Analyzing Hybrid Feature Representations for Improved Multiclass Bug Severity Classification

Kamthorn Sarawan
Faculty of Informatics, Mahasarakham University, Kantarawichai, Mahasarakham, Thailand
65011294001@msu.ac.th

Jantima Polpinij
Faculty of Informatics, Mahasarakham University, Kantarawichai, Mahasarakham, Thailand
jantima.p@msu.ac.th (corresponding author)

Gamgarn Somprasertsri
Faculty of Informatics, Mahasarakham University, Kantarawichai, Mahasarakham, Thailand
gamgarns@msu.ac.th

Bancha Luaphol
Faculty of Administrative Science, Kalasin University, Kalasin, Thailand
bancha.lu@ksu.ac.th

Received: 22 March 2025 | Revised: 16 May 2025 and 18 May 2025 | Accepted: 21 May 2025

 DOI: https://doi.org/10.48084/etasr.11090

ABSTRACT

The severity classification of software bugs plays a crucial role in the field of software maintenance, as it enables developers to prioritize issues. The present study investigates the effectiveness of hybrid feature representations in improving multiclass bug severity classification using Mozilla bug reports, which contain five predefined severity levels. This study explores the impact of integrating traditional statistical features—Term Frequency-Inverse Document Frequency (TF-IDF)—with contextual word embeddings, including Word2Vec, FastText, and Bidirectional Encoder Representations from Transformers (BERT), to enhance classification performance. Additionally, an assessment is conducted to determine the influence of feature selection techniques, including no selection (all features), Least Absolute Shrinkage and Selection Operator (LASSO), and Principal Component Analysis (PCA), on model performance and training efficiency. The classification performance is measured using three machine learning models: Logistic Regression (LR), Support Vector Machine (SVM), and Random Forest (RF). The results demonstrate that incorporating word embeddings with TF-IDF consistently improves the performance of LR across all cases, achieving an accuracy range of 65.95%–66.15%, compared to 65.69% with TF-IDF alone. Furthermore, applying LASSO for feature selection has been shown to significantly reduce training time while enhancing the performance of SVM. However, the efficacy of hybrid feature representations was found to be less effective for RF. These findings highlight the benefits of hybrid feature representations and feature selection techniques in text-based multiclass classification tasks. This research provides valuable insights for optimizing bug severity classification and can be extended to other domains requiring effective text classification strategies.

Keywords-bug severity classification; hybrid feature representation; word embeddings; feature selection; machine learning

I. INTRODUCTION

Software bug tracking plays a vital role in large-scale software development, particularly in open-source ecosystems [1], serving as a centralized platform for reporting, describing, and managing issues [2-4]. Once a bug report is submitted, it typically undergoes manual triage, during which experienced developers determine its severity and assign it accordingly [5-7]. However, the growing volume of reports makes manual classification a bottleneck [2, 5], prompting researchers to automate bug severity classification using various Natural Language Processing (NLP) techniques. These studies include

binary (severe vs. non-severe) [8-12] and multiclass classification approaches [13-18], with the objective of assigning bugs to predefined severity levels (e.g., blocker, critical, major, minor, trivial). Multiclass classification poses significant challenges due to label subjectivity, class imbalance, and the subtle differences between severity levels. As illustrated in Figure 1, a bug report typically contains structured and unstructured information, including a summary, a detailed description, reproduction steps, and expected behavior. However, only designated triagers can assign severity levels [19], which are based on the impact of a bug on system functionality. Higher severity indicates critical disruption, whereas lower severity suggests minor issues.

Recent studies have introduced advanced models to improve severity classification. For example, authors in [13] proposed RepresentThemAll, a universal representation using contrastive learning. Authors in [14] introduced Knowledge-Infused Contrastive Learning (KICL), combining domain-specific pretraining with contrastive methods. Authors in [17] enhanced severity prediction using Q&A data from Stack Overflow, whereas authors in [18] applied Pareto-based feature selection (BFSp). Additionally, authors in [20] utilized topic modeling and K-Nearest Neighbors (KNN) with developer-related metadata for prediction and fixer recommendation. A summary of these resent studies is presented in Table I.

TABLE I. SUMMARY OF PREVIOUS STUDIES

Study	Approach	Dataset	Techniques	Key Findings
[13]	Universal bug report representation for multiple tasks	Mozilla, Eclipse, GCC, NetBeans	Pre-trained transformer model (RepresentTheMAll)	Achieved state-of-the-art performance across multiple bug-related tasks, including severity prediction
[14]	Domain-specific representation learning for severity classification	Mozilla, Eclipse, GCC, NetBeans	Pre-trained model (KICL), contrastive learning, knowledge-intensified masked language modeling	Outperformed six baselines, achieving up to 30.68% improvement in weighted F1 score
[17]	Incorporating Q&A data from Stack Overflow for severity prediction	Mozilla, Eclipse, GCC	Logistic Regression (LR), Naïve Bayes, KNN, Long Short-Term Memory (LSTM)	Improved F-measure by over 20% compared to traditional models
[18]	Feature selection for severity classification	Mozilla, Eclipse, GCC	Pareto optimality, Mutual Information (MI), χ^2 independence test	Enhanced feature selection efficiency, maintaining high classification accuracy
[20]	Severity prediction and fixer recommendation	Mozilla, Eclipse, NetBeans, GCC, OpenOffice	Modified REP algorithm, KNN, topic modeling	Improved severity prediction accuracy using topic-based similarity search

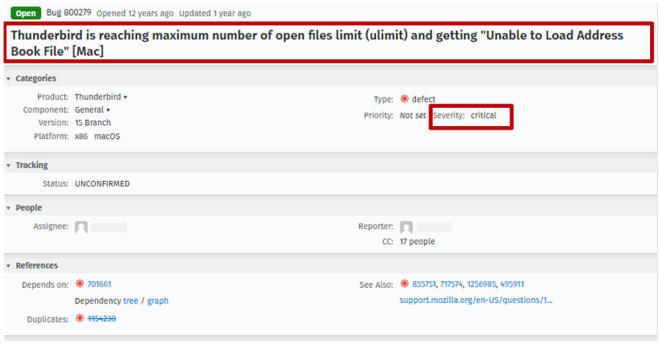

Fig. 1. Bug report from a bug tracking system.

