

Research on Software Defect Prediction Based on Static and Dynamic Feature Fusion and Machine Learning

Tianyu Yin *

School of Computer Science and Artificial Intelligence, Beijing Technology and Business University, Beijing, 102488, China

* Corresponding author Email: c197971@163.com

Abstract: Aiming at the problems that the static-dynamic feature fusion mechanism lacks systematic multi-scenario verification, the feature-model adaptation law is unclear, and the engineering practicability of existing research conclusions is insufficient, this paper proposes a static-dynamic feature fusion defect prediction method based on multi-model comparison. Taking Equinox and Eclipse open-source Java project datasets from Kaggle platform as research objects, this paper constructs a feature system including 18-dimensional static CK-OO metrics and 20-dimensional dynamic change features, compares the prediction performance of Logistic Regression, Random Forest, XGBoost and Stacking ensemble model under static, dynamic and fusion feature sets. 5-fold cross-validation is used to ensure experimental reliability, and F1-Score is taken as the core evaluation index to carry out controlled experiments. The results show that on the balanced Equinox dataset, Random Forest with fusion features achieves the best single-model performance with an F1-Score of 0.7619 and Stacking model achieves 11.83% performance improvement in the static feature scenario; on the imbalanced Eclipse dataset, XGBoost with static features achieves the best performance with an F1-Score of 0.6512; CVS version control entropy, weighted method complexity, and class response are cross-scenario general core prediction features. This study reveals the core influence mechanism of dataset category balance and project scale on feature-model adaptability, and provides an empirical reference for defect prediction practice of different types of software projects.

Keywords: Software Defect Prediction; Machine Learning; Feature Fusion; Static Code Metrics; Dynamic Change Features; Version Control Entropy.

1. Introduction

With the rapid development of the digital economy, software systems have been deeply integrated into industrial production, social services and daily life, while their growing scale and complexity have made software defects a major trigger for system failures, data breaches and economic losses. Industry data shows that the cost of fixing software defects in the operation and maintenance phase is 300 times higher than that in the requirements design stage, and post-launch security incidents caused by defects will bring irreversible damage to enterprises [1]. In this context, software defect prediction technology has become a research hotspot at the intersection of software engineering and artificial intelligence, as it can automatically identify high-risk code modules through machine learning, optimize test resource allocation, and guide code optimization [2].

After decades of development, software defect prediction has formed a research system with feature representation and model construction as the core. Existing studies have verified the effectiveness of static code metrics and dynamic change features in defect prediction, and confirmed the performance advantages of ensemble models through benchmark tests [3-6]. Hall et al. further pointed out that dataset category balance and feature engineering quality are core factors affecting model performance [7]. However, there are still three key limitations: first, most studies only verify the effectiveness of feature fusion through simple feature union, lacking systematic multi-scenario verification and in-depth analysis of the fusion mechanism; it should be noted that the feature fusion method in this paper is also a form of feature concatenation, which is essentially similar to the simple union approach criticized above. second, Catal et al. found that

ensemble models perform better on imbalanced datasets, but the adaptation scenarios of different ensemble strategies are still unclear; third, domestic scholars have enriched the methodological system of defect prediction, but existing work mostly validates feature effects based on a single dataset or model, lacking systematic cross-validation from a multi-project and multi-model perspective, and the influence mechanism of scenario factors such as project scale is insufficiently analyzed, resulting in weak engineering practicability of research conclusions [8,9].

To address the above gaps, this paper conducts preliminary dual-dataset comparison and analysis on two typical open-source project datasets (the balanced medium-sized Equinox dataset and the imbalanced large-scale Eclipse dataset). We construct a multi-dimensional feature system covering static code metrics and dynamic change features, compare the prediction performance of three classic single models and the Stacking ensemble model under different feature sets, and reveal the feature-model adaptation laws in different scenarios, to provide a practical reference for engineering defect prediction practice.

2. Relevant Theories and Technical Foundations

2.1. Core Feature System for Software Defect Prediction

This paper constructs a three-category feature system covering two dimensions of code structure design and development process evolution. Static features are extracted based on single-version code structure information using the classic CK-OO object-oriented metric standard, which can characterize the potential defect risk of code from the static

structure level [3,4]. Dynamic features are constructed based on code version control history, including basic change metrics and CVS version control entropy features, which can capture the development process defect risk that static features cannot cover [5]. Fusion features are the deduplicated union of static and dynamic features, which can realize information complementarity of the two types of features and provide more comprehensive defect signals for machine learning models [10].

2.2. Prediction Model and Evaluation System

This paper selects four mainstream models to cover different algorithm types: Logistic Regression (as the baseline model), Random Forest (Bagging ensemble), XGBoost (Boosting ensemble), and Stacking ensemble model (two-layer architecture with 3 base learners and 1 meta-learner), to fully reflect the adaptability of different models in defect prediction scenarios [11].

We build a two-dimensional evaluation system: for classification performance, we take F1-Score as the core index, supplemented by Precision, Recall and AUC; for model stability and generalization ability, we use 5-fold cross-validation for evaluation, and detect overfitting through Gap value (defined as $\text{Gap} = \text{F1}_{\text{test}} - \text{F1}_{\text{cv}}$).

3. Experimental Design and Dataset Construction

3.1. Experimental Dataset

The experimental data are from the publicly available Equinox and Eclipse defect prediction benchmark datasets on Kaggle, both of which are classic verification resources in this field with reliable data quality and annotation [5]. The two datasets correspond to two typical industrial scenarios, with consistent feature dimensions to ensure the comparability of cross-dataset results.

