Journal of Theoretical and Applied Information Technology
15th February 2024. Vol.102. No 3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

BUILDING A MODEL FOR AUTOMOTIVE SOFTWARE DEFECTS PREDICTION USING MACHINE LEARNING AND FEATURES SELECTION TECHNIQUES

RAMZ TSOULI FATHI[1*], MAROI TSOULI FATHI[1], MOHAMMED AMMARI[1], LAÏLA BEN ALLAL[1]

[1] Materials, Environment And Sustainable Development (MEDD), FSTT, ABDELMALEK ESSAADI UNIVERSITY, Tetouan, Morocco

*E-mail: r.tsouli@uae.ac.ma

ABSTRACT

Over the past few decades, the integration of software into vehicles has experienced exponential growth. This expansion affects nearly all aspects of recent automotive engineering, encompassing various components. Simultaneously, testing becomes increasingly crucial. Significant efforts are required for software verification and validation to meet security, quality, and reliability standards. These essential tests become more complex and costly with the proliferation of functionalities, risking delays in market deployment.

The integration of new technologies, such as predictive analysis based on machine learning algorithms, could anticipate the expected number of anomalies, improving resource management, reducing deployment time, and mitigating the risk of potentially disastrous software errors.

This paper aims to assess the applicability of software defect prediction methods in the automotive domain. To achieve this, we will construct a dataset from real industrial software projects, anonymized and confidential. This effort will culminate in the creation of a new and unique database that will serve as the foundation for our study. Through a series of experiments, we will evaluate the relevance of various machine learning algorithms, aiming to surpass classical approaches in constructing our predictive analysis model.

This paper introduces a novel solution for predicting software faults in the automotive domain. The innovation lies in the realistic approach, relying on the application of complex domain knowledge to a unique database derived from real industrial automotive software projects. Systematic efforts were exerted not only to optimize the usability of this dataset but also to achieve superior performance.

Keywords: *Machine Learning, Features Selection, Predictive Analysis, Automotive Software, Software Metrics, Automotive Software Projects, Software Defect Prediction*

1. INTRODUCTION

Software development, particularly in automotive systems, faces a critical challenge in predicting and addressing defects. With a significant portion of a vehicle's value now attributed to its electronic systems and software-driven functionalities, the automotive industry confronts the need for more sophisticated, reliable, and cost-effective defect prediction methods. Robert Charette [11] compares the complexity between an airplane and a car, demonstrating that it takes more lines of code (LOC) to operate a typical car than an airplane. Dvorak et al. [17] indicate that a representative car from General Motors (GM) went from 10,000 LOC in the 1970s to 100,000 LOC in 2010. Broy [7] asserts that modern premium cars can contain up to 100,000,000 LOC. In earlier work, Broy [1] reveals that the development of electronics and software consumes up to 40% of the total development budget nowadays. Which highlights the pressing need for early-stage defect prediction to ensure quality and reliability.

Numerous approaches and predictive models have been explored in software engineering to anticipate defects. Machine learning emerges as a pivotal tool in this landscape, enabling predictive analysis based on data. By discerning the appropriate mathematical transformations, machine learning models establish

correlations between inputs and outputs, catering to both linear and non-linear relationships.

Model developers face an escalating need for comprehensive insights into their models' learning processes. Various machine learning algorithms like discriminant analysis, support vector machines, logistic regression, or deep learning methods like neural networks, and hybrid methods have been explored in previous studies for software defect prediction. Similarly, an array of feature selection approaches, such as Chi-2 test, evolutionary methods, and hybrid techniques, have been proposed to enhance the result of prediction, but the combination of both have never been experimented with a real-world projects dataset from automotive industry.

Throughout this research, we aim to assess how software defect prediction techniques could be effectively applied within the automotive domain. To achieve this, we will construct a dataset from real industrial software projects, anonymized and confidential due to the sensitive nature of the studied domain. However, the initial analysis of our dataset will reveal the need for several preprocessing steps to make it usable.

In addition to conventional preprocessing techniques, we'll explore methods to reduce the number of features, identifying correlations among them. Three feature selection techniques will be studied, selecting the most suitable for our dataset. We'll then present the results of this method, highlighting key software metrics.

