**Bug Report Classification with Random Forests**

**1. Introduction**

Bug report classification is a key component of intelligent software engineering. In popular projects, a vast number of bug reports are submitted daily, and accurate classification can significantly improve issue triaging efficiency, optimize resource allocation, and enhance project maintenance. For this project, I selected the bug report classification task from Lab 1, where the baseline model employs a Naive Bayes classifier with TF-IDF features. To improve performance, I implemented a Random Forest model which demonstrated superior results compared to the baseline model. This task, incorporating both natural language processing and machine learning techniques, has deepened my understanding of intelligent software engineering.

**2. Related Work**

Traditional text representation methods include the Bag-of-Words model, TF-IDF, word vector representations, and so on (Rizvi, Imran, and Mahmood 2025). Traditional text classification methods mainly fall into two categories: rule-based approaches, which classify texts using predefined keywords, regular expressions, or templates (Pang 2024); and machine learning-based approaches, including Naive Bayes, Support Vector Machines, decision trees, and others (Rizvi, Imran, and Mahmood 2025).

**2.1. TF-IDF**

TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used statistical metric for evaluating the importance of a word within a document relative to an entire corpus. It assigns higher weights to words that appear frequently in a specific document but rarely across the corpus, while reducing the influence of common words that appear in most documents. This approach effectively captures discriminative textual features and is commonly applied in text mining, information retrieval, and natural language processing tasks.

TF-IDF is the product of two components：

,

: The number of times the word t appears in document d;

: The total number of occurrences of all words in the document;

: Total number of documents;

: Number of files containing the word t;

**2.2. Naive Bayes**

Naive Bayes is a classifier based on Bayes' theorem, which "naively" assumes that each feature is independent of each other. Despite the feature independence assumption rarely holding true, naive Bayes still performs well in practice (Rish 2001). Naive Bayes is fast to train and works well on small datasets, but it performs poorly when features are highly correlated (contextual semantics) and fails to distinguish similar words (lacks semantic understanding).

**2.3. Radom Forest**

Random Forest is an ensemble learning method that constructs multiple decision trees, each trained on a different subset of the data. These subsets are generated by randomly sampling from the original training set using the bagging technique. Each tree independently learns patterns from its corresponding subset, and the final prediction is made through majority voting across all trees. This ensemble approach enhances model performance by reducing variance and improving generalization. Random Forest is recognized for its robustness and resistance to overfitting. Furthermore, prior research has demonstrated that ensemble learning methods are effective in tasks such as bug classification and defect prediction (Lamkanfi et al. 2010).

**3.Solution**

The baseline model combines TF-IDF with a Naive Bayes classifier. In my proposed approach, I retain the TF-IDF feature extraction method but replace the Naive Bayes classifier with a Random Forest classifier.

The baseline model employs a text preprocessing strategy that includes the removal of HTML tags, emojis, and stopwords, followed by standard normalization. I consider this approach effective in eliminating irrelevant content while preserving key textual information. Using the “caffe” dataset, I conducted a comparison between the baseline model with and without text preprocessing. The results indicate that, after preprocessing, the model's accuracy improved by 9 percentage points and the F1-score increased by 15 percentage points. Therefore, I consider the baseline model’s text preprocessing method to be effective and sufficient.

Given that the dataset used in this experiment is relatively small, TF-IDF remains an effective representation for traditional text classification tasks, as it preserves keyword weight information well in small to medium-sized corpora. At the same time, nonlinear models like Random Forest can still perform well when handling high-dimensional sparse features such as TF-IDF.

Random Forest leverages the bagging technique and, unlike Naive Bayes which relies on the assumption of feature independence, it is capable of capturing more complex feature interactions. This makes it more robust and generally better at classification tasks.

**4. Setup**

4.1. **Development Environment**: Python 3.12, with the following libraries: pandas, numpy, scikit-learn, and nltk.

4.2. **Dataset**: The experiment uses the bug report classification dataset provided in Lab 1, which includes 286 reports from caffe, 516 from incubator-mxnet, 668 from keras, 752 from pytorch, and 1,490 from tensorflow.

4.3. **Baseline Model**: Naive Bayes paired with TF-IDF.

4.4. **Detailed Design**:

i. Text preprocessing: The training data undergoes text cleaning, including the removal of meaningless words and conversion to lowercase. Subsequently, the textual data is split into 70% for training and 30% for testing. Then TF-IDF vectorization is performed to convert the textual data into numerical feature representations.

ii. Model training: A Random Forest classifier is employed for classification. Grid search is applied to optimize hyperparameters for better performance. The training process is repeated 30 times to compute the average performance metrics.

iii. Evaluation metrics: The model is evaluated using Accuracy and F1-score.

**5. Experiments**

In this experiment, evaluations were conducted separately on five datasets. Each dataset was randomly shuffled and split into 70% for training and 30% for testing. Both the Naive Bayes and Random Forest classifiers were optimized using grid search to obtain the best hyperparameters. Each experiment was repeated 30 times, and the average results were reported to reduce random variance. The following are the experimental results on the five datasets.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 1. Caffe | | | | |
| Model Type | Accuracy | Precision | Recall | F1-score |
| Naive Bayes | 52.8% | 57.5% | 66.8% | 0.468 |
| Random Forest | 88.6% | 56.1% | 51.9% | 0.503 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 2. incubator-mxnet | | | | |
| Model Type | Accuracy | Precision | Recall | F1-score |
| Naive Bayes | 58.7% | 60.5% | 73.1% | 0.530 |
| Random Forest | 87.1% | 58.0% | 51.9% | 0.502 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 3. Keras | | | | |
| Model Type | Accuracy | Precision | Recall | F1-score |
| Naive Bayes | 56.7% | 63.9% | 70.3% | 0.551 |
| Random Forest | 83.8% | 82.6% | 62.7% | 0.654 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 4. Pytorch | | | | |
| Model Type | Accuracy | Precision | Recall | F1-score |
| Naive Bayes | 64.0% | 61.5% | 75.4% | 0.568 |
| Random Forest | 87.7% | 76.9% | 54.0% | 0.540 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 5. Tensorflow | | | | |
| Model Type | Accuracy | Precision | Recall | F1-score |
| Naive Bayes | 55.9% | 63.6% | 71.1% | 0.541 |
| Random Forest | 85.4% | 81.8% | 64.8% | 0.682 |

