, eight distinct classifiers, including an ensemble classifier, are used to create the required classifications, which are then confirmed

using 10-fold cross-validation. The accuracy, F-measure, and AUC of the developed requirement categorization models are extracted and compared using the Box-plot and Friedman test.

5 Empirical Results and Analysis

In this work, we have applied six different word embedding techniques, two class balancing techniques, six feature selection techniques, and fourteen different classification techniques to analyze the sentiment of the software engineering databases. Therefore, a total of 5292 [3 datasets * 6 word-embedding techniques * (2 sets of class-balancing techniques +1 set of original data) * 7 sets of features * 14 classifiers] distinct prediction models are built in this study. The predictive ability of these trained models is evaluated with Area Under Curve (AUC) and accuracy values, as seen in Tables 1 and 2 respectively. The first observation of note is that the range of accuracy from 13.73% to 98.68% shows that certain models perform exceedingly well but choosing the wrong combination of ML techniques can lead to very negative results. A second noteworthy observation is that the performance of the same ML model varies significantly between JIRA, AppReviews, and StackOverflow datasets. The performance metrics are lower for StackOverflow dataset than the other two datasets, which shows that sentiment prediction is a more challenging task for StackOverflow. This is corroborated by the fact that there was disagreement in 18.6% of the classifications, even when done by humans.

5.1 Comparative Analysis

The various models created utilizing word embedding techniques, class balancing methods, feature selection techniques, and different classifiers are compared in this section. The comparison is based on statistics like area under the ROC curve (AUC) and accuracy, with box plots serving as a visual depiction 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***- The prediction ability of models constructed for sentiment analysis utilizing various combinations of word embedding, class balancing, feature selection, and classification methods is not significantly different.
- ***Alternate Hypothesis***- The prediction ability of models constructed for sentiment analysis utilizing various combinations of word embedding, class balancing, feature selection, and classification methods is significantly different.

The Friedman test was carried out with a significance threshold of $\alpha = 0.05$ with degrees of freedom as 5 for word embedding, 2 for class balancing, 6 for feature selection, and 13 for different classifier comparisons.