The Equinox dataset is the main experimental object, which is a medium-sized project with 324 class-level samples, and the ratio of defective to non-defective samples is close to 2:3, without serious class imbalance. The Eclipse dataset is the auxiliary verification object, which is a large-scale project with 997 class-level samples, and the ratio of defective to non-defective samples is about 1:4, with obvious class imbalance, which is more consistent with the data distribution characteristics of most large-scale industrial software projects [12].

Figure 1 shows the basic information comparison between the two datasets.

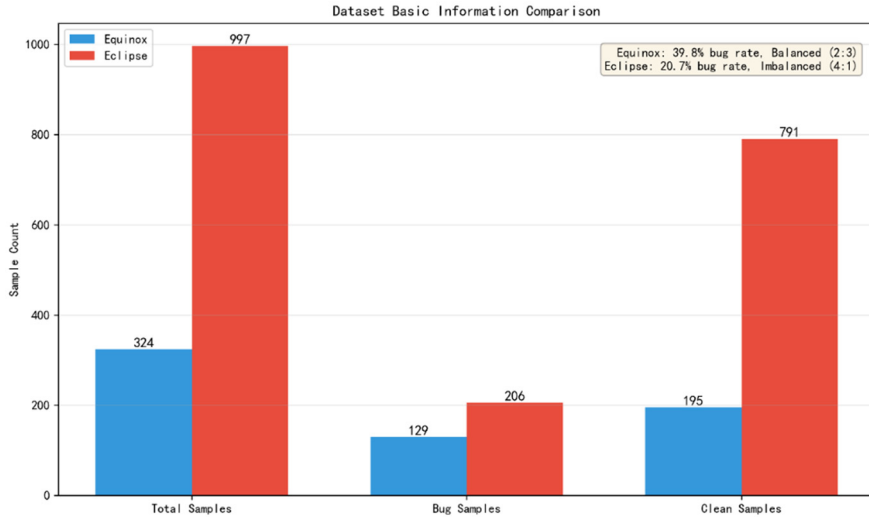


Figure 1. Comparison of basic information between the Equinox and Eclipse datasets

3.2. Data Preprocessing Process

A unified standardized preprocessing process is implemented for both datasets to eliminate the impact of data quality issues on experimental results, including five core steps: basic data cleaning, median filling for missing values, feature standardization with StandardScaler, stratified sampling for train-test split consistency and class imbalance mitigation through balanced category weights, and 8:2 stratified splitting of training and test sets to avoid data leakage [8,11].

3.3. Design of the Experimental Scheme

This paper adopts a strict controlled experiment design: feature type and model type are taken as independent variables; F1-Score, AUC, Precision, Recall and cross-validation F1 are taken as dependent variables; dataset division method, cross-validation times, evaluation metrics and random seed (fixed to 42) are set as control variables to

ensure the reproducibility of results.

The experimental process follows standardized machine learning norms: first, complete dataset preprocessing and feature set construction; then, complete model construction and hyperparameter optimization through grid search and 5-fold cross-validation; next, train models on three feature sets and evaluate performance on the independent test set; finally, extract feature importance data and summarize adaptation rules. The core objectives are to verify the performance gain of static-dynamic feature fusion, summarize optimal model-feature combinations, mine cross-scenario core prediction features, and clarify the applicable boundaries of the Stacking model [6].

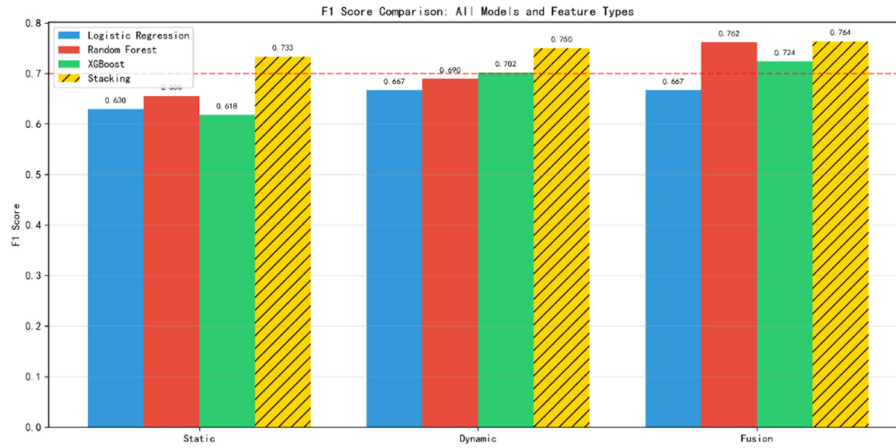
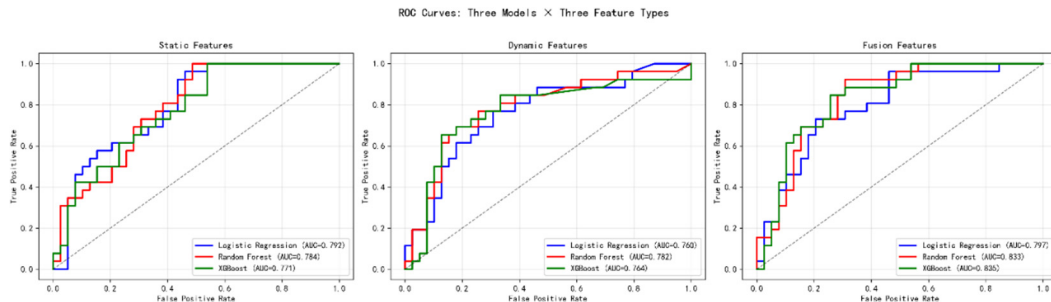
4. Experimental Results and Analysis

