ISSN: 2472-3703 DOI: 10.6911/WSRJ.202411_10(11).0005

ECU Software Defect Prediction Model Based on Machine Learning

Qiujun Zhao[1, 2, a], Lishuang Zhu[1, 2, b, *], Yu Su[1, 2, c], Ziwei Tian[1, 2, d], Shuhua Zhou[1, 2, e]

[1]China Automotive Technology & Research Center Co., Ltd, Tianjin, China

[2]CATARC Software Testing (Tianjin) Co., Ltd, China

[a]zhaoqiujun@catarc.ac.cn, [b]zhulishuang@catarc.ac.cn, [c]suyu@catarc.ac.cn, [d]tianziwei@catarc.ac.cn, [e]zhoushuhua@catarc.ac.cn

[*] Corresponding author

Abstract

With the increasing dependence of the automotive industry on Electronic Control Unit (ECU) software, software defects can lead to serious safety and performance issues. Therefore, developing effective ECU software defect prediction models is crucial for improving software quality and reducing potential risks. This article proposes a machine learning based ECU software defect prediction model, which predicts the existence of software defects by analyzing the source code, historical defect data, and static metrics of the software. Advanced data preprocessing techniques, feature selection methods, and machine learning algorithms are used to improve the accuracy and generalization ability of the model.

Keywords

Machine Learning; ECU; Software Defects; Prediction Model.

1. INTRODUCTION

In modern automotive systems, the Electronic Control Unit (ECU) plays a core role in controlling multiple critical functions, from engine management to safety systems. As the complexity of automotive software continues to increase, the probability of software defects also increases, which not only affects vehicle performance but may also endanger passenger safety. Traditional software testing methods are often time-consuming and costly, making it difficult to cover all potential defects. Therefore, researching efficient prediction of software defects has become an important topic in the field of software engineering.

1.1. Overview of ECU Software

ECU is a microcomputer inside the vehicle, responsible for controlling various functions such as engine management, braking system, airbags, etc. Due to the complexity of ECU software and high requirements for safety, the existence of software defects may lead to serious safety issues and performance degradation. In the process of ECU software development, traditional defect detection methods often rely on manual testing and empirical judgment, which are not only time-consuming and labor-intensive, but also difficult to cover all potential defects. Therefore, using machine learning technology to automatically predict software defects and improve the efficiency and accuracy of defect detection has become an urgent need. Machine learning models can learn defect patterns from historical data, and predict possible defects in software by analyzing information such as source code, change history, and test results.

ISSN: 2472-3703 DOI: 10.6911/WSRJ.202411_10(11).0005

1.2. Basic Principles of Machine Learning

As a branch of artificial intelligence, machine learning aims to enable computer systems to learn from data and automatically improve performance. In the field of ECU software defect prediction, machine learning analyzes historical data to identify patterns and features related to defects, and then predicts potential defects in future software. This process involves multiple steps such as data collection, preprocessing, feature extraction, model training, and validation. In the data collection phase, researchers need to collect various data from the development process of ECU software, including source code, compilation errors, test results, change logs, etc. These data are cleaned and preprocessed to ensure their quality and usability. Subsequently, through feature extraction techniques, useful information for defect prediction is extracted from the raw data, which may include code complexity, change frequency, historical defect records, etc. During the model training phase, appropriate machine learning algorithms are selected to learn the extracted features. Common algorithms include decision trees, random forests, support vector machines, neural networks, etc. Each algorithm has its unique advantages and applicable scenarios. Choosing the appropriate algorithm is crucial for improving prediction accuracy. During the training process, the algorithm will continuously adjust parameters to minimize prediction errors.

2. ANALYSIS OF ECU SOFTWARE DEFECT PREDICTION MODEL BASED ON MACHINE LEARNING

2.1. Model Architecture Design

When designing a machine learning based ECU software defect prediction model, the model architecture is a crucial component, where the architecture design needs to consider how to effectively extract features from the ECU software and use these features to train the machine learning model in order to accurately predict potential software defects.

When constructing a machine learning based ECU software defect prediction model, the steps of model architecture design can be described as follows:

(1) Data preprocessing: This is the first step in model architecture design, involving data cleaning, normalization, and feature engineering (see Figure 1). During this process, it is necessary to ensure data quality and prepare appropriate inputs for model training. For ECU software, this may include extracting relevant features from software code, log files, test results, and configuration parameters.