Table 1. AUC: all features

		MNB	BNB	GNB	CNB	DT	KNN	SVC-LIN	SVC-Poly	SVC-RBF	BAG	RF	EXTR	AdaB	GRaB
ORGDATA															
TFIDF	AppReviews	0.90	0.90	0.82	0.91	0.81	0.82	0.88	0.85	0.89	0.87	0.89	0.90	0.83	0.83
TFIDF	JIRA	0.96	0.97	0.86	0.96	0.91	0.90	0.95	0.93	0.93	0.93	0.95	0.95	0.92	0.90
TFIDF	StackOverflow	0.73	0.73	0.56	0.74	0.58	0.66	0.66	0.65	0.69	0.70	0.67	0.65	0.66	0.59
CBOW	AppReviews	0.48	0.40	0.56	0.74	0.63	0.76	0.83	0.82	0.83	0.77	0.77	0.79	0.77	0.77
CBOW	JIRA	0.54	0.47	0.50	0.82	0.78	0.94	0.89	0.90	0.89	0.95	0.93	0.91	0.89	0.87
CBOW	StackOverflow	0.46	0.45	0.51	0.53	0.52	0.63	0.47	0.46	0.47	0.65	0.59	0.58	0.59	0.61
SKG	AppReviews	0.64	0.39	0.70	0.65	0.66	0.72	0.85	0.82	0.84	0.73	0.75	0.76	0.81	0.82
SKG	JIRA	0.79	0.50	0.75	0.80	0.82	0.94	0.93	0.95	0.91	0.95	0.93	0.95	0.91	0.91
SKG	StackOverflow	0.47	0.45	0.54	0.54	0.52	0.61	0.55	0.55	0.51	0.62	0.61	0.60	0.62	0.63
GLOVE	AppReviews	0.89	0.42	0.86	0.89	0.71	0.83	0.89	0.86	0.91	0.86	0.83	0.81	0.84	0.84
GLOVE	JIRA	0.96	0.50	0.96	0.96	0.82	0.96	0.97	0.96	0.97	0.96	0.94	0.94	0.94	0.94
GLOVE	StackOverflow	0.74	0.48	0.71	0.74	0.56	0.67	0.75	0.73	0.76	0.73	0.67	0.65	0.69	0.67
GW2V	AppReviews	0.91	0.44	0.86	0.91	0.72	0.86	0.92	0.91	0.93	0.88	0.85	0.83	0.83	0.82
GW2V	JIRA	0.97	0.50	0.92	0.97	0.83	0.97	0.98	0.98	0.99	0.98	0.95	0.97	0.93	0.93
GW2V	StackOverflow	0.79	0.46	0.78	0.80	0.52	0.71	0.77	0.77	0.83	0.77	0.62	0.67	0.67	0.63
FASTXT	AppReviews	0.63	0.37	0.70	0.64	0.58	0.68	0.69	0.71	0.77	0.67	0.65	0.67	0.75	0.70
FASTXT	JIRA	0.83	0.53	0.78	0.83	0.77	0.87	0.88	0.91	0.91	0.88	0.87	0.87	0.84	0.84
FASTXT	StackOverflow	0.59	0.45	0.60	0.59	0.51	0.62	0.50	0.59	0.58	0.60	0.56	0.57	0.59	0.60
SMOTE															
TFIDF	AppReviews	0.86	0.87	0.80	0.87	0.83	0.87	0.90	0.92	0.93	0.89	0.93	0.95	0.85	0.80
TFIDF	JIRA	0.96	0.97	0.87	0.96	0.94	0.95	0.97	0.96	0.95	0.96	0.97	0.97	0.94	0.93
TFIDF	StackOverflow	0.84	0.85	0.66	0.84	0.88	0.91	0.83	0.87	0.92	0.89	0.94	0.95	0.72	0.70
CBOW	AppReviews	0.52	0.46	0.56	0.75	0.76	0.86	0.84	0.84	0.86	0.87	0.89	0.92	0.78	0.79
CBOW	JIRA	0.64	0.49	0.52	0.87	0.84	0.97	0.89	0.94	0.86	0.97	0.97	0.97	0.91	0.92
CBOW	StackOverflow	0.51	0.48	0.52	0.56	0.82	0.97	0.76	0.78	0.76	0.98	0.98	0.98	0.74	0.75
SKG	AppReviews	0.61	0.44	0.64	0.62	0.78	0.86	0.86	0.89	0.87	0.84	0.90	0.91	0.82	0.81
SKG	JIRA	0.79	0.48	0.77	0.79	0.90	0.96	0.95	0.96	0.93	0.97	0.98	0.99	0.94	0.94
SKG	StackOverflow	0.50	0.48	0.54	0.60	0.85	0.96	0.80	0.81	0.80	0.96	0.98	0.99	0.78	0.79
GLOVE	AppReviews	0.86	0.45	0.87	0.87	0.78	0.91	0.93	0.94	0.97	0.92	0.93	0.93	0.86	0.85
GLOVE	JIRA	0.97	0.47	0.97	0.97	0.88	0.97	0.99	0.98	0.99	0.98	0.98	0.99	0.96	0.96
GLOVE	StackOverflow	0.76	0.47	0.76	0.76	0.85	0.96	0.89	0.99	0.99	0.98	0.98	0.99	0.81	0.80
GW2V	AppReviews	0.88	0.47	0.86	0.88	0.78	0.92	0.98	0.98	0.99	0.91	0.93	0.94	0.84	0.86
GW2V	JIRA	0.97	0.52	0.93	0.97	0.86	0.97	1.00	1.00	1.00	0.99	0.98	0.98	0.95	0.94
GW2V	StackOverflow	0.85	0.46	0.85	0.85	0.87	0.95	0.97	0.99	1.00	0.97	0.98	0.99	0.82	0.82
FASTXT	AppReviews	0.63	0.45	0.62	0.64	0.69	0.78	0.70	0.79	0.75	0.75	0.84	0.88	0.78	0.76
FASTXT	JIRA	0.84	0.56	0.77	0.84	0.82	0.93	0.90	0.95	0.94	0.93	0.96	0.97	0.88	0.88
FASTXT	StackOverflow	0.65	0.48	0.62	0.66	0.85	0.97	0.53	0.91	0.88	0.97	0.97	0.99	0.77	0.76
BLSMOTE															
TFIDF	AppReviews	0.87	0.88	0.77	0.87	0.83	0.87	0.90	0.89	0.93	0.89	0.93	0.93	0.85	0.81
TFIDF	JIRA	0.96	0.98	0.87	0.96	0.94	0.92	0.97	0.96	0.96	0.96	0.98	0.97	0.92	0.92
TFIDF	StackOverflow	0.87	0.89	0.70	0.87	0.91	0.94	0.87	0.91	0.95	0.93	0.96	0.97	0.75	0.74
CBOW	AppReviews	0.52	0.46	0.55	0.74	0.72	0.86	0.83	0.84	0.85	0.86	0.89	0.91	0.74	0.77
CBOW	JIRA	0.54	0.47	0.52	0.62	0.86	0.95	0.88	0.92	0.76	0.96	0.97	0.98	0.88	0.88
CBOW	StackOverflow	0.53	0.48	0.53	0.68	0.84	0.97	0.77	0.78	0.78	0.98	0.98	0.98	0.77	0.76
SKG	AppReviews	0.61	0.47	0.59	0.61	0.75	0.85	0.85	0.89	0.87	0.82	0.90	0.91	0.82	0.87
SKG	JIRA	0.66	0.49	0.61	0.66	0.88	0.94	0.91	0.94	0.89	0.96	0.99	0.99	0.90	0.91
SKG	StackOverflow	0.53	0.49	0.55	0.55	0.89	0.95	0.82	0.82	0.82	0.96	0.98	0.99	0.80	0.80
GLOVE	AppReviews	0.84	0.45	0.87	0.85	0.81	0.90	0.94	0.92	0.97	0.91	0.93	0.94	0.87	0.88
GLOVE	JIRA	0.92	0.50	0.93	0.93	0.86	0.94	0.97	0.98	0.99	0.97	0.98	0.98	0.93	0.93
GLOVE	StackOverflow	0.76	0.48	0.76	0.76	0.87	0.96	0.89	0.98	0.99	0.98	0.98	0.99	0.82	0.82
GW2V	AppReviews	0.88	0.45	0.90	0.89	0.78	0.89	0.96	0.98	0.98	0.89	0.93	0.94	0.84	0.84
GW2V	JIRA	0.97	0.49	0.98	0.97	0.89	0.92	0.99	1.00	1.00	0.98	0.99	0.99	0.95	0.95
GW2V	StackOverflow	0.85	0.45	0.85	0.85	0.86	0.95	0.97	0.98	1.00	0.97	0.98	0.99	0.84	0.84
FASTXT	AppReviews	0.60	0.44	0.60	0.61	0.69	0.80	0.66	0.79	0.75	0.77	0.87	0.90	0.78	0.79
FASTXT	JIRA	0.78	0.56	0.70	0.79	0.81	0.91	0.85	0.94	0.92	0.91	0.96	0.97	0.86	0.85
FASTXT	StackOverflow	0.65	0.48	0.62	0.65	0.86	0.96	0.56	0.92	0.89	0.96	0.98	0.99	0.76	0.78