4.1. Experimental Results and Analysis of the Equinox Dataset

Table 1. Complete Experimental Results (4 Models $\times$ 3 Feature Sets $\times$ 2 Datasets)

Dataset	Model	Feature Set	F1	Precision	Recall	AUC	CV-F1	Gap
Eclipse	Logistic Regression	Static	0.5909	0.5532	0.6341	0.8392	0.5999	-0.009
Eclipse	Random Forest	Static	0.5854	0.5854	0.5854	0.8596	0.5856	-0.0003
Eclipse	XGBoost	Static	0.6279	0.6	0.6585	0.8567	0.6176	0.0103
Eclipse	Stacking	Static	0.6279	0.6	0.6585	0.8658	0.6222	0.0057
Eclipse	Logistic Regression	Dynamic	0.5652	0.5098	0.6341	0.8037	0.6006	-0.0354
Eclipse	Random Forest	Dynamic	0.575	0.5897	0.561	0.8162	0.5853	-0.0103
Eclipse	XGBoost	Dynamic	0.5747	0.5435	0.6098	0.8092	0.606	-0.0313
Eclipse	Stacking	Dynamic	0.5476	0.5349	0.561	0.8296	0.632	-0.0843
Eclipse	Logistic Regression	Fusion	0.5806	0.5192	0.6585	0.8319	0.5553	0.0253
Eclipse	Random Forest	Fusion	0.5974	0.6389	0.561	0.8512	0.6208	-0.0234
Eclipse	XGBoost	Fusion	0.6207	0.587	0.6585	0.8438	0.6317	-0.011
Eclipse	Stacking	Fusion	0.6047	0.5778	0.6341	0.855	0.6172	-0.0126
Equinox	Logistic Regression	Static	0.6296	0.6071	0.6538	0.7919	0.6585	-0.0288
Equinox	Random Forest	Static	0.6557	0.5714	0.7692	0.784	0.7123	-0.0565
Equinox	XGBoost	Static	0.6182	0.5862	0.6538	0.7771	0.6739	-0.0558
Equinox	Stacking	Static	0.7333	0.6471	0.8462	0.8254	0.6402	0.0931
Equinox	Logistic Regression	Dynamic	0.6667	0.6129	0.7308	0.7599	0.7206	-0.0539
Equinox	Random Forest	Dynamic	0.6897	0.625	0.7692	0.7816	0.766	-0.0763
Equinox	XGBoost	Dynamic	0.7018	0.6452	0.7692	0.7653	0.7359	-0.0341
Equinox	Stacking	Dynamic	0.75	0.7	0.8077	0.8757	0.7183	0.0317
Equinox	Logistic Regression	Fusion	0.6667	0.6129	0.7308	0.7968	0.7113	-0.0447
Equinox	Random Forest	Fusion	0.7619	0.6486	0.9231	0.8304	0.7624	-0.0005
Equinox	XGBoost	Fusion	0.7241	0.6562	0.8077	0.8245	0.7324	-0.0082
Equinox	Stacking	Fusion	0.7636	0.7241	0.8077	0.8777	0.7609	0.0028

Note: Gap = F1(Test) - F1(CV). Positive Gap indicates potential overfitting. Best single-model performance per dataset-feature combination is highlighted.

**Figure 2.** Comparison of F1-scores of all models and feature types in the Equinox dataset**Figure 3.** ROC curves of the models under the three types of features in the Equinox dataset

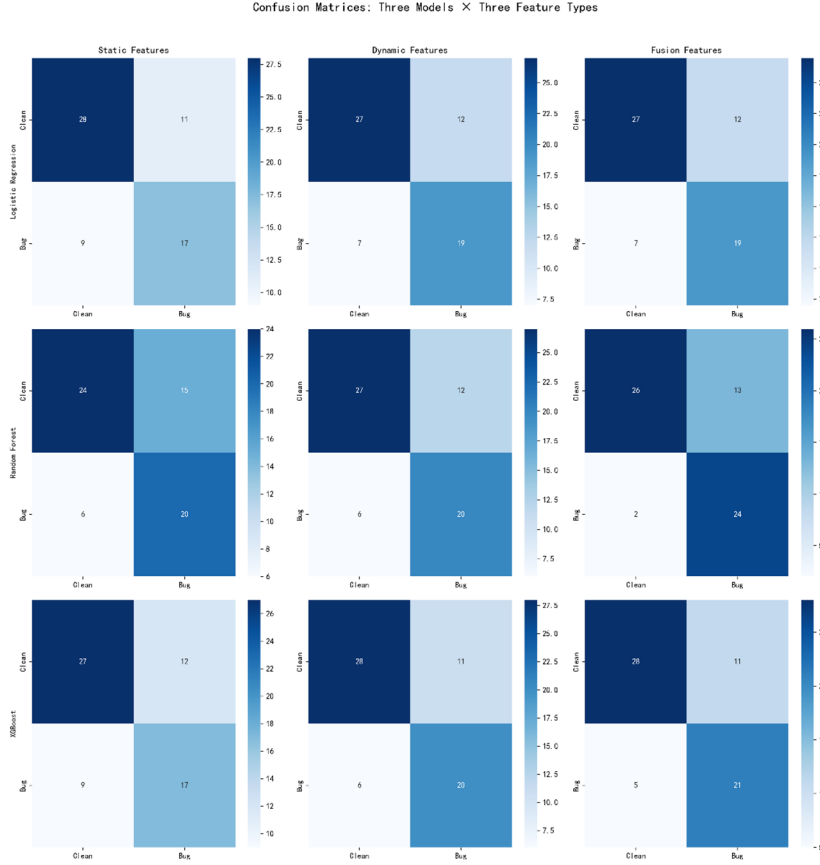


Figure 4. Confusion matrix of three types of models and three types of features in the Equinox dataset

For the balanced Equinox dataset, Random Forest combined with fusion features achieves the best single-model performance (F1-Score=0.7619, recall=92.31%), XGBoost performs best in the dynamic feature scenario, and Logistic Regression is stable in all scenarios with no serious overfitting.