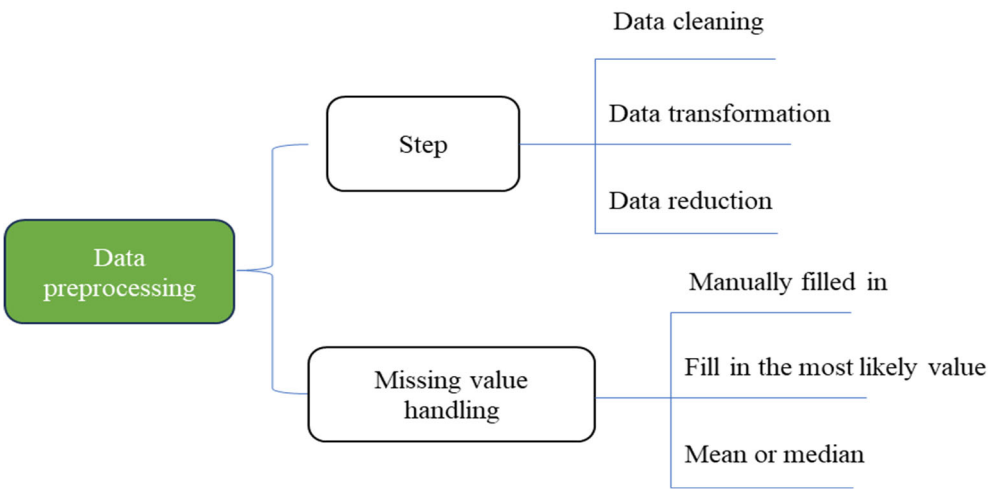

Figure 1. Data Preprocessing

ISSN: 2472-3703 DOI: 10.6911/WSRJ.202411_10(11).0005

(2) Choosing machine learning algorithms: Choosing the appropriate algorithm is crucial in architecture design. Deep learning, especially Convolutional Neural Networks (CNN), has shown effectiveness in software defect prediction. CNN excels at processing large amounts of data and identifying complex patterns, making it suitable for handling high-dimensional data (see Figure 2).

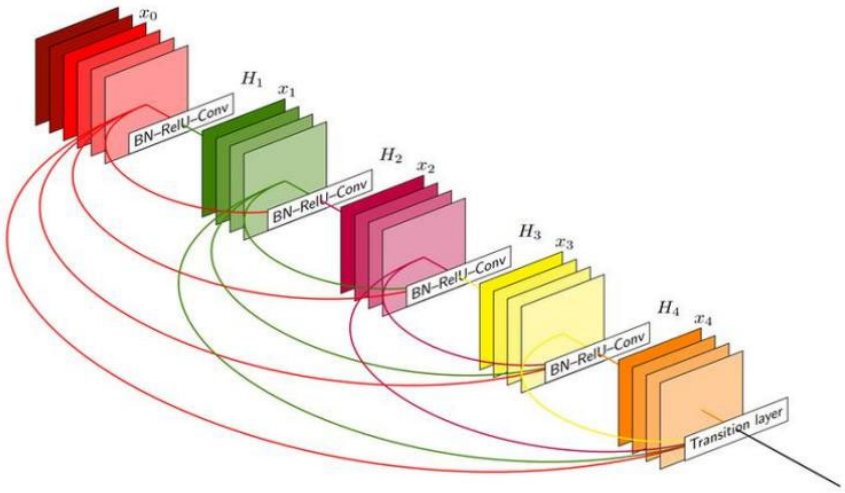

Figure 2. Convolutional Neural Network

(3) Dealing with data imbalance: A common problem in software defect prediction is data imbalance (see Figure 3). To improve the predictive performance of the model, oversampling or undersampling techniques can be used to balance the dataset. Random oversampling is a simple and effective method for dealing with data imbalance.