Table 2. Accuracy: all features

AF															
		MNB	BNB	GNB	CNB	DT	KNN	SVC-LIN	SVC-Poly	SVC-RBF	BAG	RF	EXTR	AdaB	GRaB
ORGDATA															
TFIDF	AppReviews	82.40	83.87	72.14	80.65	81.82	79.47	79.77	72.73	81.52	78.59	83.28	81.23	82.70	83.87
TFIDF	JIRA	90.39	92.33	81.64	91.25	90.71	82.07	90.93	84.56	91.25	87.69	91.68	91.79	92.01	91.47
TFIDF	StackOverflow	88.20	88.20	36.27	62.93	83.13	87.87	88.07	87.67	87.73	88.00	85.40	83.80	88.13	88.53
CBOW	AppReviews	61.88	58.36	47.51	51.32	65.69	74.78	61.58	76.83	61.88	75.66	72.43	73.02	72.14	73.61
CBOW	JIRA	68.68	68.57	67.82	68.03	80.56	90.06	68.79	87.91	68.68	90.28	87.47	87.91	84.56	83.59
CBOW	StackOverflow	88.13	87.33	19.20	76.93	76.53	87.87	88.13	88.13	88.13	88.13	85.80	87.33	87.20	87.33
SKG	AppReviews	61.88	57.48	59.82	58.94	68.04	68.33	77.71	79.47	76.83	68.33	68.62	70.97	74.49	75.95
SKG	JIRA	76.89	69.44	80.45	77.21	84.13	91.68	87.58	91.47	86.72	91.58	90.50	90.50	87.69	88.55
SKG	StackOverflow	88.13	87.67	21.87	50.20	78.27	86.60	88.13	88.13	88.13	87.80	86.87	86.87	87.87	87.33
GLOVE	AppReviews	72.14	57.77	76.83	80.65	71.85	78.01	81.82	78.01	84.46	76.54	75.07	75.37	76.54	78.01
GLOVE	JIRA	80.67	69.44	89.63	91.14	84.34	93.63	93.63	91.36	94.28	92.44	90.82	89.96	89.63	88.98
GLOVE	StackOverflow	88.13	87.53	73.93	68.93	79.60	86.93	88.00	84.40	88.13	87.93	86.53	87.60	86.60	86.80
GW2V	AppReviews	82.70	56.31	80.94	81.52	73.31	81.23	84.16	85.63	84.75	80.94	77.42	76.25	75.66	76.25
GW2V	JIRA	87.47	66.74	92.01	90.17	85.42	95.14	93.84	94.17	95.25	94.71	90.39	90.60	87.91	88.88
GW2V	StackOverflow	88.13	86.13	79.87	75.13	79.73	87.73	86.47	85.20	88.20	88.53	87.33	88.20	87.40	87.07
FASTXT	AppReviews	62.17	50.73	64.22	58.06	59.24	62.46	65.10	66.28	72.43	62.17	63.93	64.52	68.04	63.93
FASTXT	JIRA	82.59	68.11	69.95	78.70	79.78	84.00	84.86	85.19	86.16	83.78	83.68	84.11	80.65	80.54
FASTXT	StackOverflow	88.09	85.81	48.80	61.45	75.64	87.01	88.09	85.68	88.09	87.95	86.01	87.01	87.62	86.95
SMOTE															
TFIDF	AppReviews	81.90	81.36	66.49	75.99	83.87	81.54	84.23	80.82	89.07	80.29	85.66	89.96	83.33	84.59
TFIDF	JIRA	87.74	90.02	86.71	88.99	90.72	85.69	89.23	85.38	90.17	88.05	91.27	91.75	88.99	88.29
TFIDF	StackOverflow	74.95	79.85	55.56	72.07	87.91	84.83	78.25	79.46	86.12	82.26	88.41	90.23	74.78	76.49
CBOW	AppReviews	66.67	64.70	43.37	55.38	77.78	81.00	65.95	79.21	66.67	80.82	82.08	86.20	76.88	75.45
CBOW	JIRA	48.90	48.66	50.24	65.02	83.57	88.76	75.47	87.34	48.82	87.74	91.12	92.61	83.18	85.38
CBOW	StackOverflow	66.67	66.33	37.25	35.94	83.24	89.87	66.72	66.67	66.75	91.30	92.86	94.51	72.60	73.92
SKG	AppReviews	66.67	64.70	53.76	51.97	79.93	81.00	78.14	87.10	76.88	78.32	82.62	85.30	78.67	79.75
SKG	JIRA	72.64	49.61	75.08	72.56	89.70	88.60	86.95	91.35	83.33	89.62	92.61	94.50	86.16	87.50