Through a series of predictive analysis method experiments, we'll evaluate the relevance of various machine learning classifiers (e.g., SVM, KNN, LR, RF, XGBoost), to build a strong predictive analysis model.

The performance assessment encompasses, confusion matrices, and ROC curve and AUC score to compare the effectiveness of these methods on both raw and new datasets. The resulting selected features are subsequently utilized to compare the performance of the classifiers, providing a comprehensive insight into their comparative efficacy.

2. SUBJECT AREA

The importance of electronics in the automotive industry is on the rise, with car manufacturing also following this trend. Nowadays, cars heavily rely on electronics, and most of the added value in modern automotive comes from the integration of over 50 embedded computers known as Electronic Control Units (ECUs) and numerous sensors within the vehicle's physical system [28] [3]. These ECUs require embedded software to connect and function within the car network. To gain a competitive edge, car manufacturers focus on developing software that enables new safety, comfort, and economy functions, resulting in more complex software development. However, this complexity brings challenges such as increased time and cost pressures and the demand for high-quality software.

One of the most challenging aspects within software engineering is automotive software development. Software errors and misunderstandings of specifications often arise in the late phases of the car's life cycle, typically occurring when the car is almost ready for market launch.

In the realm of software defect prediction, research has flourished due to the growing complexity of software. Detecting software failures plays a crucial role in ensuring software quality and reducing the risk of bugs that could lead to incorrect or unexpected behavior. Traditionally, statistical-based approaches such as logistic regression, classification trees, and logical classification methods were used to predict software defects [10]. However, these methods have their limitations, and alternative approaches have been explored.

Machine learning techniques have gained popularity in recent times for software defect prediction [24] [9]. The process involves several steps: first, the construction of a dataset, which can be collected from car manufacturing archives or software suppliers, and issue tracking systems used by software developers can serve as valuable sources for the dataset. Next, feature extraction techniques are applied to identify key features representing software complexity and development process. Once the dataset is labeled, a model is constructed and trained using the selected features. Finally, the model is evaluated using a test set, and predictions can be made based on the trained model.

Through this approach, the automotive industry can anticipate potential software challenges and optimize development processes to ensure robust performance. The use of machine learning techniques empowers the system to predict the software's reliability and functionality, ultimately contributing to improved safety, comfort, and overall performance of vehicles.

In our previous work we experimented several supervised machine learning algorithms on publicly available dataset, our main goal was to experiment those algorithms to test their performance [18]. In

this paper we build a model based on machine learning algorithms and feature selection techniques adapted to perform on an industrial dataset.

3. METHODOLOGIE

Most original equipment manufacturers (OEMs) in the automotive industry adopt Model-Driven Development (MDD). In both literature and development standards, the software development lifecycle in the automotive domain is depicted through approaches based on the V-model [15], [2].

The process followed in each iteration within the production software development phase can be described using a V-model, essentially comprising the following steps for each iteration: first, requirements are defined or revised, followed by System Design (functional design and system architecture). After system design, Electronic Control Unit (ECU) specifications are established, which can also be termed as software design because the software is usually designed for specific ECUs and is commonly co-developed and optimized for particular functionalities.

Next comes the implementation, where the designed software is put into action (in the form of code, either manually written in an object-oriented language or automatically generated from a functional model built using a domain-specific language like Matlab/Simulink). According to Broy et al. [8] and Bock et al. [5], the majority of automotive software is developed using graphical or model-driven programming approaches, such as Matlab Simulink [26], with automatic code generation, for instance, using TargetLink [16]. Following this process, correlated code and model files are available during the development stages.

The objective of this study is to construct a dataset from industrial software projects, then to develop a predictive analysis system for automotive software by establishing connections between various software metrics and software performance. To achieve this, we collected extensive datasets from OEM archives, encompassing information about software defects, development processes, and other relevant metrics. Through feature extraction techniques, key indicators of software complexity, structure, etc... are identified.

First, this paper aims to demonstrate the benefits of feature selection from complexe dataset of automotive software, in order to conduct an analysis for fault prediction and to introduce a robust machine learning model. To achieve these objectives, we define two hypotheses.