In this study, we investigate how hybrid feature representations, combining Term Frequency-Inverse Document Frequency (TF-IDF) with contextual embeddings such as FastText, Word2Vec, and Bidirectional Encoder Representations from Transformers (BERT), affect the performance of traditional machine learning models in multiclass bug severity classification. We also explore three feature selection strategies: Least Absolute Shrinkage and Selection Operator (LASSO), Principal Component Analysis (PCA), and no selection. The experiments are conducted on a large dataset of Mozilla bug reports. The research aims to answer the following question: How does the integration of TF-IDF with various word embeddings influence the efficacy of machine learning models in multiclass bug severity classification? Our key contributions are as follows:

1. A systematic analysis of hybrid text representations for severity prediction.
2. An evaluation of feature selection methods for dimensionality reduction and their impact on model performance.
3. Practical insights into efficient, low-resource classification models that support real-world bug triage systems.

II. METHODOLOGY

A. *Dataset and Preprocessing*

The dataset utilized in this study is sourced from Mozilla bug reports, originally published in [21], and consolidates reports from various projects, including Core, Firefox, Thunderbird, and Bugzilla, into a unified dataset. In our previous work [10], we used this dataset for binary severity classification, whereas in the current study, we adopted a multiclass classification approach that preserves all five original severity levels: trivial, minor, major, critical, and blocker. Each bug report contains a "short_desc" or "title" field, which are used as the textual input for classification. The severity label assigned to each report, following the scheme established in [12, 13], represents the output for supervised learning. To ensure a balanced and effective evaluation, the dataset is partitioned into 80% for training and 20% for testing using stratified sampling to preserve class distribution.

The data preprocessing procedure employed in this study follows a moderate cleaning approach rather than an overly stringent one. This decision is based on the consideration that

certain elements, such as URLs and stop words, serve as crucial features in the classification process and should be retained. The preprocessing steps primarily involve converting all text to lowercase and removing punctuation, ensuring consistency while preserving essential information for subsequent analysis. These choices were determined through empirical observation of their impact on model performance. The distribution of severity levels across the training and testing partitions is presented in Table II.

TABLE II. DISTRIBUTION OF BUG SEVERITY LEVELS IN TRAINING AND TESTING SETS

Dataset	trivial	minor	major	critical	blocker	Total
Training	2831	6721	13734	15386	811	39483
Testing	672	1677	3483	3826	213	9871
Total	3503	8398	17217	19212	1024	49354

B. Term Frequency-Inverse Document Frequency and Word Embeddings

TF-IDF [22-24] is a statistical measure used in natural language processing and information retrieval to evaluate the importance of a word in a document relative to a collection of documents (corpus). It helps to reduce the influence of commonly occurring words while emphasizing domain-specific or meaningful terms. TF-IDF is widely used in text classification, search engines, and machine learning applications involving text data. TF-IDF is computed as the product of Term Frequency (TF) and Inverse Document Frequency (IDF)

$$TF - IDF(t, d) = TF(t, d) \times IDF(t) \quad (1)$$

Word embeddings [25, 26] are a type of text representation technique used in NLP to capture semantic and syntactic relationships between words in a numerical format. Unlike traditional methods, such as TF-IDF, which rely solely on word frequency, word embeddings map words into dense, continuous vector spaces where similar words are positioned closer together. These representations enable machine learning models to better understand the contextual meaning of words, improving performance in tasks such as text classification, sentiment analysis, and machine translation.

To transform textual data into numerical feature vectors, three different embedding techniques are utilized: BERT, FastText, and Word2Vec. Each of these techniques captures different aspects of textual semantics, with BERT providing contextualized embeddings, FastText incorporating sub-word information, and Word2Vec generating word-based vector representations. Following the extraction of embeddings, the mean pooling method is applied where applicable, and standardization is performed by employing the Z-score normalization technique to enhance numerical stability.

1) Bidirectional Encoder Representations from Transformers

BERT [27, 28] embeddings are generated by tokenizing the input text and passing it through a pre-trained transformer model. The model outputs contextualized word representations, from which the last hidden states are e extracted and the mean pooling is computed across all token embeddings to obtain a fixed-size sentence representation:

$$E_{BERT}(x) = \frac{1}{L}\sum_{i=1}^{L} h_i \quad (2)$$

where h_i is the hidden state of the i-th token, and L is the total number of tokens in the sequence.

Since BERT embeddings have a high-dimensional feature space, we apply Z-score normalization using StandardScaler to ensure a zero-mean and unit-variance transformation:

$$Z_{i,j} = \frac{E_{BERT,i,j} - \mu_j}{\sigma_j} \quad (3)$$

where μ_j and σ_j are the mean and standard deviation computed across training samples for feature dimension j.