Figure 2 shows the F1-Score comparison of all models and feature types; Figure 3 shows the ROC curves of models

under three feature types; Figure 4 shows the confusion matrices of three models and three feature types.

The Stacking ensemble model achieves 11.83% F1-Score improvement in the static feature scenario and 6.88% improvement in the dynamic feature scenario, but only 0.22% gain in the fusion feature scenario, with no overfitting. Figure 5 shows the overfitting analysis results of all models and feature types.

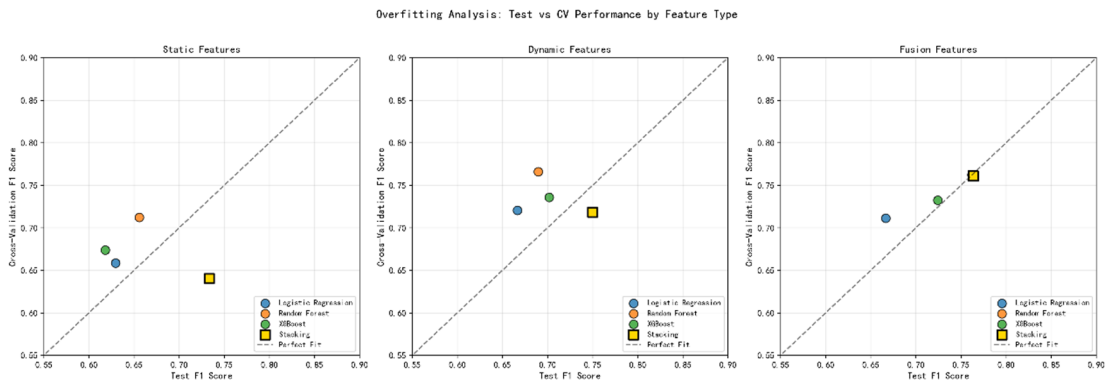


Figure 5. Overfitting analysis of all models and feature types in the Equinox dataset

Feature importance analysis shows that weighted method complexity and lines of code are core static features, while CVS entropy-related features dominate dynamic features. Figures 6-8 show the top 10 important static, dynamic, and fusion features, respectively, verifying the information complementarity of static and dynamic features [13].

4.2. Experimental Results and Analysis of the Eclipse Dataset

For the imbalanced Eclipse dataset, XGBoost combined

with static features achieves the best overall performance (F1-Score=0.6512), and still maintains the best performance in dynamic and fusion feature scenarios. Random Forest and Logistic Regression underperform XGBoost in all scenarios, indicating that dynamic feature noise is amplified by class imbalance in large-scale projects.

Figure 9 shows the F1-Score comparison of all models and feature types; Figure 10 shows the ROC curves of models under three feature types; Figure 11 shows the confusion matrices of three models and three feature types.

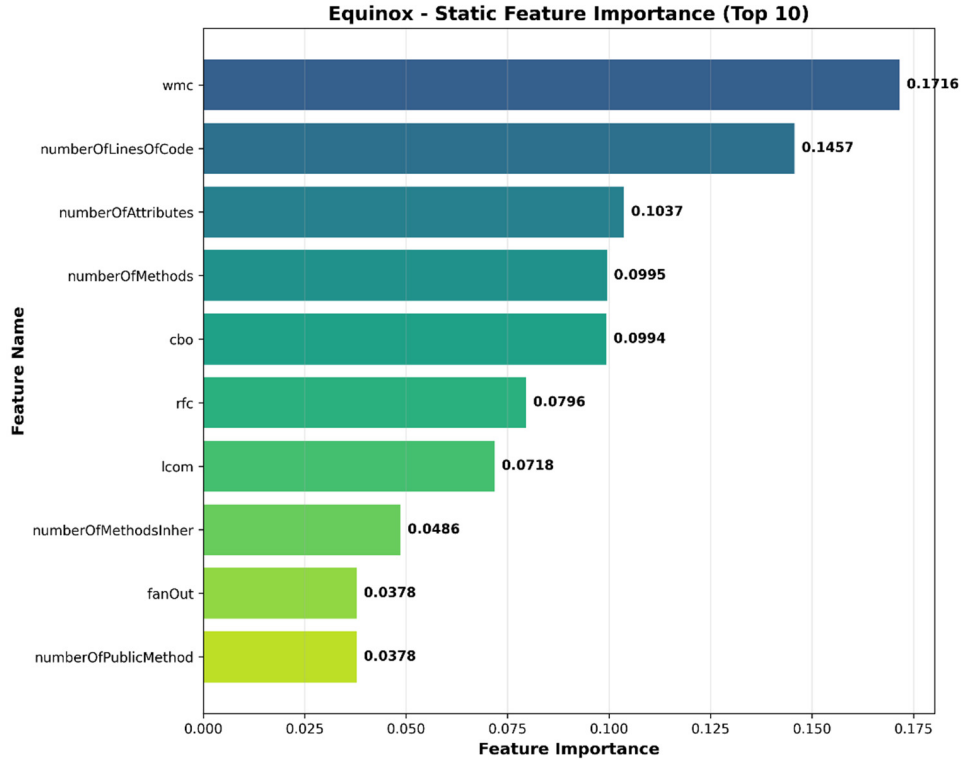


Figure 6. Top10 Most Important Static Features of the Equinox dataset

The Stacking model underperforms the optimal single model in static and dynamic feature scenarios, with slight overfitting in the dynamic feature scenario, and is basically the same as the optimal single model only in the fusion feature scenario.