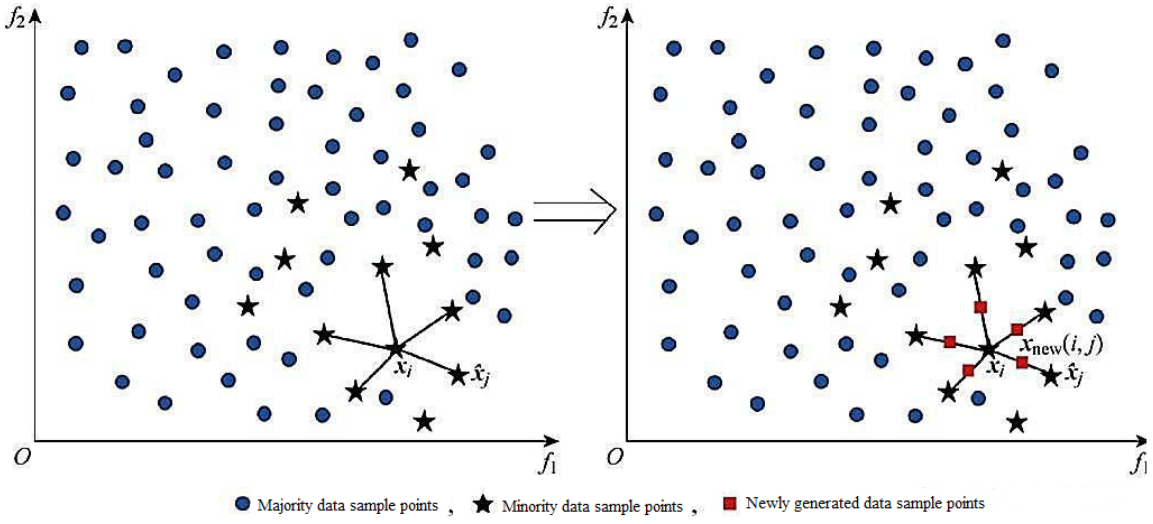

Figure 3. Dealing with Data Imbalance Issues

(4) Applying regularization techniques: To prevent overfitting of the model, regularization techniques such as dropout can be used (see Figure 4). This is a method of randomly "discarding" certain neurons during neural network training to reduce their dependencies during the training process.

ISSN: 2472-3703 DOI: 10.6911/WSRJ.202411_10(11).0005

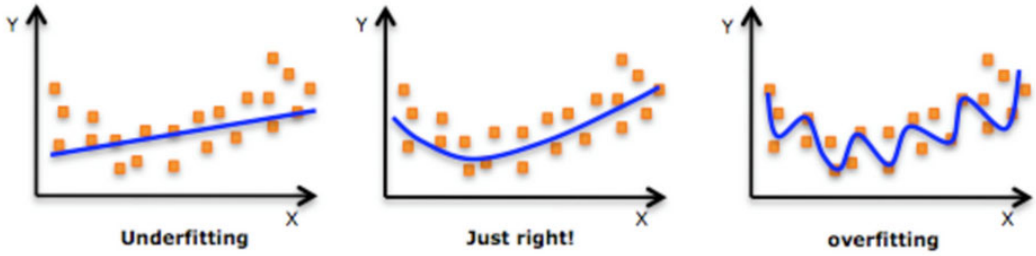

Figure 4. Application of Regularization

(5) Model evaluation: The final step in model architecture design is to implement effective evaluation strategies to ensure the accuracy and reliability of the model, which typically involves using cross validation and other statistical methods to evaluate the performance of the model. Through the above steps, a machine learning model that can accurately predict ECU software defects can be constructed, and these design considerations help ensure the effectiveness and reliability of the model in practical applications [1].

2.2. Algorithm Implementation Details

When implementing a machine learning based ECU software defect prediction model, the implementation details of the algorithm are crucial, which involves extracting features from the Abstract Syntax Tree (AST) of the source code, and then using these features to train a deep learning model. Specifically, the algorithm implementation includes the following key steps:

(1) Feature extraction: Firstly, an AST is constructed from the source code of the ECU software. AST can represent the syntax structure of the code in detail, including various declarations and statements in the program. From AST, rich code features can be extracted, such as node types, quantities, and relationships between them. These features can capture the semantic information of the code and provide strong data support for subsequent defect prediction.

(2) Data preprocessing: After extracting AST features, the data needs to be preprocessed, including addressing data imbalance issues. For example, random oversampling can be used to increase the number of minority class samples, so that the number of samples in each class is balanced during model training. In addition, the features need to be normalized to eliminate the dimensional influence between different features.

(3) Model design: Deep learning based models, such as Convolutional Neural Networks (CNNs), can be designed to process extracted AST features. CNNs can automatically learn hierarchical representations of features to capture more abstract code patterns. In model design, pre trained network structures, such as GoogLe Net, can be used to improve model performance through transfer learning methods.

