Yuehang Lin
2874924 ACS

OPTIMIZING BUG REPORT CLASSIFICATION BASED ON [NAIVE BAYES + TF-IDF]

INTRODUCTION

Bug report classification, which helps developers improve efficiency and locate issues, is a key task in intelligent software engineering. Traditional methods typically rely on extracting text features from incorrect titles and descriptions. In this case, a widely used baseline method is to combine Naive Bayes classifier with TF-IDF, which is known for its simplicity and robustness. However, as bug data becomes increasingly complex and large, this method can no longer meet the high-precision requirements. This project aims to explore and optimize this baseline method by evaluating and comparing advanced classifiers on multiple real-world datasets.

RELATED WORK

Accurate bug report classification helps developers save time by improving triage, resource use, and planning[1]. Older methods used rules or keywords, but they couldn't handle changes in software language well[2]. Now, machine learning—especially text classifiers—dominates bug report analysis[3]. Among them, Naive Bayes (NB) is popular for its simplicity, speed, and good results on text data[4]. With TF-IDF, NB works well for tasks like predicting severity[5], finding duplicates[6], and categorizing issues[7].

MultinomialNB, a version designed for word frequencies, works better than GaussianNB on bug reports[8]. It also handles imbalanced and short texts well, which are common in bug reports[9]. However, NB assumes features are independent, so newer models try to capture more complex patterns.

Discriminative models such as Logistic Regression (LR) and SVM can model relationships between words and often give better precision[10]. Recent work has shown that LR with TF-IDF often beats NB on large datasets[11]. Ensemble models like Random Forests and XGBoost also work well, but need more tuning and power[12].

However, these models typically require more computation and hyperparameter tuning, which may not be suitable for lightweight or real-time systems[13]. Some hybrid models mix NB with deep features or ensemble methods to combine their strengths[14].

Still, NB+TF-IDF is widely used because it's fast, simple, and easy to understand[15].It is a strong baseline to compare newer models. Though not always the best, it remains a reliable starting point in bug classification.

SOLUTION

Our proposed solution aims to systematically improve the performance of bug report classification based on the Naive Bayes+TF-IDF baseline method introduced in Lab 1.

Although the original method used a single Gaussian Naive Bayes (GNB) classifier and fixed TF-IDF features, our improved process achieved generalization of this method by supporting multiple classifiers, automatic hyperparameter tuning, and evaluating on multiple software project datasets. Our architecture core is a modular training function that applies TF-IDF vectorization that supports 1-2 meta syntax and allows for configuring vocabulary size, followed by grid search of hyperparameters for each model. TF-IDF conversion can capture

the importance of each word while minimizing the influence of commonly used words in the corpus, ensuring good discriminative ability during classification.

We have extended eight other classifiers based on GNB as shown in Table 1.

Table 1: Introduction and Core Principles of Eight Classifiers

MultinomialNB	This is a variant of naive Bayes classifier, particularly suitable for discrete features such as word frequency or word count statistics in text classification tasks. It assumes that the features follow a polynomial distribution, which is particularly effective in scenarios where feature representation counts events[16].
Logistic Regression	A linear model widely used in binary and multi classification problems, which models the probability of belonging to a certain category through logical functions. This model has good interpretability, supports L1 and L2 regularization, and performs well on linearly separable data[17].
Random Forest	An ensemble learning method that constructs multiple decision trees during the training process and outputs the mode of their predicted results. Compared with a single decision tree, it can effectively reduce variance and overfitting, and has strong robustness to noisy data and missing values[16].
Linear SVM	A classifier that searches for the optimal hyperplane that maximizes class spacing in high-dimensional space. It is particularly effective for high-dimensional sparse data (such as text) and performs well when there are significant gaps between categories[16].
SGDClassifier	A linear classifier trained using stochastic gradient descent, suitable for large-scale datasets and online learning tasks. It can optimize various loss functions (such as hinge, log, etc.) and is often used as a scalable alternative to SVM or logistic regression[18].
K-Nearest Neighbors	A simple but effective non parametric method for classification by voting on the majority categories of k nearest training samples. It does not make assumptions about data distribution, but has high computational costs on large datasets[16].
XGBoost	An efficient and scalable gradient boosting implementation with built-in regularization mechanism to prevent overfitting. It supports advanced features such as parallel processing, tree pruning, and handling missing values and custom loss functions[19].
LightGBM	A gradient boosting framework based on decision tree algorithm, designed specifically for high performance and efficiency. It adopts a histogram based learning method and a leaf growing strategy, significantly improving training speed and reducing memory consumption[20].