2) FastText

FastText [29, 30] generates word embeddings that capture morphological structures through sub-word information. Each word in the input text is mapped to a 300-dimensional vector retrieved from a pre-trained FastText model. To obtain a sentence-level representation, we apply mean pooling over all word embeddings in the sentence:

$$E_{FastText}(x) = \frac{1}{L}\sum_{i=1}^{L} v_i \quad (4)$$

where v_i is the embedding vector of word i. Similar to BERT, we apply StandardScaler to normalize FastText embeddings:

$$Z_{i,j} = \frac{E_{FastText,i,j} - \mu_j}{\sigma_j} \quad (5)$$

3) Word2Vec

Word2Vec [31-33] embeddings are extracted using a pre-trained Word2Vec model. Each word in the input text is converted into a 300-dimensional word vector, and if the word is not present in the model's vocabulary, it is ignored. The final sentence representation is computed using mean pooling:

$$E_{Word2Vec}(x) = \frac{1}{L}\sum_{i=1}^{L} w_i \quad (6)$$

where w_i is the Word2Vec embedding of word i. Unlike BERT and FastText, Word2Vec embeddings are not normalized in the original model, so we apply Z-score normalization to standardize the feature space:

$$Z_{i,j} = \frac{E_{Word2Vec,i,j} - \mu_j}{\sigma_j} \quad (7)$$

C. Feature Selection

Feature selection is a process in machine learning that aims to improve model performance by selecting the most relevant features while reducing dimensionality. This technique helps remove redundant or irrelevant features, leading to better generalization, reduced computation time, and lower risk of overfitting. In text classification tasks, where high-dimensional representations such as TF-IDF and word embeddings are used, feature selection is particularly important for optimizing model efficiency and interpretability. The feature selection methods considered in this work include:

- LASSO [34, 35] is a regression-based feature selection method that applies L1 regularization to penalize the absolute values of model coefficients. This penalty forces some coefficients to shrink to zero, effectively eliminating

less important features and promoting sparsity in the feature space. In this study, LASSO is implemented using LogisticRegression(penalty='l1', solver='liblinear', max_iter=1000), where the liblinear solver supports L1 regularization and ensures stable convergence, particularly for high-dimensional text data. LASSO is particularly beneficial for selecting a subset of the most relevant features while reducing overfitting in classification models.

- PCA [36, 37] is a dimensionality reduction technique that transforms the original feature space into a smaller set of uncorrelated components, known as principal components. These components are ordered based on the variance they capture, with the goal of preserving as much information as possible while reducing the feature space complexity. In this study, PCA is implemented using PCA(n_components=0.95, svd_solver='full'), where n_components=0.95 retains 95% of the variance in the data, ensuring that the most informative features are preserved. While PCA is effective in reducing redundancy and improving computational efficiency, it may result in loss of interpretability since transformed features are linear combinations of the original variables.

D. Experimental Design

This study employs two primary experimental approaches to evaluate the effectiveness of different text representations and their impact on multiclass bug severity classification. The experiments are designed to compare standalone feature representations with hybrid feature combinations, while also assessing the influence of feature selection techniques. Both approaches utilize three machine learning models: LR, Support Vector Machine (SVM), and Random Forest (RF).

In the first approach, individual feature representations are used to evaluate their standalone effectiveness. The study examines three feature selection methods: no feature selection (baseline comparison), LASSO, and PCA. The text representation considered in this approach is TF-IDF. Each feature representation is directly fed into the machine learning models to assess its performance in bug severity classification. This experiment establishes a baseline comparison, providing insights into how different feature extraction techniques impact classification accuracy.

The second approach focuses on combining TF-IDF with various word embeddings to explore whether hybrid representations improve classification performance. The following feature combinations are evaluated: TF-IDF + Word2Vec, TF-IDF + FastText, TF-IDF + BERT, and TF-IDF + Word2Vec + FastText + BERT, as defined in (2) – (7). To address the high dimensionality introduced by concatenating TF-IDF with embeddings, feature selection techniques are applied to each hybrid representation. Feature selection is performed in the same manner as in the first experiment. The results will help determine whether integrating traditional frequency-based features with semantic-rich embeddings enhances model effectiveness and computational efficiency.

The hybrid feature representation is constructed using horizontal stacking (concatenation):

$$X_{hybrid} = \text{hstack}(X_{tfidf},\ X_{embed}) \quad (8)$$

$$X_{hybrid} \in R^{N \times (d_1 + d_2)} \quad (9)$$

where:

- N is the number of bug reports (samples).
- d_1 is the dimensionality of the TF-IDF features.
- d_2 is the dimensionality of the word embedding features.
- R denotes the set of real numbers, indicating that the matrix contains real-valued elements.
- X_{hybrid} is the resulting hybrid feature matrix, with each row representing a combined feature vector for a single bug report.

E. Classifier Models

1) Logistic Regression

LR [10, 38] is a widely used classification algorithm that models the probability of a given input belonging to a particular class using the logistic (sigmoid) function. It is effective for linearly separable problems and is interpretable due to its probabilistic output. In this study, we use solver='liblinear', max_iter=1000, where the liblinear solver is well-suited for smaller datasets and supports L1 and L2 regularization to prevent overfitting. The maximum iterations (1000) ensure convergence during training, particularly when handling high-dimensional text data.

2) Support Vector Machine

SVM [10, 34] is a powerful supervised learning algorithm that finds an optimal hyperplane to maximize the margin between different classes. It is particularly effective for high-dimensional text classification tasks. In this study, we use kernel='rbf', probability=True, where the Radial Basis Function (RBF) [9] kernel allows the model to handle non-linearly separable data by mapping inputs into a higher-dimensional space. The probability=True parameter enables the model to estimate class probabilities, which is useful for confidence-based decision-making.

3) Random Forest

RF [10, 39] is an ensemble learning method that constructs multiple decision trees and aggregates their predictions to enhance accuracy and reduce overfitting. It is robust against noise and performs well on high-dimensional data. In this study, we configure n_estimators=100, random_state=42, where 100 trees (n_estimators=100) ensure stable performance, and the random state (42) guarantees reproducibility. The model's ability to capture complex decision boundaries makes it a strong candidate for multiclass bug severity classification.