SKG	StackOverflow	66.67	66.47	39.71	47.33	85.61	87.97	66.75	66.61	66.67	87.01	93.62	94.82	72.77	75.23
GLOVE	AppReviews	66.85	64.16	80.82	81.54	81.72	85.13	87.10	89.96	90.14	81.18	86.02	86.74	82.97	79.93
GLOVE	JIRA	89.70	47.64	90.57	89.39	87.66	88.13	94.26	95.13	95.60	90.33	93.40	94.03	87.66	88.36
GLOVE	StackOverflow	66.67	66.41	65.02	67.87	86.01	89.00	84.19	96.42	95.24	88.13	94.01	95.66	75.15	75.04
GW2V	AppReviews	82.26	63.26	77.06	78.67	80.29	84.05	92.83	93.91	91.94	80.11	85.66	88.17	79.39	79.21
GW2V	JIRA	86.48	52.99	88.60	86.40	86.01	86.64	96.31	96.70	97.80	87.58	93.40	94.10	86.48	86.32
GW2V	StackOverflow	69.07	65.91	76.91	75.45	87.41	87.29	93.59	96.78	98.18	86.23	94.29	95.52	76.60	77.53
FASTXT	AppReviews	67.92	61.47	54.84	63.98	72.94	73.84	71.86	77.24	73.30	70.43	78.32	82.97	76.88	72.76
FASTXT	JIRA	76.65	55.90	72.64	76.81	82.08	83.18	82.08	88.92	88.05	82.55	90.33	90.72	79.09	79.01
FASTXT	StackOverflow	66.64	65.99	53.50	61.91	85.32	88.86	66.67	87.48	79.58	88.19	91.84	94.85	73.64	74.51
							BLSMOTE								
TFIDF	AppReviews	80.11	81.36	59.68	73.84	83.69	80.65	84.95	79.21	87.99	79.39	88.53	89.61	84.59	83.69
TFIDF	JIRA	90.72	92.61	86.24	91.67	92.06	87.97	89.78	87.11	91.51	88.76	93.08	93.32	85.46	86.87
TFIDF	StackOverflow	76.07	81.50	59.78	77.02	90.79	87.99	78.65	80.24	88.19	85.50	91.13	92.33	76.85	77.16
CBOW	AppReviews	66.67	64.87	41.94	56.81	75.45	80.65	66.31	77.60	66.67	78.32	80.82	81.90	73.48	75.81
CBOW	JIRA	50.31	48.27	52.12	53.07	86.01	85.06	57.00	85.30	50.31	80.58	91.51	92.53	79.32	80.97
CBOW	StackOverflow	66.67	66.33	38.32	37.14	84.80	89.98	66.67	67.03	66.67	92.02	93.00	94.37	73.22	72.82
SKG	AppReviews	66.67	64.70	51.43	46.77	77.96	79.57	75.99	86.74	75.99	76.52	84.59	82.97	80.29	82.44
SKG	JIRA	57.55	50.55	60.53	58.57	88.05	85.38	80.66	88.29	74.61	82.70	95.05	94.10	81.21	84.98
SKG	StackOverflow	66.67	66.47	40.78	52.50	89.08	87.74	66.64	67.23	66.67	88.02	94.07	94.32	75.17	75.54
GLOVE	AppReviews	67.03	64.16	79.93	78.32	83.87	84.23	86.02	89.78	90.32	82.62	87.10	86.74	80.65	83.51
GLOVE	JIRA	84.36	49.61	83.81	84.75	86.32	82.86	92.53	94.73	95.83	80.50	93.40	94.18	84.75	84.83
GLOVE	StackOverflow	66.67	66.41	64.82	68.71	87.32	87.57	83.82	96.42	95.13	88.11	93.51	95.33	75.65	76.49
GW2V	AppReviews	78.49	63.26	81.90	78.14	80.65	80.65	91.40	92.11	94.44	77.06	87.46	87.10	79.21	79.75
GW2V	JIRA	89.47	52.83	93.47	89.23	88.68	79.32	96.78	97.25	98.66	76.34	93.47	95.99	88.29	88.76
GW2V	StackOverflow	68.88	65.91	76.66	77.05	86.93	86.65	94.29	96.25	98.07	85.73	95.02	96.08	77.92	78.90
FASTXT	AppReviews	66.49	62.37	53.76	56.45	72.40	72.76	67.74	74.01	69.89	70.43	79.93	81.54	73.12	74.91
FASTXT	JIRA	68.79	56.37	67.30	68.87	81.45	79.95	75.79	88.60	83.10	77.28	88.60	90.64	76.97	77.12
FASTXT	StackOverflow	66.61	65.94	52.97	63.04	86.02	88.72	66.67	87.51	81.49	87.51	93.45	95.11	73.84	74.68