Figure 12-14 show the top 10 important static, dynamic and fusion features of the Eclipse dataset respectively; Figure 15 shows the overfitting analysis results of all models and feature types. The core defect-driven features are consistent with the Equinox dataset, verifying the universality of core indicators.

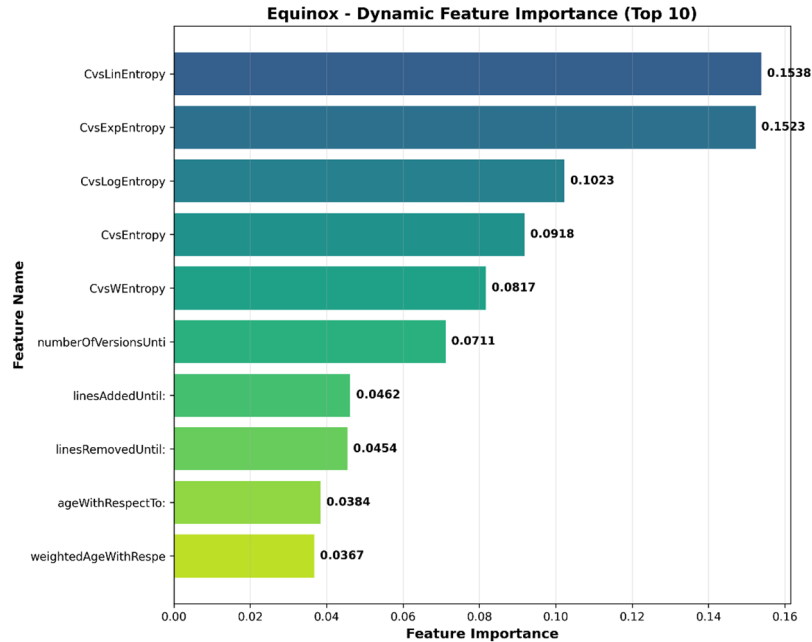


Figure 7. Top10 Dynamic Features of Importance in the Equinox dataset

4.3. Summary of Feature-Model Fit Rules

Based on the dual-dataset experimental results, this paper summarizes clear feature-model preliminary patterns. In terms of feature type adaptability, balanced datasets are more suitable for fusion features to leverage the information

complementarity of static and dynamic features, while imbalanced datasets are more suitable for low-noise static features, as dynamic feature noise is easily amplified by category imbalance. In terms of model adaptability, Random Forest performs best with static and fusion features on balanced datasets, XGBoost has the strongest adaptability to

imbalanced data and dynamic feature scenarios, and Logistic Regression can serve as a stable baseline model for all scenarios. In terms of the applicable boundaries of the Stacking ensemble model, it can significantly improve

performance in single-feature scenarios with insufficient information, but has limited gains in fusion feature scenarios, and is prone to overfitting on severely imbalanced datasets, requiring cautious use in engineering practice.

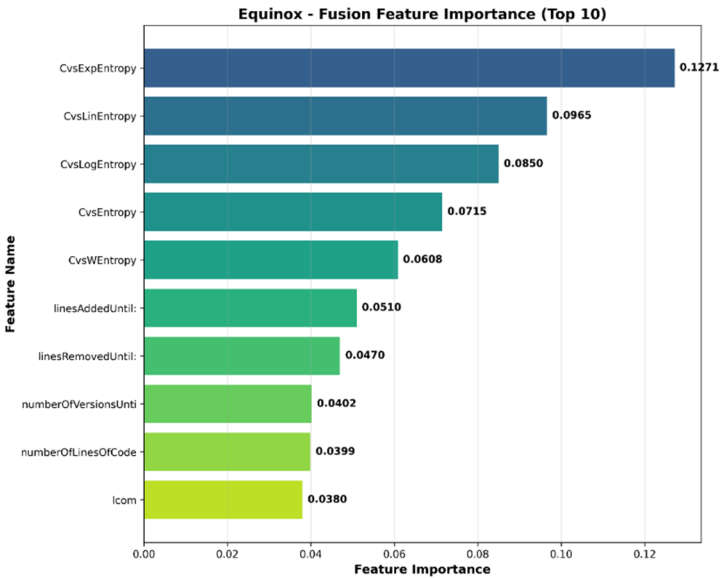


Figure 8. Top10 Most Important Fusion Features in the Equinox dataset

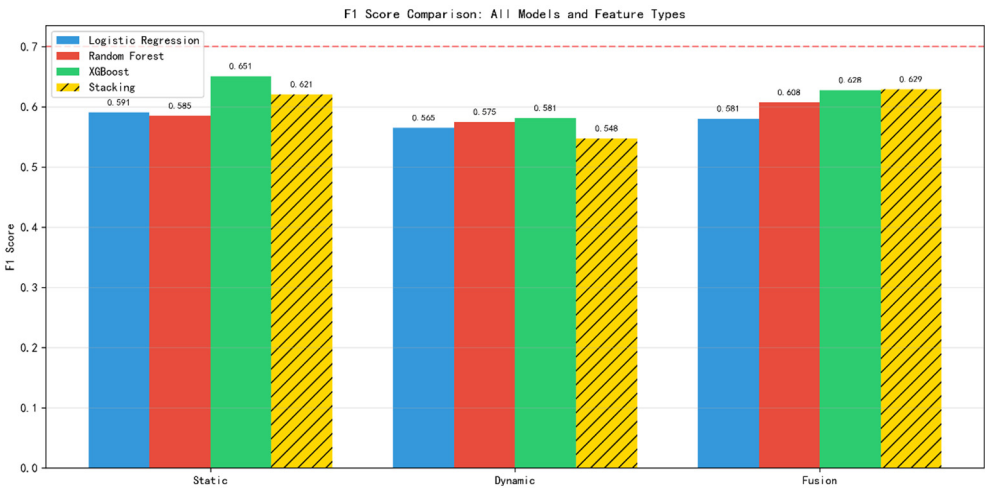


Figure 9. Comparison of F1-scores of all models and feature types in the Eclipse dataset