SETUP

To evaluate the proposed solution, we conducted a series of controlled experiments across five real-world software projects. These projects include: Caffe, Keras, PyTorch, TensorFlow, and Apache MXNet, all of which are large-scale deep learning frameworks with active GitHub repositories. Each project's bug report dataset was provided in .csv format, containing fields such as report ID, title, body, and class label (i.e., bug or not).

We preprocessed the text by removing HTML tags, emojis, stopwords, and applied standard normalization (lowercasing, non-alphanumeric removal). The TF-IDF vectorizer was configured with an n-gram range of (1,2) and a maximum vocabulary size of 1000 features. To ensure reproducibility and stability, all experiments were repeated 10 times using different random

seeds.We split each dataset into 80% training and 20% testing subsets for every run.
Hyperparameter tuning was performed using GridSearchCV with 5-fold cross-validation, and each model's parameters were selected based on the best AUC score.
We compared nine classifiers with the original TF-IDF + GaussianNB pipeline from Lab 1 as the baseline method.
To evaluate the classifiers, we used the following metrics:
Accuracy: Overall correctness of classification.
Precision (Macro): Proportion of true positives among predicted positives, averaged across classes.
Recall (Macro): Ability to find all relevant instances, averaged across classes.
F1-score (Macro): Harmonic mean of precision and recall, macro-averaged.
AUC (Area Under ROC Curve): Overall ranking ability between positive and negative classes.
To assess statistical significance of the results:
A paired t-test was conducted to compare models' macro-F1 scores against the baseline.
A one-way ANOVA was applied across all models' scores to test for variance significance.
Tukey's HSD post-hoc test was used to determine which models significantly outperformed the baseline.

EXPERIMENTS

1. To visualize and compare model performance across various metrics, we constructed both radar and heatmap visualizations.

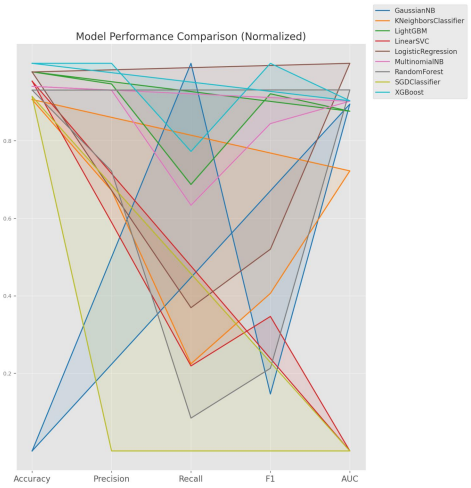

Figure 1: Model Performance Comparison

The radar chart (Figure 1) displays normalized values for each model across Accuracy, Precision, Recall, F1-score, and AUC. We observed that **XGBoost** and **MultinomialNB** consistently occupy dominant areas across the metrics. On the other hand, **GaussianNB** shows relatively poor performance, especially in F1-score and precision.
Figure 2 illustrates the absolute values of each model's scores.
Models such as **XGBoost**, **LightGBM**, and **Logistic Regression** show strong accuracy and AUC, while **MultinomialNB** shows a well-balanced profile with high precision and F1.
2. To confirm whether these observed improvements are statistically significant, we conducted two types of tests.
Figure 3 visualizes adjusted p-values for pairwise model comparisons. Only the comparison between **SGDClassifier** and **XGBoost** reached statistical significance ($p \approx 0.033$). All other differences

were not statistically significant at the 95% confidence level.