F. Evaluations

To assess the performance of the models, we utilize accuracy and F1 score as evaluation metrics. Given that the problem involves multiclass classification, we adopt the weighted average approach [13, 14] to account for class imbalances. The metrics are formally defined as follows:

$$Accuracy = \frac{\sum_{i=1}^{C} TP_i}{N} \quad (10)$$

$$Precision_{weigthed} = \sum_{i=1}^{C} w_i \cdot \frac{TP_i}{TP_i + FP_i} \quad (11)$$

$$Recall_{weigthed} = \sum_{i=1}^{C} w_i \cdot \frac{TP_i}{TP_i + FN_i} \quad (12)$$

$$F1_{weigthed} = \sum_{i=1}^{C} w_i \cdot \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i} \quad (13)$$

where:

- C is the number of classes.
- N is the total number of instances in the dataset.
- $w_i = \frac{N_i}{N}$, where N_i is the number of true instances in class i.
- TP_i denotes the true positives for class i.
- FP_i denotes the false positives for class i.
- FN_i denotes the false negatives for class i.

The hybrid feature representation framework for bug severity prediction is illustrated in Figure 2. Additionally, we compare the training time across different models and feature selection methods. This analysis facilitates the evaluation of the computational efficiency of each approach, thereby providing insights into the trade-off between model performance and training cost. Specifically, we assess the impact of different feature selection techniques (LASSO, PCA, no selection) and text representations (TF-IDF, Word2Vec, FastText, BERT, and hybrid combinations) on training time.

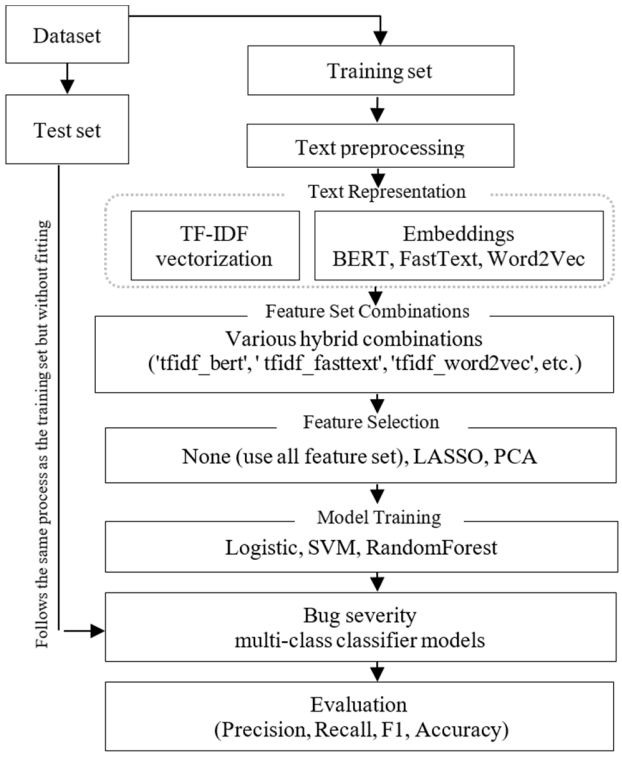

Fig. 2. Hybrid feature representation framework for bug severity prediction.

III. RESULTS AND ANALYSIS

The primary goal of our experiments is to evaluate how different feature representations and selection techniques affect the classification of bug severity into five predefined categories. In this study, we compare the baseline TF-IDF features with hybrid combinations incorporating contextual embeddings (Word2Vec, FastText, and BERT), and assess model performance using accuracy, weighted F1 score, and training time. The input to all models is the preprocessed "short_desc" field, whereas the output is the corresponding severity class. The results of the first experiment, which utilizes TF-IDF as the sole feature representation with various feature selection methods, are presented in Table III.

TABLE III. PERFORMANCE COMPARISON OF MACHINE LEARNING MODELS USING TF-IDF WITH DIFFERENT FEATURE SELECTION METHODS

Feature selection	Model	F1 (%)	Acc (%)	Time (s)
None	LR	64.17	65.69	1.96
None	SVM	64.84	66.67	3363.12
None	RF	60.64	64.03	162.81
LASSO	LR	63.9	65.58	0.61
LASSO	SVM	64.98	66.70	2367.69
LASSO	RF	61.57	64.27	73.18
PCA	LR	64.17	65.69	0.98
PCA	SVM	64.84	66.67	3075.57
PCA	RF	60.64	64.03	147.88

The results presented in Table III demonstrate the performance of the models under different feature selection methods. Without feature selection, SVM achieves the highest accuracy (66.67%) and F1 score (64.84%), indicating its strong performance in bug severity classification. However, it also requires the longest training time (3363.12 s), suggesting that SVM is computationally expensive when trained on high-dimensional TF-IDF features. In contrast, LR achieves competitive accuracy (65.69%) with significantly lower training time (1.96 s), making it a more computationally efficient choice. RF shows the lowest accuracy (64.03%) and F1 score (60.64%), whereas its training time (162.81 s) remains moderate compared to SVM. The application of LASSO feature selection leads to a notable reduction in training time across all models, particularly in LR (0.61 s) and RF (73.18 s), while maintaining similar classification performance. For SVM, LASSO reduces the training time from 3363.12 s to 2367.69 s, making it computationally more feasible without significantly affecting accuracy (66.70%). These results indicate that LASSO effectively removes redundant features, allowing models to train faster while preserving their predictive power. Similarly, PCA feature selection also helps reduce training time, although its impact is less pronounced compared to LASSO. For LR, PCA maintains the same accuracy (65.69%) and F1 score (64.17%) while slightly reducing training time (0.98 s). In the case of SVM, PCA reduces training time from 3363.12 s to 3075.57 s, but the improvement is marginal. RF experiences a moderate reduction in training time (from 162.81 s to 147.88 s) without affecting performance. These findings suggest that while PCA helps reduce feature dimensionality, LASSO is more effective in

reducing computational overhead while maintaining classification [36, 40].