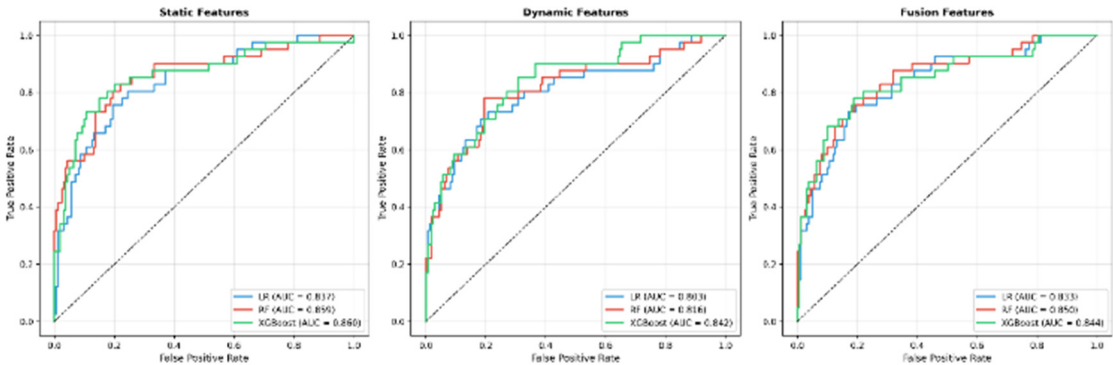


Figure 10. ROC curves of each model under three types of features in the Eclipse dataset

5. Causes of Dataset Differences and Engineering Applications

Figure 16 shows the four-quadrant comparison of core differences between the two datasets. The optimal F1-Score gap exceeds 11 percentage points, but both datasets share

common core defect drivers: code complexity and code evolution chaos are cross-scenario universal indicators. The core performance difference stems from two factors: first, category balance difference. The balanced distribution of Equinox allows the model to learn feature-defect associations without bias, fully exploiting the complementarity of static and dynamic features; while the severe imbalance of Eclipse

leads to majority class bias, making low-noise static features more reliable. Second, project scale and defect pattern difference. Medium-sized Equinox mainly has structural defects, where static and dynamic features can complement

each other effectively; large-scale Eclipse mainly has evolutionary defects, and normal functional iterations further amplify dynamic feature noise, reducing the effectiveness of fusion features.

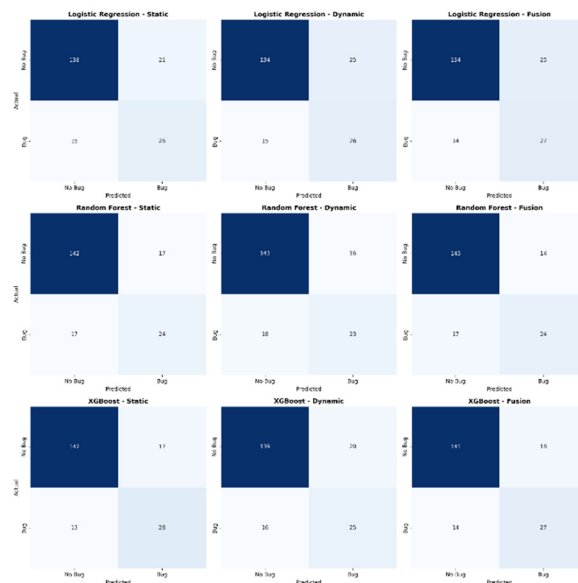


Figure 11. Confusion matrix of three types of models and three types of features in the Eclipse dataset