The first hypothesis: The choice of feature subsets significantly impacts defect classification performance.

The second hypothesis: Combine strong classifiers will lead to a stronger one that will demonstrate superior performance compared to individual classifiers when applied to our dataset.

3.1 Data-related Challenges

Confidentiality is the first difficulty faced on this domain. Our work is based on dataset acquired from four different automotive embedded software projects that have been developed for an automotive constructor within last seven years.

From these projects we generated a new database based on software metrics. We acknowledge the limitations of relying on a single metric to encompass all aspects of interest. As a result, we opt to utilize a comprehensive set of metrics that specifically target internal attributes, such as code structure, size, and complexity.

These selected metrics are chosen based on our expertise and with a primary objective of encompassing fault-related aspects of the source code that can be automatically measured using internal product metrics. By employing this diverse set of metrics, we aim to gain a deeper understanding of the software and identify areas that require attention in the context of fault prediction.

3.2 Feature Selection

In software fault prediction, features correspond to software metrics extracted from the source code. However, certain features might be redundant or irrelevant, necessitating the removal of the latter. In fact, this preprocessing step is anticipated to significantly enhance the classification performance. In this section we will present the methods that we used to optimize our dataset.

Correlation-based feature selection ranks features based on their correlation with the target variable. It measures the linear relationship between each feature and the target and features with higher correlation values are considered more relevant.

First, it calculates the correlation coefficient between each feature and the target variable. The correlation coefficient is calculated within various methods: Pearson correlation (for continuous variables) and Point-Biserial correlation which is a special case for a binary target variable and a continuous feature, in our case, the second approach is more adapted. [23]

Point-Biserial Correlation Coefficient measures the correlation between our target variable and a continuous variable. This coefficient quantifies the strength and direction of the linear relationship between the target and feature.

Then, we rank the features based on their correlation coefficients, features with higher absolute correlation values are considered more relevant to the target.

Even if, it only captures linear relationships, and may not consider non-linear dependencies between features and the target, this method is simple and computationally efficient. And can be used for both regression and classification tasks.

Recursive Feature Elimination (RFE) is a wrapper method that recursively removes less important features by training the model iteratively and selecting the features with the lowest importance scores.

Initially, the model is trained on the entire set of features. After the model is trained, it ranks the features based on their importance or contribution to the model's performance. This ranking is typically obtained from the model-specific feature importance scores. The least important feature(s) according to the ranking are removed from the dataset. The model is retrained on the reduced feature set (with the least important features removed). We iterate until the model's performance no longer improves. [23]

This method accounts for inter-feature dependencies, can work with any machine learning algorithm that provides feature importance scores.

But it is more computationally intensive compared to correlation-based feature selection and forward selection.

Last, forward selection starts with an empty feature set and iteratively adds one feature at a time based on its individual performance in improving the model's predictive accuracy.

It is a simple and computationally efficient method, that can be effective for small to medium-sized feature sets.

However, it may not always find the globally optimal feature subset. And can be prone to overfitting with a large number of features [21].

3.6 Software Metrics Employed for Data Construction

In every engineering domain, including computer science, objective measurement is essential for rating and evaluating processes and systems. Software metrics, which measure the level of input belonging to a system product or process, play a crucial role in this context. For software metrics to be truly valuable, effective characterization and validation are necessary to establish their proven worth.

Radjenovi´c et al. [29] presented an overview of software metrics and studied their applicability in software fault prediction. They categorized software metrics on three categories: traditional, object oriented and process metrics. Their overall analysis on the usage of metrics showed metric effectiveness in terms of size, programing language and overall.

Based on the result of the study, to calculate the complexity of the program Mc Cabe cyclomatic complexity (CC) [27] was used. For size and complexity measures Halstead's software metrics [22] were used. Even if the study presented weak effectiveness of those metrics, we consider them due to their simplicity of calculation, ease to use and language independent, not to mention that we wanted to confirm one the hypothesis, the correlation between automotive software complexity and effort and fault tendency.