The results of the second experiment, which utilizes TF-IDF as the base representation and concatenates it with various word embeddings, are presented in Table IV. The performance of different machine learning models when combining TF-IDF (T) with various word embeddings (Word2Vec (W), FastText (FT), and BERT (BE)) is investigated. The evaluation focuses on F1 score, accuracy, and training time, allowing for a comparative analysis of different hybrid feature sets and feature selection methods.

TABLE IV. PERFORMANCE COMPARISON OF MACHINE LEARNING MODELS USING HYBRID FEATURE REPRESENTATIONS (TF-IDF + WORD EMBEDDINGS) WITH DIFFERENT FEATURE SELECTION METHODS

Feature set	Feature selection	Model	F1 (%)	Acc (%)	Time (s)
T+W	None	LR	64.74	66.13	55.89
T+W	None	SVM	55.47	61.38	6689.21
T+W	None	RF	57.06	60.47	159.16
T+W	LASSO	LR	64.24	65.77	17.15
T+W	LASSO	SVM	57.21	62.43	3853.86
T+W	LASSO	RF	57.42	60.66	146.16
T+W	PCA	LR	63.73	65.43	116.38
T+W	PCA	SVM	59.66	63.61	7407.12
T+W	PCA	RF	54.42	59.89	264.01
T+FT	None	LR	64.88	66.15	462.74
T+FT	None	SVM	60.17	63.06	8921.53
T+FT	None	RF	54.09	59.09	199.95
T+FT	LASSO	LR	64.62	65.92	298.22
T+FT	LASSO	SVM	60.15	63.04	7953.32
T+FT	LASSO	RF	56.08	60.57	260.07
T+FT	PCA	LR	58.25	60.76	17.02
T+FT	PCA	SVM	59.51	62.47	2432.85
T+FT	PCA	RF	53.18	57.62	101.56
T+BE	None	LR	65.07	66.06	1040.15
T+BE	None	SVM	63.35	65.32	21774.04
T+BE	None	RF	54.52	59.54	272.72
T+BE	LASSO	LR	64.94	65.92	754.46
T+BE	LASSO	SVM	63.34	65.32	20385.1
T+BE	LASSO	RF	56.35	60.73	469.66
T+BE	PCA	LR	60.01	61.97	31.67
T+BE	PCA	SVM	62.81	64.85	3648.03
T+BE	PCA	RF	51.10	57.41	135.33
T+W+FT+BE	None	LR	65.08	65.95	3562.43
T+W+FT+BE	None	SVM	64.19	66.02	34127.96
T+W+FT+BE	None	RF	57.37	61.42	389.31
T+W+FT+BE	LASSO	LR	64.67	65.59	1629.05
T+W+FT+BE	LASSO	SVM	64.41	66.19	28579.87
T+W+FT+BE	LASSO	RF	58.28	62.00	642.55
T+W+FT+BE	PCA	LR	61.73	63.33	43.48
T+W+FT+BE	PCA	SVM	63.91	65.76	3896
T+W+FT+BE	PCA	RF	52.04	58.16	121.21

An analysis of the feature sets reveals that LR, without feature selection, consistently achieves the highest accuracy across most configurations (65.95%-66.15%). Notably, T+FT with LR achieves 66.15% accuracy, whereas T+W reaches 66.13%, indicating that hybridizing TF-IDF with word embeddings enhances classification performance. Moreover, the highest overall accuracy (66.19%) is observed when combining all embeddings (T+W+FT+BE) with SVM, although at a substantial computational cost (28,579.87 s). Feature selection methods play a significant role in reducing training time while maintaining competitive performance. LASSO proves to be more effective than PCA, as it reduces training time significantly without causing major drops in accuracy. For instance, LASSO reduces LR's training time for T+BE from 1040.15 s to 754.46 s while maintaining nearly the same accuracy (65.92%). In contrast, PCA leads to a more significant drop in accuracy, especially for RF, where performance declines to 57.41% for T+BE and 58.16% for T+W+FT+BE. This indicates that while PCA effectively reduces dimensionality, it may also remove critical features needed for optimal classification.

Furthermore, incorporating word embeddings with TF-IDF consistently improves the performance of LR across all cases, achieving an accuracy range of 65.95%–66.15%. In contrast, using TF-IDF alone results in a lower accuracy of 65.69%, indicating that the integration of embeddings enhances the model's ability to capture meaningful textual patterns and improve classification performance.

Overall, these results suggest that hybrid feature representations incorporating TF-IDF with embeddings contribute to improved classification accuracy. However, the computational cost is a major trade-off, especially for SVM, where training times are substantially higher than those for other models. LR emerges as the most balanced model, providing strong classification performance with significantly lower training costs. Future research could explore more efficient dimensionality reduction techniques tailored to hybrid embeddings to further optimize both accuracy and computational efficiency.

Figure 3 presents a boxplot illustrating the distribution of F1 scores across different feature sets. The findings indicate that the performance of each feature set is significantly different. The T+W+FT+BE feature set has the highest median F1 score, indicating that the most accurate prediction is achieved when all of these features are combined. The T+BE and T+W sets also perform well; however, their aggregate scores are slightly lower and their scores vary more. Conversely, the T+FT set has the lowest median F1 score, which implies that this combination may not be as effective in classification tasks.