5.2 Word-Embedding

The numerical vector of the three software engineering datasets was computed using six different word embedding techniques, such as 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), and fasttext (FST). The AUC statistic has been used to compare the prediction ability of models created using various word embedding approaches.

Box-Plot: Word-Embedding. Figure 1 graphically depicts the performance value of various word embedding algorithms. Models developed with the word vectors computed by GLOVE and GW2V are more predictive than other models, as shown in Fig. 1. The models created with CBOW have a low predictive performance when compared to other approaches. The average AUC of GW2V-created models is 0.84, with a maximum AUC of 1.00 and a Q3 AUC of 0.95, implying that 25% of GW2V-created models have an AUC greater than 0.95. The accuracy data of these models supports these conclusions, revealing that sentiment prediction models based on GW2V and GLOVE outperform models based on alternative word-embedding approaches.

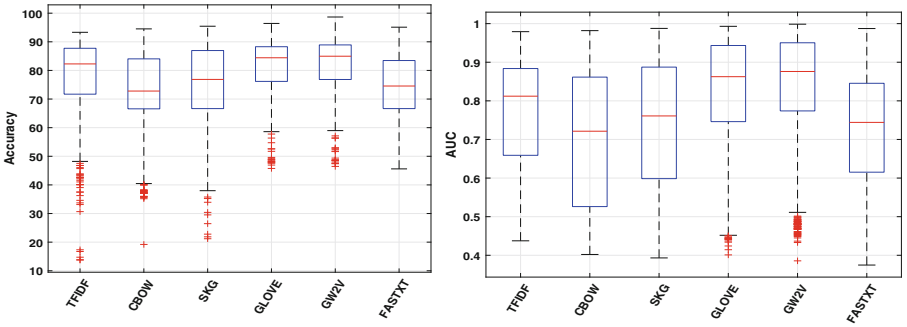

Fig. 1. Box-plot: word-embedding

Friedman Test: Word-Embedding. In this work, the Friedman Test is also utilized to examine the predictive ability of the models created using different word embedding techniques. The goal of the test is to see if the null hypothesis is correct or not. The null hypothesis asserts that "the various word embedding techniques have no discernible impact on the performance of the sentiment analysis models." Table 3 shows the mean ranks for the various word embedding algorithms. The lower the mean rank, the better the models' performance. GW2V has the lowest mean rank of 1.715, while CBoW has the highest mean rank of 4.906. GW2V outperforms GLOVE by a small margin, with a mean rank of 2.204. This performance supports the findings of Biswas et al. where

word embedding with generic pre-trained vectors had a similar or better performance than other word embedding techniques, even software domain-specific word embeddings [1].