For the analysis of classes, three popular Chidamber and Kemerer (1994) [12], [14], were used WMC metric (Weighted Methods per Class) is defined as the sum of methods complexity CBO (Coupling between objects) and RFC (Response for a class) measures the coupling. Chidamber and Kemerer metrics are not only the most used in studies, but also demonstrated great effectiveness.

And finally, Lines of Code (LOC) is any line of text in a code that is not a comment or blank line, due to its sole focus on code volume, Lines of Code (LOC) can only be utilized for comparing or estimating projects written in the same programming language and adhering to identical coding standards.

3.6 Presentation of Mccabe Selected Metric

McCabe Cyclomatic Complexity (CC) is a widely adopted software metric that plays a vital role in measuring the complexity of a software system. Introduced by Thomas

J. McCabe in 1976, CC provides valuable insights into the maintainability, testability, and overall quality of software code.

McCabe Cyclomatic Complexity stands out as a powerful and widely used measure to effectively manage and evaluate software complexity. Software complexity is a critical aspect that significantly influences software quality and maintainability.

McCabe Cyclomatic Complexity is based on the graph-theoretic approach. The complexity of a software program is determined by calculating the number of independent paths within its control flow graph [27]. The formula for calculating McCabe Cyclomatic Complexity is as follows:

$$CC = E - N + 2P \quad (1)$$

Where:

E Represents the number of edges N Denotes the number of nodes

P Signifies the number of connected components

3.5 Presentation of Halstead selected metrics

Halstead software metrics, proposed by Maurice H. Halstead in 1977, have become widely recognized and adopted for quantifying various aspects of software complexity and performance.

Halstead software metrics offer a valuable set of measures to quantitatively evaluate software programs based on their code structure and implementation characteristics. The metrics are grounded in information theory and aim to measure the volume, effort, and difficulty of software code. they present complex metrics based on the

number of operators and operands. [22]

- n1: the number of unique operators
- n2: the number of unique operands
- N1: the total number of operators
- N2: the total number of operands

3.5.1 Halstead Program Volume

The Halstead program volume (V) serves as an indicator of program size and is computed using the following formula [22]:

$$V = N * \log2(n) \quad (2)$$

Where:

N Represents the total number of program operands and operators n Denotes the number of distinct program operands and operators

Halstead program volume quantifies the raw size of a program, regardless of its functionality.

3.5.2 Halstead Program Difficulty

Halstead program difficulty (D) is a measure of how challenging it is to comprehend the code logic and is calculated as [20]:

$$D = (n1/2) * (N 2 / n2) \quad (3)$$

Halstead program difficulty provides valuable insights into the complexity of the software code, aiding in identifying areas that may require improvement or refactoring.

3.5.3 Halstead Program Effort

Program effort (E) provides insights into the implementation effort required for a software program and is determined by the following equation [20]:

$$E = V / D \quad (4)$$

Where:

D Is the program difficulty

Halstead program effort represents the human effort needed to understand, implement, and maintain the software code.

Halstead software metrics offer valuable tools for quantitatively evaluating software programs based on their size, effort, and complexity.

3.6 Presentation of Chidamber and Kemerer selected metrics

Chidamber and Kemerer software metrics, proposed by S.R. Chidamber and C.F. Kemerer in 1994 [12], have gained significant recognition in the field of software engineering for assessing software quality and complexity.

The metrics aim to evaluate object-oriented software design characteristics and identify potential areas for improvement.

3.6.1 Weighted Methods per Class (WMC)

The Weighted Methods per Class (WMC) metric is used to measure the complexity and size of a class by calculating the sum of complexities of all its methods [13]. The formula for calculating WMC is as follows:

$$WMC = \Sigma \text{ complexity(method)} \quad (5)$$

Where "complexity(method)" the complexity of each local method is calculated using McCabe's cyclomatic complexity.

The WMC is not really an OO metric, but it is more of a size or complexity metric, depending on implementation.

3.6.2 Coupling Between Objects (CBO)

Coupling Between Objects (CBO) measures the degree of interdependence between classes in a software system [4]. It quantifies the number of other classes to which a class is coupled. In other words,

CBO indicates how many other classes the class in question interacts with.