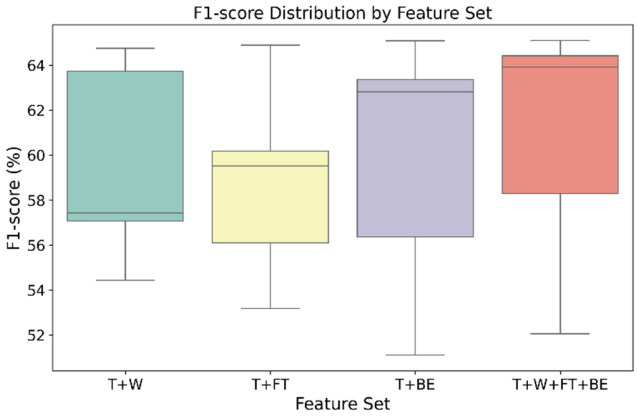

Fig. 3. F1 score distribution by feature set.

The impact of various feature selection methods on classification efficiency is illustrated by the F1 score distribution in the boxplot depicted in Figure 4. The median F1 scores are highest for the no selection and LASSO approaches. This indicates that the retention of classification-relevant information relies on either maintaining most features or selecting penalized ones. Conversely, PCA exhibits a diminished median F1 score and an expanded range, indicating that the reduction in dimensionality may result in the loss of critical data necessary for accurate predictions. This study demonstrates that the selection of appropriate features can enhance a model's usefulness; nonetheless, it is essential that these features are meticulously picked to provide adequate dimensionality reduction and predictive accuracy.

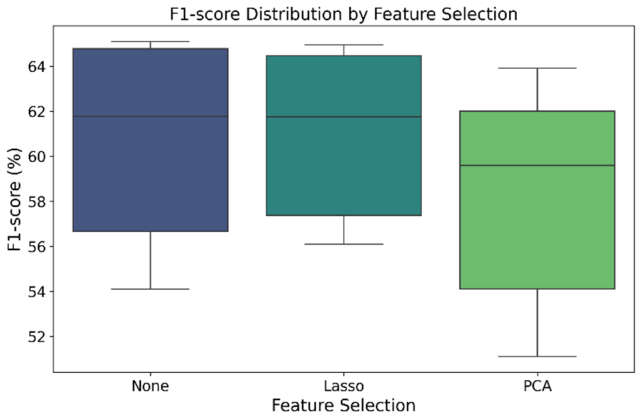

Fig. 4. F1 score distribution by feature selection method.

Figure 5 presents a boxplot of F1 score distributions across different classifier models, highlighting their comparative effectiveness. LR achieves the highest median F1 score, demonstrating its strong and stable performance across different feature sets. SVM follows closely, but with a wider distribution, indicating that its effectiveness heavily depends on the selected feature set and hyperparameters. RF exhibits the lowest median F1 score and the highest variability, suggesting that it struggles with capturing relevant patterns in the dataset. This analysis suggests that LR is the most reliable model for this classification task, whereas SVM may perform well with careful tuning, and RF might require further optimization to improve its predictive power.

Additionally, we compare our findings with those of previous studies that focus specifically on multiclass bug severity prediction. For the purpose of this comparison, the best-performing F1 score derived from our own experiments is reported. This score corresponds to the combination of TF-IDF, BERT, FastText, and Word2Vec without feature selection using LR. It is important to clarify that the values from other studies shown in Table V are directly cited from the original papers and were not reproduced or re-evaluated on our dataset. Specifically, the result from RepresentThemAll [13] represents the F1 score reported in their paper, which was based on their own dataset and experimental setting. Therefore, while this comparative analysis provides a general benchmark, it is not intended as a strict side-by-side evaluation due to differences in datasets, feature engineering, and model architecture. Furthermore, those prior works employed a range of methods, including both traditional machine learning approaches [17] and advanced deep learning techniques [7-8, 15], particularly transformer-based models [13].

TABLE V. COMPARISON OF BUG SEVERITY PREDICTION MODELS WITH PREVIOUS WORK

Model	P (%)	R (%)	F1 (%)	Acc (%)
BSP-QASO [17]	54.21	53.47	52.42	53.47
DNNSPBP [8]	54.83	54.18	54.12	54.18
PPWGCN [7]	55.54	55.24	55.22	55.24
CLBSP [15]	61.34	62.77	61.88	62.77
RepresentThemAll [13]	65.11	64.72	64.73	64.72
T+W+FT+BE (LR)	65.21	65.95	65.08	65.95

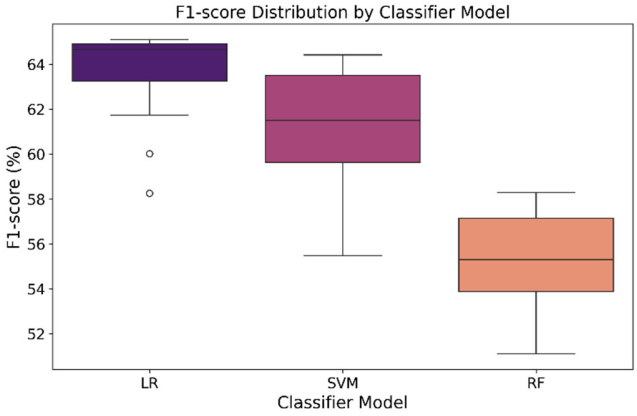

Fig. 5. F1 score distribution by classifier model.

IV. CONCLUSION

This study presents an approach that leverages hybrid feature representations to enhance the performance of multiclass bug severity classification using bug reports from Mozilla, which contain five predefined severity levels. The results demonstrate that incorporating hybrid feature representations combining statistical and contextual features, including Word2Vec, FastText, and Bidirectional Encoder Representations from Transformers (BERT), significantly improves classification performance compared to traditional methods relying solely on Term Frequency-Inverse Document Frequency (TF-IDF) or conventional word embeddings. Notably, the Logistic Regression (LR) model consistently achieved superior results when incorporating word embeddings. Furthermore, an analysis was conducted to assess the impact of different feature selection techniques, including the utilization of all features, Least Absolute Shrinkage and Selection Operator (LASSO), and Principal Component Analysis (PCA). The findings indicate that these techniques effectively reduce model training time without significantly affecting classification performance. Specifically, the LASSO-based feature selection method yielded both improved classification performance and reduced training time for the Support Vector Machine (SVM) model. However, hybrid feature representations were found to be less suitable for models such as the Random Forest (RF). These findings provide valuable insights for addressing similar challenges in

other domains within the scope of text-based multiclass classification.