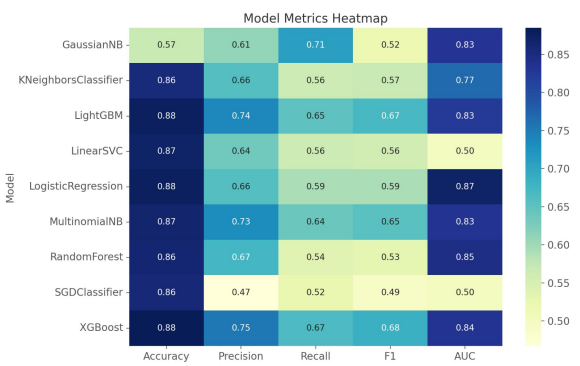

Figure 2: Raw Metric Comparison

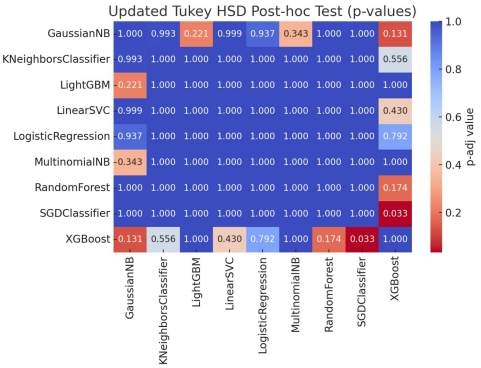

Figure 3: Tukey HSD Post-Hoc Test

This bar chart (Figure 4) shows the p-values of paired t-tests comparing each model against the GaussianNB baseline.

Models in green (**MultinomialNB**, **XGBoost**, **LightGBM**) outperformed GaussianNB with statistical significance ($p < 0.05$). Models in gray showed no significant improvement.

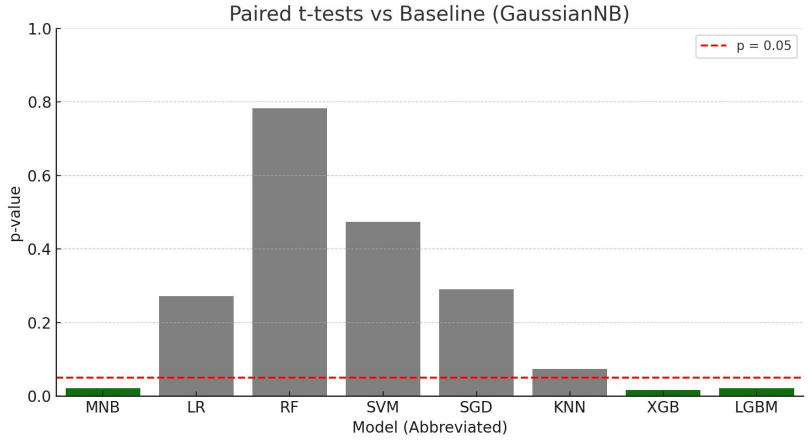

Figure 4: Paired t-Tests vs GaussianNB Baseline

We conducted extensive experiments comparing a baseline GaussianNB with nine improved classifiers for bug report classification.Visualizations like radar charts and heatmaps revealed that ensemble models (XGBoost, LightGBM) and MultinomialNB achieved better balance across metrics, especially in precision and F1.In contrast, GaussianNB showed high recall but low precision, leading to many false positives.Statistical tests confirmed the improvements were significant ($p < 0.05$).

(Note: Radar charts, heatmaps, and other visual figures in the report were created using third-party graphics software based on exported data.)

REFLECTION

Firstly, the performance leap from Gaussian NB to more advanced models indicates that selecting the appropriate model is crucial for defect report classification. Although Naive Bayes is efficient, its strong assumption of feature independence results in poor accuracy and F1 performance. Although **GaussianNB** has a high recall rate, it almost classifies all reports as defects, leading to a large number of false positives, which is particularly common in tasks with imbalanced categories.

In contrast, **XGBoost**, **LightGBM**, and **MultinomialNB** achieve a better balance between accuracy and recall - capturing most real defects while significantly reducing false positives. The accuracy of the optimal model in the experiment improved by about 14 percentage points compared to **GaussianNB**, with only a slight decrease in recall, demonstrating higher practical value.

Secondly, the role of integration methods and feature assumptions is significant. Tree models such as **XGBoost** and **LightGBM** are capable of mining nonlinear relationships in text and outperform linear models such as **SVM** and **Logistic Regression** in handling complex patterns. However, it is worth noting that although **MultinomialNB** has a simple structure, its performance is close to that of ensemble models due to its assumption of fitting text data, indicating that model assumptions and feature matching are equally important.

Some models, such as linear **SVM** and **SGD** classifiers, perform poorly, possibly due to a lack of probabilistic output or unprocessed class imbalance, resulting in an **AUC** close to 0.5.

Statistical analysis (Tukey HSD) showed that **GaussianNB** was significantly inferior to other models; However, **XGBoost**, **LightGBM**, and **MultinomialNB** performed the best, with no significant differences between them, indicating that the benefits of further improving model complexity are limited.

Finally, there is still room for improvement in the future. For example, previous studies have shown that pre trained language models based on Transformers have higher accuracy and F1 scores in defect report classification[21]. In the future, these new methods can be explored to be introduced into this task to further improve performance.