5.3 Feature Selection

In the proposed research, we use six different types of feature selection techniques- Principal Components Analysis, Analysis of Variance (ANOVA), GainRatioAttributeEval, InfoGainAttributeEval, ClassifierAttributeEval, OneRAttributeEval and in the seventh set of models, we use all of the original features for training predictive models for sentiment prediction.

Table 3. Friedman test: word-embedding

	TFIDF	CBOW	SKG	GLOVE	GW2V	FASTXT	Rank
TFIDF	1.00	0.00	0.00	0.00	0.00	0.00	3.51
CBOW	0.00	1.00	0.00	0.00	0.00	0.00	4.91
SKG	0.00	0.00	1.00	0.00	0.00	0.05	4.15
GLOVE	0.00	0.00	0.00	1.00	0.02	0.00	2.20
GW2V	0.00	0.00	0.00	0.02	1.00	0.00	1.72
FASTXT	0.00	0.00	0.05	0.00	0.00	1.00	4.52

Box-Plot: Feature Selection. Figure 2 suggests that all feature selection techniques resulted in high performance, with the original set of features having the highest mean of AUC. AF and Anova have a very close mean AUC of 0.81 for all features (AF), 0.80 for Anova test features, and 0.75 for PCA, 0.75 for GainRatioAttributeEval, 0.77 for InfoGainAttributeEval, 0.74 for OneRAttributeEval, and 0.76 for ClassifierAttributeEval. AF has a min of 0.37 AUC, max of 1.00 AUC, Q1 of 0.73 AUC, and Q3 of 0.94 AUC. AF and Anova seem to give

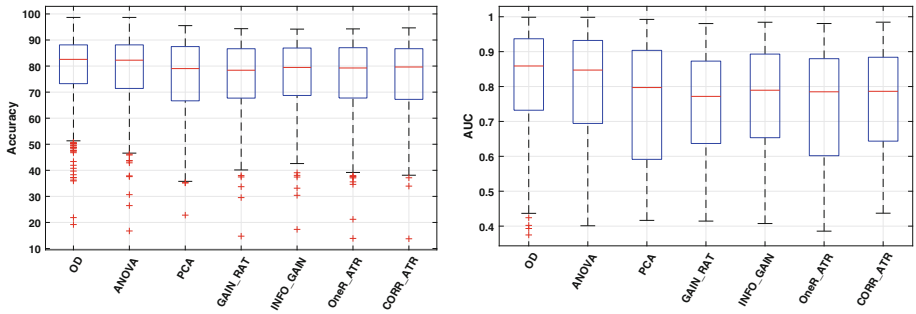

Fig. 2. Box-plot: feature selection techniques

similar results as per the box plot, with the original set of features marginally outperforming the Anova set of features.

Friedman Test: Feature Selection. To assess the different feature selection strategies based on their predictive ability, we used the Friedman test on the AUC values of the models, which were generated using seven distinct sets of features. "The sentiment prediction models created using different sets of features do not have a substantial difference in their predictive ability," is the null hypothesis, which must be accepted or rejected based on the Friedman test. The Friedman test was performed with degrees of freedom of 6 and a significance threshold of $\alpha = 0.05$. The mean ranks of the seven sets of features are shown in Table 4. The Friedman test's mean rankings can be used to compare the performance of various techniques under investigation. Lower mean ranks correspond to better performance in comparison to others. The models trained using the set of original features have the lowest mean rank of 2.42, followed by ANOVA with 2.56, and the worst-performing was OneRAttributeEval with 5.06. The mean ranks show that the models perform better when all features are included, and the use of feature selection techniques only regresses the performance of the models.

Table 4. Friedman test: feature selection techniques

	OD	ANOVA	PCA	GAIN_RAT	INFO_GAIN	OneR_ATR	CORR_ATR	Rank
OD	1.00	0.15	0.00	0.00	0.00	0.00	0.00	2.42
ANOVA	0.15	1.00	0.00	0.00	0.00	0.00	0.00	2.56
PCA	0.00	0.00	1.00	0.56	0.22	0.28	0.78	4.53
GAIN_RAT	0.00	0.00	0.56	1.00	0.03	0.54	0.34	4.92
INFO_GAIN	0.00	0.00	0.22	0.03	1.00	0.01	0.23	3.92
OneR_ATR	0.00	0.00	0.28	0.54	0.01	1.00	0.12	5.06
CORR_ATR	0.00	0.00	0.78	0.34	0.23	0.12	1.00	4.59

5.4 Classification Techniques

To predict the sentiment, the study used fourteen distinct classifiers. These classifiers are used with a variety of word-embedding approaches and on the three datasets, JIRA, AppReviews, and StackOverflow. Multinomial Naive Bayes (MNB), Bernoulli's Naive Bayes (BNB), Gaussian Naive Bayes (GNB), Complement Naive Bayes (CNB), Decision Tree Classifier, k-Neighbours Classifier, SVM with Linear Kernel (SVML), SVM with Polynomial Kernel (SVMP), SVM with RBF Kernel (SVMR), MinmaxScaler, Bagging Classifier, Random Forest Classifier, Extra Trees Classifier, AdaBoost Classifier, and Gradient Boosting Classifier classifiers were employed in this study.