Based on the bug report classification results from the five deep learning projects, the Random Forest classifier consistently achieved accuracy between 83% and 89%, which is significantly higher than the Naive Bayes model’s accuracy, which ranged from 52% to 64%. A paired t-test was performed on the accuracy results of the two models across the five projects, yielding a t-value of 14.587 and a p-value of 0.00013, which is significantly less than 0.05. This indicates that the improvement in accuracy achieved by the Random Forest classifier over the Naive Bayes model is statistically significant.

In terms of F1-score, Random Forest showed better performance on the Caffe, Keras, and TensorFlow datasets. However, for Pytorch and MXNet, although the accuracy was higher, the improvement in F1-score was not as significant. One possible explanation is the presence of class imbalance in the dataset, which may limit the extent of improvement observed in precision and recall metrics.

In addition, I conducted an extra experiment by training and evaluating both models on a combined dataset that merges all reports from Caffe, MXNet, Keras, PyTorch, and TensorFlow. The experimental results are shown in Table 6.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 6. Combined dataset | | | | |
| Model Type | Accuracy | Precision | Recall | F1-score |
| Naive Bayes | 68.0% | 64.9% | 77.0% | 0.624 |
| Random Forest | 88.6% | 84.0% | 70.0% | 0.739 |

As shown in Table 6, when the five datasets are merged into a single dataset for training, the Random Forest model demonstrates superior performance, achieving an accuracy of 88.6% and an F1-score of 73.9%, which are significantly higher than those of the Naive Bayes model (accuracy: 68.0%, F1-score: 62.4%). These results suggest that with a larger sample size, the Random Forest model not only performs better, but also exhibits stronger generalization ability and greater robustness.

**6. Reflection**

Experimental results on the five individual datasets show that the Random Forest classifier achieves a significant improvement in accuracy and exceeds 85% accuracy on multiple datasets, the statistical results of the F1-score show that Random Forest only performs better on some of the datasets. In contrast, when trained on the combined dataset—which is larger and more complex—Random Forest outperforms Naive Bayes in both accuracy and F1-score, demonstrating its superior performance under more challenging conditions.

One possible reason for this is the class imbalance in the training data—across all datasets, the proportion of performance-related reports ranges only from 11% to 21%. In addition, due to the relatively small number of textual samples in each individual dataset and the inability of TF-IDF to capture contextual semantics, the model may fail to extract sufficient feature information.

The Random Forest classifier also has some limitations. First, it requires a longer training time, which is significantly higher compared to the Naive Bayes model. Second, its interpretability is relatively poor, making it more difficult to analyze the contribution of individual features compared to the more transparent Naive Bayes classifier.

Future improvements could involve using more advanced text representation methods such as Word2Vec or BERT for feature extraction. In cases where data is limited, pre-trained models can be leveraged to enhance feature quality. Additionally, techniques for addressing class imbalance, such as SMOTE, could be explored to increase the number of minority class samples. Finally, further investigation into deep learning models may help improve the overall performance of text classification.

**7. Conclusion**

This experiment focuses on the bug classification task by improving the baseline model—replacing the Naive Bayes classifier with a Random Forest classifier. Comparative experiments on five mainstream open-source projects (Caffe, MXNet, Keras, PyTorch, and TensorFlow) show that Random Forest significantly outperforms Naive Bayes in terms of accuracy, with an average improvement of nearly 29%. This improvement was further validated through a paired t-test, confirming its statistical significance (p = 0.00013). In terms of F1-score, the Random Forest model only showed better performance on some of the individual datasets. However, after merging all datasets, the Random Forest classifier significantly outperformed the Naive Bayes model, which indicates that the Random Forest model has better generalization capability.

Overall, the Random Forest classifier, as an ensemble learning method, has proven effective in improving the performance of bug report classification. In the future, performance can be further enhanced by employing more sophisticated text feature extraction techniques and applying methods to address class imbalance.

**8. Artifact**

**Reference**

[1] Rizvi, Syed Mustafa Haider, Ramsha Imran, and Arif Mahmood. 2025. Text Classification Using Graph Convolutional Networks: A Comprehensive Survey. ACM Computing Surveys 57 (8): Article 201. <https://doi.org/10.1145/3714456>.

[2] Pang, Ronghui. 2024. Research on Text Classification Applications Based on NLP Technology. In Proceedings of the 2024 5th International Conference on Computing, Networks and Internet of Things (CNIOT 2024), May 24–26, Tokyo, Japan. New York, NY: ACM. <https://doi.org/10.1145/3670105.3670124>.

[3] Rish, I. 2001. An Empirical Study of the Naive Bayes Classifier. T.J. Watson Research Center, IBM.

[4] Lamkanfi, Ahmed, Serge Demeyer, Emanuel Giger, and Bart Goethals. 2010. *Predicting the Severity of a Reported Bug*. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 1–10. IEEE.