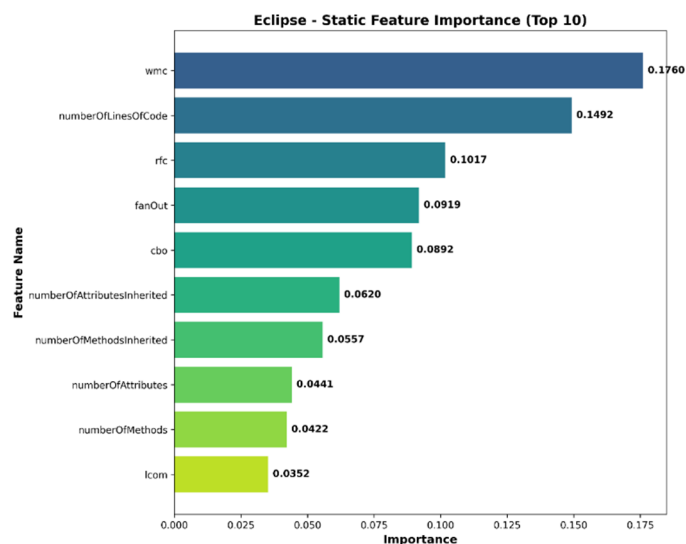


Figure 12. Top10 Static Features of Importance in the Eclipse dataset

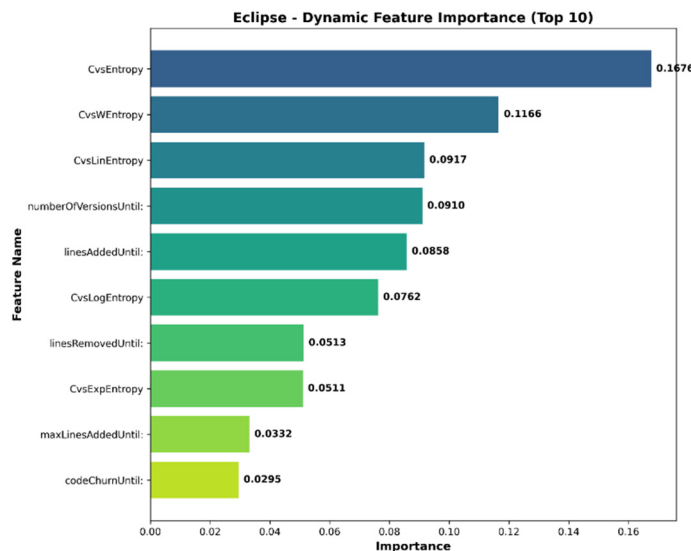


Figure 13. Top10 Dynamic Features of Importance in the Eclipse Dataset

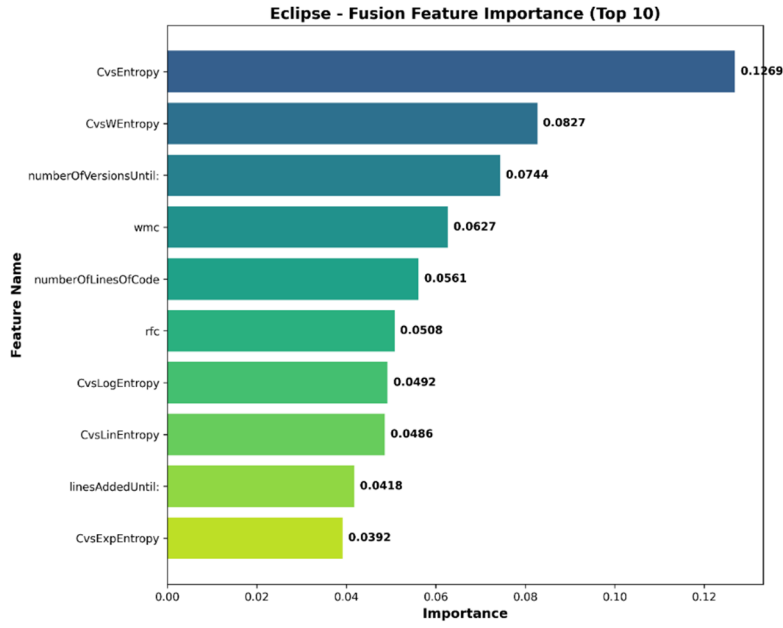


Figure 14. Top10 Importance of Fusion Features in the Eclipse dataset

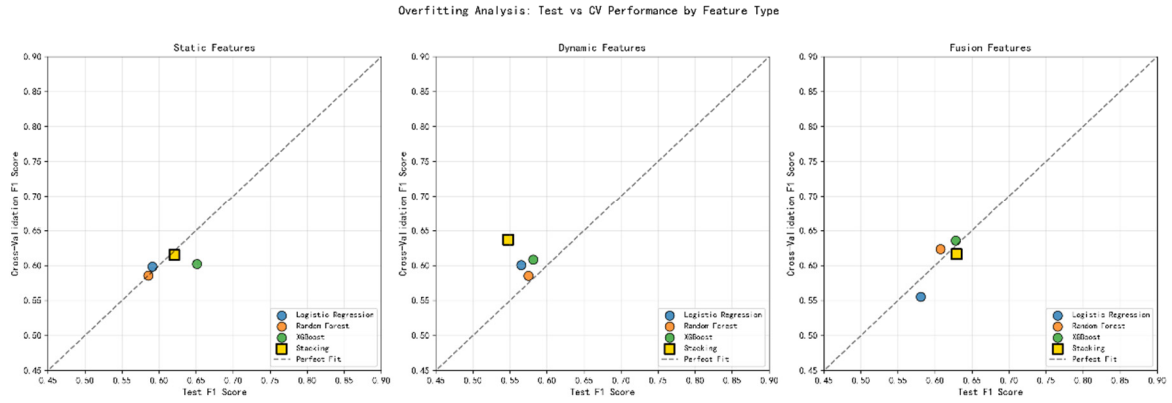


Figure 15. Overfitting analysis of all models and feature types in the Eclipse dataset