Box-Plot: Classification Techniques. Figure 3 depicts the performance of the fourteen classifiers in visual form. With a mean AUC of 0.87, the models that were trained using the Extra Trees classifier achieved the greatest overall performance. The Multinomial Naive Bayes and Bernoulli's Naive Bayes classifiers performed the worst in contrast to other classifiers, with mean AUCs of 0.68 and 0.51, respectively. The Extra Trees classifier produces models with a maximum AUC of 0.99, a minimum AUC of 0.48, a Q1 AUC of 0.84, and a Q3 AUC of 0.97. The Bagging and Random Forest classifiers fared slightly worse than the Extra Trees classifier but far better than the others.

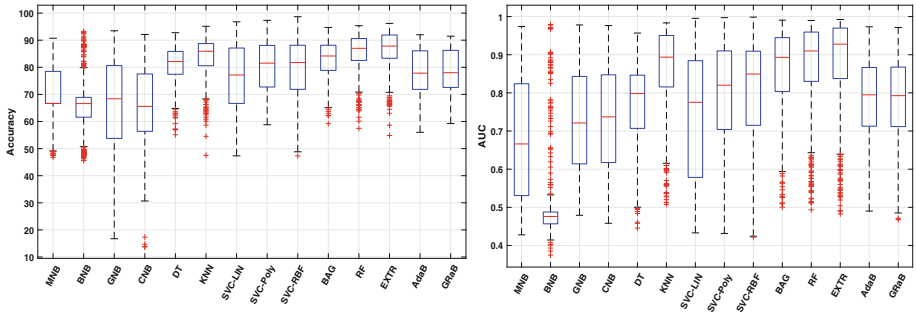

Fig. 3. Box-plot: machine learning

Friedman Test: Classification Techniques. In order to statistically compare the performance of the models, the Friedman test is also applied to the performance metrics of the various classifiers. The goal of the test is to see if the null hypothesis is correct or not. The null hypothesis for this test is that "the sentiment analysis models developed utilizing the different classifiers do not have a significant variation in their prediction abilities." The Friedman test was performed with degrees of freedom as 13 and a significance threshold of = 0.05. After applying the Friedman test, the mean rank of several classifiers is shown in Table 5. We can see that the Extra Trees classifier has the lowest mean rank of all the classifiers, at 3.352. Bernoulli's Naive Bayes classifier has the highest mean rank of 12.847 and thus symbolizes that the classifier is the worst out of the fourteen. The Bagging classifier and the Random Forest classifier also perform well with mean ranks of 4.357 and 4.106, respectively.

Table 5. Friedman test: machine learning

	MNB	BNB	GNB	CNB	DT	KNN	SVC-LIN	SVC-Poly	SVC-RBF	BAG	RF	EXTR	AdaB	GRaB	Rank
MNB	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.95
BNB	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	12.85
GNB	0.00	0.00	1.00	0.32	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.05
CNB	0.00	0.00	0.32	1.00	0.00	0.00	0.38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.64
DT	0.00	0.00	0.00	0.00	1.00	0.00	0.58	0.00	0.00	0.00	0.00	0.00	0.05	0.11	9.77
KNN	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.75	0.03	0.00	0.00	0.00	4.94
SVC-LIN	0.00	0.00	0.09	0.38	0.58	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	8.52
SVC-Poly	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.28	0.00	0.00	0.00	0.06	0.03	6.91
SVC-RBF	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.28	1.00	0.00	0.00	0.00	0.00	0.00	6.12
BAG	0.00	0.00	0.00	0.00	0.00	0.75	0.00	0.00	0.00	1.00	0.02	0.00	0.00	0.00	4.36
RF	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.02	1.00	0.01	0.00	0.00	4.11
EXTR	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	1.00	0.00	0.00	3.35
AdaB	0.00	0.00	0.00	0.00	0.05	0.00	0.01	0.06	0.00	0.00	0.00	0.00	1.00	0.81	7.63
GRaB	0.00	0.00	0.00	0.00	0.11	0.00	0.02	0.03	0.00	0.00	0.00	0.00	0.81	1.00	7.81

5.5 SMOTE

In this paper, there are three different types of models based on the data used for training the models. The first set is class-balanced using SMOTE, the second set is class-balanced using borderline-SMOTE, and the other set contains the original imbalanced classes.