CBO is an essential metric to assess the maintainability and flexibility of a software system. Lower CBO values indicate better design and reduced interdependence.

3.6.3 Response for a Class (RFC)

Response For a Class (RFC) is used to determine the number of methods in a class that can potentially be executed in response to a message [30]. It quantifies the ability of a class to respond to various events and messages. The formula for RFC is as follows:

RFC = Number of Methods Receiving a Message (6)

When calculating RFC for a class, we consider all the methods in that class that are triggered in response to messages coming from outside the class. These messages can be in the form of method calls, event notifications, or other forms of communication between classes.

RFC provides insights into the complexity and responsiveness of a class, helping in identifying classes that may require reorganization or better cohesion.

4. SYSTEM DESIGN TO BUILD OUR MODEL

4.1 Design methodology model

To address the software defect prediction problem, we followed a structured process, the proposed framework is described in Figure 1. Firstly, we collected the necessary data for our analysis, an essential phase in any machine learning problem. Next, we conducted features extraction and definition to create an analytical dataset, enabling us to identify the relevant software defect factors.

Due to excessive size, redundant, and irrelevant features of the original dataset, feature selection techniques were used to optimize it. To validate our primary hypothesis, machine learning algorithms were applied to both, the original dataset and to the optimized datasets resulting from the three distinct feature selection techniques. Comparative analysis of model performance across the three optimized datasets allowed us to identify the most suitable dataset resulting from the best-performing feature selection technique for our case study, which we further utilized in the subsequent steps.

Data pre-processing was then performed to prepare the data for machine learning algorithms. The dataset was split into a training set (70%) and a testing set (30%).

Following this, our pursuit to construct automotive software defect prediction models led us to create an ensemble classifier integrating six distinct learning algorithms: logistic regression, SVM, KNN classifier, random forests, gradient boosting, and XGBoost. These classifiers have diverse yet complementary performances.

For robust model validation, we utilized techniques like Confusion Matrices, which helped in evaluating the models' performance by analyzing their classifications. The corresponding performance of our model and of two basic classifiers is assessed in terms of the area under the receiver operating characteristic (ROC) curve (AUC) scores, which is widely used to assess the performance.

Figure 1: Software defect prediction model design

4.2 Data Pre-processing

Before applying machine learning algorithms, several data treatments were performed to prepare the analytical dataset. This involved handling null values, treating outliers, and addressing garbage values. The resulting dataset consisted of 8954 rows and 8 columns.

Subsequently, as specified feature selection was conducted to enhance the classification performance and increase the predictive capacity of the model. We considered three feature selections models that helps us first limit the number of features on 10 (8 relevant, 1 description and target variable, and consider the most relevant features for our target variable. Details of this experiment will be included in result section (5.2.1).

4.3 Feature description

This section presents the features selected to conduct the second part of our experiment; these are the result of feature selection:

Table 1: Presentation of selected features

Feature	*Description*
SubProject_num	Subproject number

CC	Mc Cabe cyclomatic complexity (CC)
LOC	Lines of Code
H(V)	Halstead volume
H(D)	Halstead difficulty
H(E)	Halstead effort
WMC	Weighted Methods per Class
CBO	Coupling between objects
RFC	Response for a class
Bug	Bug presence (booleen)

4.4 Ensemble learning

In this research study, we present an ensemble of classifiers tailored to address the intricate challenge of robust software defect classification. For our proposed model, the final prediction is determined by averaging the predictions from all the base classifiers. This ensemble approach is strategically designed to mitigate common issues encountered within software defect datasets, such as data imbalance. Notably, the ensemble's robustness is fortified through the averaging of classification outcomes across multiple classifiers, which systematically mitigates uncorrelated errors, leading to a marked enhancement in the overall classification performance.

Within our proposed Ensemble model, each individual classifier within the ensemble is tasked with predicting the probability that a software component pertains to the fault class. Subsequently, these output probabilities are averaged to yield the final probability estimation. A visual representation of the proposed model is described in Fig. 1, wherein six distinct base classifiers are synergistically amalgamated to constitute the model. These base classifiers are logistic regression, SVM, KNN classifier, random forests, gradient boosting and XGBoost.