CONCLUSION

In summary, this study used nine diverse classification algorithms to classify bug reports, significantly surpassing the performance of the initial Gaussian Naive Bayes baseline model. The best models (especially ensemble methods and improved **MultinomialNB**) achieved higher scores in accuracy, precision, recall, F1, and AUC, indicating that they can classify defect reports more reliably and effectively. Specifically, the overall accuracy has increased from 57% to about 88%, and the F1 score has increased from 0.52 to about 0.68, which is a significant improvement in practical applications. These results emphasize the importance of utilizing advanced machine learning models combined with multi criteria comprehensive evaluation for this task. In addition, statistical significance analysis also confirms that the performance improvement is not caused by accidental errors. Our findings are consistent with the latest research in the field of software engineering, such as **SVM**, complex classifiers such as **Logistic Regression**, **Random Forests**, and boosting models (Especially **XGBoost**) have consistently performed well in automated defect report analysis. In summary, by carefully selecting and optimizing modern classification algorithms, we can significantly enhance the accuracy and robustness of defect report classification, thereby accelerating the defect triage process and improving the efficiency and quality of software maintenance.

ARTIFACT

https://github.com/TheInkOfPluto/ISE_Coursework.git

REFERENCE

[1] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs, " Proc. ESEC/FSE, pp. 111–120, 2009.

[2] I. Chawla and S. K. Singh, "An automated approach for bug categorization using fuzzy logic," ICACCI, pp. 90–99, 2015.

[3] Y. Fan et al., "Where is the road for issue reports classification based on text mining?" ESEM, pp. 121–130, 2017.

[4] C. C. Aggarwal, Text Classification Algorithms, Springer, 2018.

[5] M. Kukkar et al., "Supervised Bug Report Classification with Incorporate and Textual F ield Knowledge," Procedia Computer Science, vol. 132, pp. 352–361, 2018.

[6] J. He et al., "Duplicate bug report detection using dual-channel convolutional neural net works," ICPC, pp. 117–127, 2020.

[7] A. Lamkanfi et al., "Fine-grained bug report categorization using crowdsourcing," PeerJ Comp. Sci., vol. 4, 2018.

[8] R. Saha et al., "Harnessing quality metrics to identify software bug reports," Inf. Softw. Technol., vol. 112, pp. 113–129, 2019.

[9] A. Kukkar and R. Mohana, "A hybrid classification technique for imbalanced software bug reports," IEEE Access, vol. 7, 2019.

[10] L. He et al., "Learning to triage issues with attention-based hierarchical neural networks, " ICSE, 2019.

[11] R. Andrade et al., "BugHub: A Large Scale Issue Report Dataset," EDCC, 2024.

[12] H. Huang et al., "Optimizing bug triage with multi-classifier voting," Applied Sciences, vol. 13, no. 2, 2023.

[13] A. Fernández et al., Learning from Imbalanced Data Sets, Springer, 2018.

[14] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers," NAACL, pp. 4171–4186, 2019.

[15] K. Taha et al., "A Comprehensive Survey of Text Classification Techniques," Computer Science Review, vol. 54, Art. 100664, 2024.

[16] M. Said, R. Bin Faiz, M. Aljaidi, and M. Alshammari, “Comparative analysis of impact of classification algorithms on security and performance bug reports,” Journal of Intelli gent Systems, vol. 33, no. 1, Art. no. 20240045, Dec. 2024.

[17] Ö. Köksal and C. E. Öztürk, “A survey on machine learning-based automated software bug report classification,” in Proc. 2022 Int. Symp. Multidisciplinary Studies and Innova tive Technologies (ISMSIT), Oct. 2022, pp. 635–640.

[18] X. Du, Z. Zhou, B. Yin, and G. Xiao, “Cross-project bug type prediction based on tran sfer learning,” Software Quality Journal, vol. 28, no. 1, pp. 39–57, Mar. 2020.

[19] R. Arora and A. Kaur, “Automated categorization of Bug reports as security related or non-security related: a machine learning based solution,” Vietnam Journal of Computer Science, Jan. 2025.

[20] T. Li, Z. Wang, and P. Shi, “Within-project and cross-project defect prediction based on model averaging,” Scientific Reports, vol. 15, no. 1, Art. no. 6390, Feb. 2025.

[21] E. H. Yilmaz, I. H. Toroslu, and Ö. Köksal, "A Comparative Study of Contemporary L earning Paradigms in Bug Report Priority Detection," IEEE Access, 2024.