Box-Plot: SMOTE. Figure 4 shows the visual depiction of the predictive ability of the models trained on the two balanced datasets compared to that of the models trained on the imbalanced dataset. The models trained using SMOTE outperformed the models trained on the original dataset in most box plot metrics while marginally outperforming borderline-SMOTE. The models trained with SMOTE had a mean AUC of 0.79, maximum AUC of 1.00, minimum AUC of 0.41, and 0.93 Q3 AUC.

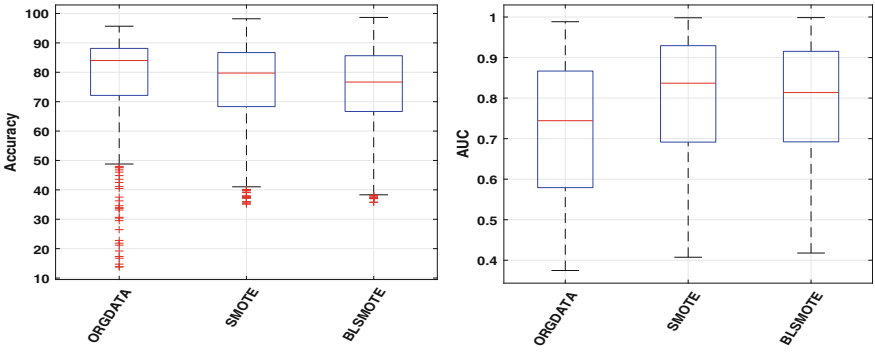

Fig. 4. Box-plot: ORIGINAL DATA and SMOTE

Friedman Test: SMOTE. The Friedman test is applied to the performance metrics of the models trained on the two class-balanced datasets and the imbalanced dataset to help compare the performance of the three sets of models. The purpose of this test is to accept or reject the null hypothesis, which is, "whether the models trained with balanced or imbalanced classes, there is no significant impact on their performance and the ML technique used to balance classes is also of no significance to the performance." The Friedman test was carried out with a significance level of $\alpha = 0.05$ with degrees of freedom as 2. The models trained on the SMOTE balanced dataset have the lowest mean rank of 1.67, whereas the models trained on the borderline-SMOTE balanced dataset have a mean rank of 1.86. The models trained on the original dataset with imbalanced classes have a mean rank of 2.47 (Table 6).

Table 6. Friedman test: ORIGINAL DATA and SMOTE

	ORGDATA	SMOTE	BLSMOTE	Rank
ORGDATA	1.00	0.00	0.00	2.47
SMOTE	0.00	1.00	0.01	1.67
BLSMOTE	0.00	0.01	1.00	1.86

6 Conclusion

Software sentiment analysis is a field with huge potential with applications ranging from evaluating project progress [4] to evaluating app reviews. Automated sentiment analysis can help save a lot of human labor and time. Various word embedding approaches, class balancing techniques, feature selection techniques, and numerous classifiers were used in this research. The goal of this paper is to improve the accuracy of sentiment analysis and help future researchers select the right combination of ML techniques for sentiment analysis. The results of this research can be compared with the results of software sentiment analysis using different classification techniques and help understand the merits and demerits of each technique. The key conclusions presented in this paper are:

- Models trained using GW2V and GLOVE outperformed models trained with other word-embedding techniques.
- The models with the Extra Trees classifier were markedly better than other classifiers at sentiment classification.
- The use of feature selection techniques resulted in a regression in performance, with the original set of features being the best choice.
- Class balancing helped improve the sentiment analysis models, with SMOTE achieving better results than BL-SMOTE.

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

References

1. Biswas, E., Vijay-Shanker, K., Pollock, L.: Exploring word embedding techniques to improve sentiment analysis of software engineering texts. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp. 68–78 (2019)
2. Islam, M.R., Zibran, M.F.: Leveraging automated sentiment analysis in software engineering. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 203–214 (2017)
3. Jindal, R., Malhotra, R., Jain, A.: Software defect prediction using neural networks. In: Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization, pp. 1–6. IEEE (2014)
4. Jurado, F., Rodriguez, P.: Sentiment analysis in monitoring software development processes: an exploratory case study on github's project issues. J. Syst. Softw. **104**, 82–89 (2015)
5. Kumar, L., Kumar, M., Murthy, L.B., Misra, S., Kocher, V., Padmanabhuni, S.: 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), pp. 477–484. IEEE (2021)
6. Lin, B., Zampetti, F., Bavota, G., Di Penta, M., Lanza, M., Oliveto, R.: Sentiment analysis for software engineering: how far can we go? In: Proceedings of the 40th International Conference on Software Engineering, pp. 94–104. ICSE 2018, Association for Computing Machinery, New York, NY, USA (2018)
7. Sari, G.I.P., Siahaan, D.O.: 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)