The results of the study indicate that integrating traditional statistical features with contextual embeddings can effectively enhance classification accuracy in multiclass bug severity prediction tasks. LR models demonstrate consistent and efficient performance when hybrid features are used. Additionally, feature selection with LASSO significantly improves training efficiency, especially for high-dimensional models like SVM. These findings are practically valuable in real-world bug tracking systems, where fast and accurate classification is essential. Future work may extend this research by incorporating additional bug report fields (e.g., comments, reproduction steps), exploring deep learning architectures, or applying transfer learning from domain-specific pretrained models.

ACKNOWLEDGEMENT

This research project was fully financial supported by Mahasarakham University.

REFERENCES

[1] C. Vassallo, G. Grano, F. Palomba, H. C. Gall, and A. Bacchelli, "A large-scale empirical exploration on refactoring activities in open source software projects," *Science of Computer Programming*, vol. 180, pp. 1–15, Jul. 2019, https://doi.org/10.1016/j.scico.2019.05.002.

[2] S. Akbarinasaji, B. Caglayan, and A. Bener, "Predicting bug-fixing time: A replication study using an open source software project," *Journal of Systems and Software*, vol. 136, pp. 173–186, Feb. 2018, https://doi.org/10.1016/j.jss.2017.02.021.

[3] N. Serrano and I. Ciordia, "Bugzilla, ITracker, and other bug trackers," *IEEE Software*, vol. 22, no. 2, pp. 11–13, Mar. 2005, https://doi.org/10.1109/MS.2005.32.

[4] H. M. Tran, S. T. Le, S. V. Nguyen, and P. T. Ho, "An Analysis of Software Bug Reports Using Machine Learning Techniques," *SN Computer Science*, vol. 1, no. 1, Jun. 2019, Art. no. 4, https://doi.org/10.1007/s42979-019-0004-1.

[5] D.-G. Lee and Y.-S. Seo, "Improving bug report triage performance using artificial intelligence based document generation model," *Human-centric Computing and Information Sciences*, vol. 10, no. 1, Jun. 2020, Art. no. 26, https://doi.org/10.1186/s13673-020-00229-7.

[6] T. S. S. Angel, G. S. Kumar, V. M. Sehgal, and G. Nayak, "Effective Bug Processing and Tracking System," *Journal of Computational and Theoretical Nanoscience*, vol. 15, no. 8, pp. 2604–2606, Aug. 2018, https://doi.org/10.1166/jctn.2018.7506.

[7] S. Fang, Y. Tan, T. Zhang, Z. Xu, and H. Liu, "Effective Prediction of Bug-Fixing Priority via Weighted Graph Convolutional Networks," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 563–574, Jun. 2021, https://doi.org/10.1109/TR.2021.3074412.

[8] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep Neural Network-Based Severity Prediction of Bug Reports," *IEEE Access*, vol. 7, pp. 46846–46857, 2019, https://doi.org/10.1109/ACCESS.2019.2909746.

[9] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *2010 7th IEEE Working Conference on Mining Software Repositories*, Cape Town, South Africa, 2010, pp. 1–10, https://doi.org/10.1109/MSR.2010.5463284.

[10] K. Sarawan, J. Polpinij, and B. Luaphol, "Machine Learning-Based Methods for Identifying Bug Severity Level from Bug Reports," in *Proceedings of the 19th International Conference on Computing and Information Technology*, Bangkok, Thailand, 2023, pp. 199–208, https://doi.org/10.1007/978-3-031-30474-3_17.

[11] N. K.-S. Roy and B. Rossi, "Towards an Improvement of Bug Severity Classification," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, Verona, Italy, 2014, pp. 269–276, https://doi.org/10.1109/SEAA.2014.51.

[12] A.-H. Dao and C.-Z. Yang, "Severity Prediction for Bug Reports Using Multi-Aspect Features: A Deep Learning Approach," *Mathematics*, vol. 9, no. 14, Jul. 2021, Art. no. 1644, https://doi.org/10.3390/math9141644.

[13] S. Fang, T. Zhang, Y. Tan, H. Jiang, X. Xia, and X. Sun, "RepresentThemAll: A Universal Learning Representation of Bug Reports," in *2023 IEEE/ACM 45th International Conference on Software Engineering*, Melbourne, Australia, 2023, pp. 602–614, https://doi.org/10.1109/ICSE48619.2023.00060.

[14] Y. Wei, C. Zhang, and T. Ren, "Improving Bug Severity Prediction With Domain-Specific Representation Learning," *IEEE Access*, vol. 11, pp. 62829–62839, 2023, https://doi.org/10.1109/ACCESS.2023.3279205.

[15] J. Kim and G. Yang, "Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm," *IEEE Access*, vol. 10, pp. 94643–94651, 2022, https://doi.org/10.1109/ACCESS.2022.3204689.

[16] A. Kukkar, R. Mohana, and Y. Kumar, "Does bug report summarization help in enhancing the accuracy of bug severity classification?," *Procedia Computer Science*, vol. 167, pp. 1345–1353, 2020, https://doi.org/10.1016/j.procs.2020.03.345.

[17] Y. Tan, S. Xu, Z. Wang, T. Zhang, Z. Xu, and X. Luo, "Bug severity prediction using question-and-answer pairs from Stack Overflow," *Journal of Systems and Software*, vol. 165, Jul. 2020, Art. no. 110567, https://doi.org/10.1016/j.jss.2020.110567.