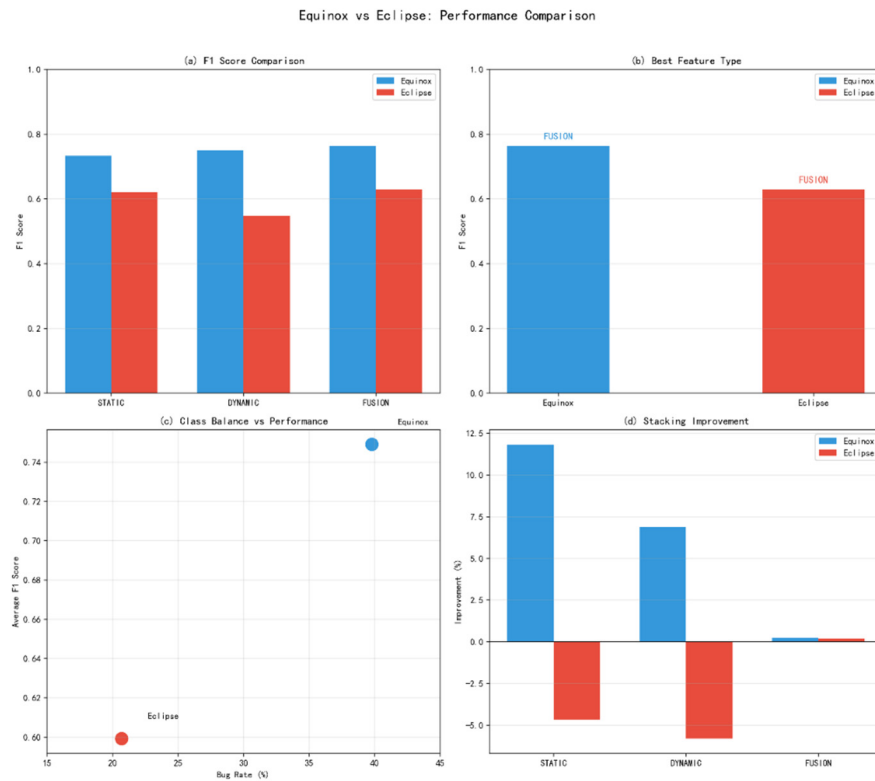


Figure 16. Comparison of the four quadrants of core differences between Equinox and Eclipse datasets

Based on the experimental conclusions, this paper proposes targeted engineering application suggestions: for category-balanced medium-sized projects, prioritize the Random Forest + fusion feature scheme, and introduce Stacking integration when feature information is insufficient; for category-imbalanced large-scale projects, prioritize the XGBoost + static feature scheme, and mitigate imbalance through class weight adjustment and stratified sampling. All projects should focus on monitoring weighted method complexity, lines of code and CVS version control entropy, establish a full-life-cycle defect prevention and control system from requirement design to operation and maintenance, optimize code structure and change processes, and reduce defect risks from the source.

6. Conclusion and Prospects

This paper conducts systematic controlled experiments with dual datasets, multiple models and multi-dimensional features, verifies the performance of static-dynamic feature fusion and different machine learning models in software defect prediction tasks, and reveals the feature-model adaptation laws in different scenarios. The main contributions and implications of this study are threefold. In terms of theoretical contribution, this study reveals the core influence mechanism of dataset category balance and project scale on feature-model adaptability, providing empirical evidence for the context-dependent nature of defect prediction. In terms of practical contribution, the proposed feature-model matching guidelines offer actionable references for engineering teams to select appropriate prediction schemes based on project characteristics. In terms of methodological contribution, the dual-dataset controlled comparison framework demonstrates a replicable approach for cross-project validation in defect prediction research.

This study still has limitations: the experiments only verify two Java open-source project datasets, and the model selection only covers traditional machine learning and basic Stacking framework. In the future, we will expand the experimental dataset, optimize the model architecture and feature fusion strategy, and develop a lightweight defect prediction tool to promote the engineering application of research results.

Future work will focus on the following directions: expanding dataset diversity by incorporating additional open-source projects and industrial datasets to enhance generalizability; exploring advanced feature fusion strategies such as weighted fusion mechanisms, feature selection-based fusion, and attention-based feature interaction methods to overcome the limitations of simple concatenation; investigating deep learning architectures for capturing complex feature interactions; and developing a lightweight, configurable defect prediction toolkit for practical

engineering adoption.

References

- [1] Boehm, B. W., & Basili, V. R. (2001). Software defect reduction top 10 list. *IEEE Computer*, 34(1), 135-137. <https://doi.org/10.1109/2.889936>.
- [2] Zhang, H., Harman, M., & Jia, Y. (2015). Software defect prediction: a survey. *IEEE Transactions on Software Engineering*, 42(1), 78-108. <https://doi.org/10.1109/TSE.2015.2442237>.
- [3] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6), 476-493. <https://doi.org/10.1109/32.295895>.
- [4] Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1), 2-13. <https://doi.org/10.1109/TSE.2007.256941>.
- [5] D'Ambros, M., Lanza, M., & Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance* (pp. 310-319). IEEE Press. <https://doi.org/10.1109/ICSM.2010.5609731>.
- [6] Lessmann, S., Baesens, B., & Mues, C. (2008). Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4), 485-496. <https://doi.org/10.1109/TSE.2008.37>.
- [7] Hall, T., Beecham, S., & Bowes, D. (2012). A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6), 1276-1304. <https://doi.org/10.1109/TSE.2011.103>.
- [8] Catal, C., & Diri, B. (2009). A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4), 7346-7354. <https://doi.org/10.1016/j.eswa.2008.10.027>.
- [9] Wu, F., Jing, X. Y., & Yao, Y. F. (2014). A review of software defect prediction methods. *Computer Science*, 41, 1-6. (In Chinese)
- [10] Wang, L. L. (2023). Research on software defect prediction method based on multi-feature fusion [Master's thesis]. Shanghai Normal University. (In Chinese)
- [11] Zhou, Z. H. (2016). Machine learning. Tsinghua University Press. (In Chinese)
- [12] Peters, F., Menzies, T., & Marcus, A. (2013). Better cross-company defect prediction. *IEEE Transactions on Software Engineering*, 39(11), 1138-1152. <https://doi.org/10.1109/TSE.2013.34>.
- [13] Zimmermann, T., & Nagappan, N. (2008). Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th International Conference on Software Engineering* (pp. 531-540). ACM Press. <https://doi.org/10.1145/1368088.1368161>.