(4) Prevention of overfitting: When training deep learning models, dropout strategy can be used to prevent overfitting. Dropout is a regularization technique that randomly discards a portion of neurons during the training process, making the model more robust.

(5) Model training and evaluation: Train the model using historical engineering data and evaluate its performance through methods such as cross validation. Common evaluation metrics include Area Under Curve (AUC) and F1 measure, which can comprehensively reflect the accuracy and stability of the model. During the model training and evaluation phase, we use historical engineering data to train the model and employ methods such as cross validation to evaluate its performance. Common evaluation metrics include AUC and F1 measure, and the following are the calculation formulas for these metrics:

ISSN: 2472-3703 DOI: 10.6911/WSRJ.202411_10(11).0005

① AUC: AUC is the area under the receiver operating characteristic curve (ROC curve), which represents the performance of the model at all possible classification thresholds. The higher the AUC value, the better the classification performance of the model. The calculation of AUC can be performed using the following formula:

$$ACU = \int_{t=-\infty}^{\infty} TPR\ （t）\ dFPR(t)$$

Among them, TPR (True Positive Rate) is the true rate, and FPR (False Positive Rate) is the false positive rate.

② F1 measure: F1 measure is the harmonic mean of precision and recall, which strikes a balance between the two and is particularly effective in cases of class imbalance. The calculation formula for F1- measure is:

$$F1 - measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Or

$$F1 - measure = \frac{2 * TP}{2 * TP + FP + FN}$$

Among them, TP is the number of true cases, FP is the number of false positive cases, and FN is the number of false negative cases. These indicators can comprehensively reflect the accuracy and stability of the model, and are important tools for evaluating the performance of classification models. Through these indicators, we can quantitatively analyze the predictive ability of the model and select the best model configuration and parameters.

(6) Experiment and Comparison: The proposed model will be compared with several other methods, such as logistic regression, deep belief networks, and single-layer CNNs. Through comparison, the effectiveness of the proposed model can be verified.

Through the above steps, an effective machine learning based ECU software defect prediction model can be constructed, which can automatically learn the patterns of defect existence from source code and perform defect prediction in new software projects, thereby improving software quality and reducing potential risks.

Table 1. Algorithm implementation details

Number	Name	Description
1	Feature Extraction	Build AST from ECU software source code, extract features such as node types, quantities, and relationships, and capture code semantic information.
2	Data Preprocessing	Use random oversampling to handle data imbalance and normalize features to eliminate dimensional effects.
3	Model Design	Design a deep learning model based on CNN, using pre trained network structures such as GoogLeNet, and improve performance through transfer learning.
4	Prevent Overfitting	Applying regularization techniques such as dropout to randomly discard neurons and enhance model robustness.
5	Model Training and Evaluation	Train the model using historical engineering data and evaluate its performance through cross validation and metrics (AUC, F1 measure).
6	Experiment and Comparison	Compare the proposed model with other methods such as logistic regression, DBN, and single-layer CNN to validate its effectiveness.

ISSN: 2472-3703 DOI: 10.6911/WSRJ.202411_10(11).0005

In the analysis of ECU software defect prediction models based on machine learning, model optimization strategies are a key step in improving model performance. Optimization strategies involve multiple aspects, including but not limited to learning rate adjustment, batch normalization, regularization techniques, data augmentation, model distillation, etc.

Learning rate is a key hyperparameter that controls the magnitude of parameter updates during the training process of a model. By adjusting the learning rate, gradient vanishing or exploding problems can be avoided, ensuring stable convergence of the model. Common learning rate adjustment strategies include learning rate decay, which gradually reduces the learning rate as the training progresses, and the use of adaptive learning rate algorithms such as Adam, which can automatically adjust the learning rate based on the update history of parameters, thereby accelerating convergence speed and improving model performance. Batch Normalization (BNL) is another commonly used optimization technique that reduces internal covariate bias by normalizing the input of the normalization layer, thereby accelerating training speed and improving model stability. Batch normalization is typically used in the hidden layers of a model to normalize the data of each batch, making the gradient distribution of the model more stable during training. Regularization techniques, such as L1 and L2 regularization, are used to prevent model overfitting. By adding regularization terms to the loss function, the complexity of the model can be penalized, prompting it to learn simpler representations and improve its generalization ability. In addition, Dropout is another effective regularization method that randomly discards a portion of neurons during training, reducing the model's dependence on specific data and enhancing its generalization performance. Data augmentation is the process of transforming training data to generate new data, in order to expand the diversity of the training set. In software defect prediction, small perturbations can be added to code snippets or new data points can be generated by modifying the code structure, thereby improving the model's ability to recognize different types of defects[2]. Model distillation is a technique of transferring knowledge from a large, complex model to a small, efficient model. By training a student model to mimic the output of a teacher model, the student model can learn the generalization ability of the teacher model while maintaining a small model complexity. Finally, model optimization also includes adjusting hyperparameters such as batch size, training period, etc., as well as using automated hyperparameter adjustment tools such as Bayesian optimization to find the optimal combination of hyperparameters. Through the comprehensive application of the above optimization strategies, the performance and generalization ability of the ECU software defect prediction model based on machine learning can be significantly improved.