[18] S. Sharmin, F. Aktar, A. A. Ali, M. A. H. Khan, and M. Shoyaib, "BFSp: A feature selection method for bug severity classification," in *2017 IEEE Region 10 Humanitarian Technology Conference*, Dhaka, Bangladesh, 2017, pp. 750–754, https://doi.org/10.1109/R10-HTC.2017.8289066.

[19] L. A. F. Gomes, R. da S. Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: A survey and research opportunities," *Information and Software Technology*, vol. 115, pp. 58–78, Nov. 2019, https://doi.org/10.1016/j.infsof.2019.07.009.

[20] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, Jul. 2016, https://doi.org/10.1016/j.jss.2016.02.034.

[21] A. Lamkanfi, J. Pérez, and S. Demeyer, "The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information," in *2013 10th Working Conference on Mining Software Repositories*, San Francisco, CA, USA, 2013, pp. 203–206, https://doi.org/10.1109/MSR.2013.6624028.

[22] K. Sparck Jones, "A Statistical Interpretation of Term Specificity and its Application in Retrieval," *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, Jan. 1972, https://doi.org/10.1108/eb026526.

[23] D. Ali, M. M. S. Missen, and M. Husnain, "Multiclass Event Classification from Text," *Scientific Programming*, vol. 2021, no. 1, Jan. 2021, Art. no. 6660651, https://doi.org/10.1155/2021/6660651.

[24] W. Zhang, T. Yoshida, and X. Tang, "A comparative study of TF*IDF, LSI and multi-words for text classification," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2758–2765, Mar. 2011, https://doi.org/10.1016/j.eswa.2010.08.066.

[25] D. S. Asudani, N. K. Nagwani, and P. Singh, "Impact of word embedding models on text analytics in deep learning environment: a review," *Artificial Intelligence Review*, vol. 56, no. 9, pp. 10345–10425, Sep. 2023, https://doi.org/10.1007/s10462-023-10419-1.

[26] S. Selva Birunda and R. Kanniga Devi, "A Review on Word Embedding Techniques for Text Classification," in *Innovative Data Communication Technologies and Application: Proceedings of ICIDCA 2020*, Coimbatore, India, 2021, pp. 267–281, https://doi.org/10.1007/978-981-15-9651-3_23.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*,

Minneapolis, MN, USA, 2019, pp. 4171–4186, https://doi.org/10.18653/v1/N19-1423.

[28] A. K. Chanda, "Efficacy of BERT embeddings on predicting disaster from Twitter data." arXiv, Aug. 08, 2021, https://doi.org/10.48550/arXiv.2108.10698.

[29] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, Jun. 2017, https://doi.org/10.1162/tacl_a_00051.

[30] B. Athiwaratkun, A. G. Wilson, and A. Anandkumar, "Probabilistic FastText for Multi-Sense Word Embeddings." arXiv, Jun. 07, 2018, https://doi.org/10.48550/arXiv.1806.02901.

[31] G. Di Gennaro, A. Buonanno, and F. A. N. Palmieri, "Considerations about learning Word2Vec," *The Journal of Supercomputing*, vol. 77, no. 11, pp. 12320–12335, Nov. 2021, https://doi.org/10.1007/s11227-021-03743-2.

[32] S. J. Johnson, M. R. Murty, and I. Navakanth, "A detailed review on word embedding techniques with emphasis on word2vec," *Multimedia Tools and Applications*, vol. 83, no. 13, pp. 37979–38007, Apr. 2024, https://doi.org/10.1007/s11042-023-17007-z.

[33] Y. Tang, H. Zhou, and H. Su, "Automatic Classification of Software Bug Reports Based on LDA and Word2Vec," in *2022 2nd International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology*, Nanjing, China, 2022, pp. 491–495, https://doi.org/10.1109/CEI57409.2022.9950207.

[34] K. Wang, L. Liu, C. Yuan, and Z. Wang, "Software defect prediction model based on LASSO–SVM," *Neural Computing and Applications*, vol. 33, no. 14, pp. 8249–8259, Jul. 2021, https://doi.org/10.1007/s00521-020-04960-1.

[35] R. Tibshirani, "Regression Shrinkage and Selection Via the Lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, Jan. 1996, https://doi.org/10.1111/j.2517-6161.1996.tb02080.x.

[36] S. Iqbal, R. Naseem, S. Jan, S. Alshmrany, M. Yasar, and A. Ali, "Determining Bug Prioritization Using Feature Reduction and Clustering With Classification," *IEEE Access*, vol. 8, pp. 215661–215678, 2020, https://doi.org/10.1109/ACCESS.2020.3035063.

[37] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (PCA)," *Computers & Geosciences*, vol. 19, no. 3, pp. 303–342, Mar. 1993, https://doi.org/10.1016/0098-3004(93)90090-R.

[38] Z. H. Arif and K. Cengiz, "Severity Classification for COVID-19 Infections based on Lasso-Logistic Regression Model," *International Journal of Mathematics, Statistics, and Computer Science*, vol. 1, pp. 25–32, Apr. 2023, https://doi.org/10.59543/ijmscs.v1i.7715.

[39] H. Shi, S. Liu, J. Chen, X. Li, Q. Ma, and B. Yu, "Predicting drug-target interactions using Lasso with random forest based on evolutionary information and chemical structure," *Genomics*, vol. 111, no. 6, pp. 1839–1852, Dec. 2019, https://doi.org/10.1016/j.ygeno.2018.12.007.

[40] S. R. Basha, J. K. Rani, and J. J. C. P. Yadav, "A Novel Summarization-based Approach for Feature Reduction Enhancing Text Classification Accuracy," *Engineering, Technology & Applied Science Research*, vol. 9, no. 6, pp. 5001–5005, Dec. 2019, https://doi.org/10.48084/etasr.3173.