3. SUMMARY AND PROSPECT

In summary, the machine learning based ECU software defect prediction model proposed in this article achieves efficient prediction of potential defects in ECU software by integrating advanced technologies such as data preprocessing, feature engineering, and deep learning algorithms. The model innovatively integrates source code analysis and static metrics, using CNN to automatically extract code features, significantly improving the accuracy and generalization ability of defect recognition. In addition, by implementing data augmentation and regularization strategies, the model performs well in handling data imbalance and preventing overfitting.

Looking ahead, we plan to further optimize model performance by introducing more software engineering data and adopting more advanced machine learning techniques such as ensemble learning and transfer learning to adapt to more complex practical application scenarios. At the same time, we will continue to pay attention to the interpretability of the model, strive to improve its transparency and credibility, and make it more practical in

ISSN: 2472-3703 DOI: 10.6911/WSRJ.202411_10(11).0005

industrial applications. In addition, we will explore the application of the model in different types of software systems to verify its generality and flexibility.

REFERENCES

[1] Wang Tao, Li Weihua, Liu Zun, et al. Software Defect Prediction Model Based on Support Vector Machine [J]. Journal of Northwestern Polytechnical University, 2011, 29 (6): 7.

[2] Fu Yiqi, Dong Wei, Yin Liangze, et al. A software defect prediction model based on combinatorial machine learning algorithm [J]. Computer Research and Development, 2017, 54 (3): 9.

[3] Wang Tao, Li Weihua, Liu Zun, et al. Software Defect Prediction Model Based on Support Vector Machine [J]. Journal of Northwestern Polytechnical University, 2011, 29 (6): 864-870.

[4] Yu Anlei, Pidchang. Software Defect Prediction Model Based on PSO-BP [J]. Computer Engineering and Applications, 2013, 49 (7): 4.

[5] Liu Yang. Research on Software Defect Prediction Based on Machine Learning [J]. Computer Engineering and Applications, 2006, 42 (28): 49-49.

[6] Wang Yuhong, Fan Jing, Lei Min, et al. Software Defect Prediction Model Based on NPE-SVM [J]. Journal of Chengdu University of Information Science and Technology, 2018, 33 (3): 4.

[7] Xie Huaxiang, Gao Jianhua, Huang Zijie. Software Defect Prediction Model Based on SDL LightGBM Ensemble Learning [J]. Computer Engineering and Design, 2024, 45 (3): 769-776.

[8] Yu Anlei, Pidchang. Software Defect Prediction Model Based on PSO-BP, November 26, 2012 [J]. Computer Engineering and Applications, 2012:64-67.

[9] Xue Canguan, Yan Xuefeng. Software defect prediction based on improved deep forest algorithm [J] ComputerScience,2018,45(8):160-165.

[10] Liu Yang. Research on Software Defect Prediction Based on Machine Learning [J]. Computer Engineering and Applications, 2006, 42 (28): 5.

[11] Wei Liangfen. Research on Software Defect Prediction Technology Based on Machine Learning [J]. Journal of Changchun University, 2017, 27 (10): 4.

[12] Wang Hailin, Yu Qian, Li Tong, et al. Research on Software Defect Prediction Model Based on CS-ANN [J]. Computer Application Research, 2017, 34 (2): 7.

[13] Yu Anlei, Pidchang. Software Defect Prediction Model Based on PSO-BP, November 26, 2012 [J]. Computer Engineering and Applications, 2012:64-67.