Each of the base classifiers employed within our proposed ensemble learning model is introduced in detail in the following sections.

4.4.1 Logistic regression

Logistic regression relies on the logistic function, also known as the sigmoid function:

$$1/ (1 + e - x) \quad (6)$$

The logistic function is represented by an S-shaped curve that can take any real value and map it to a value between 0 and 1.

The coefficients (beta values) need to be estimated. The model's output for each input value,

$$X = [x1, x2, ..., xn], \quad (7)$$

is calculated as follows:

$$y = 1/(1 + e - (\beta0 + \beta1x1 + \beta2x2 + ... + \beta nxn)) \quad (8)$$

Therefore, we need to estimate all the coefficients of the model:

β0is the intercept.

β1 is the coefficient for x1.

Bn is the coefficient for xn.

These coefficients are determined by maximizing the likelihood function, which involves maximizing the probability of the observed data:

$$\max(\beta0, \beta1) = Y(P (Y = yi) | (X = xi)) \quad (9)$$

To maximize this function, we take its derivative and find the points where it equals zero. Since the derivative of a product is complex to solve, we apply the logarithm to transform this product into a sum:

$$\max(\beta0, \beta1) = \max(\log(\beta0, \beta1)) \quad (10)$$

Several models are available to find the beta values that maximize this function (such as the Logit model from Statsmodels or LogisticRegression from scikit-learn). Here, we apply the scikit-learn model.

These models take various hyperparameters, which can be user-defined or set to default values. The goal is to find the best model, meaning the hyperparameters that yield the best results. To achieve this, we use GridSearchCV, allowing us to obtain the optimal model with the available data.

4.4.2 SVM

Support Vector Machines (SVM) are classification methods that extend linear classifiers. These methods essentially create a line that minimizes the distance between data points. The placement of this line depends on the support points, which are the points from each of the two distinct classes that are closest to each other. The distance between points is calculated using the L2 norm.

The distance between the line (M = (Mx, My)) and a support point (D (a, b)), characterized by a directional vector ax + by + c = 0. It's worth noting that regardless of the number of dimensions, this formula can be generalized for a plane of any dimension.

Furthermore, SVMs can separate various types of data, not limited to linearly separable data, by

constructing a hyperplane (using a kernel function) to transform non-linearly distributed data into linearly distributed data. In other words, this method transforms" the space of data points into a space where these points are linearly separable, making separation straightforward. The kernel function is responsible for this transformation.

4.4.3 KNN Classifier

K-Nearest Neighbors (KNN) classification involves placing a point from the dataset X into a space of the same dimension as the number of features in X. The estimated Y value is determined by considering the k nearest neighbors to this new point and the Y values associated with those neighbors. Consequently, the Y value associated with this new point is a result of a "majority vote" among its k closest neighbors. If the majority of these neighbors are classified as 1, then this point will also be classified as 1.

4.4.4 Random Forest

Random Forests consist of multiple unpruned classification or regression trees, and these trees are constructed using random feature selection from bootstrap samples of the training data [6]. In a classification problem, each data sample is passed through every tree in the random forest. Each tree independently classifies the data sample, and the random forest makes its final decision by selecting the class that receives the majority of votes from the individual trees. This ensemble approach has been shown by Breiman [6] to be effective, with error rates depending on the individual tree's strength and the correlation between the trees in the forest. One notable advantage of random forests is their ability to handle high-dimensional data and mitigate overfitting, thanks to the feature subsets and bootstrapped samples used in each tree's construction. Despite their effectiveness, results derived from random forests can be challenging to interpret due to the complexity of the ensemble.

4.4.5 Gradient boosting

Gradient boosting, introduced by Friedman [19], is a powerful technique for regression problems. It utilizes an ensemble of weak predictors, typically decision trees. The algorithm iteratively minimizes the error function through gradient descent. The gradient provides the direction for descending towards the minimum. The challenge lies in avoiding local minima and selecting an appropriate learning step size to prevent overshooting. For classification problems, gradient boosting can be applied by transforming them into regression tasks using an appropriate loss function [13], as expressed in the formula below:

$$f(x) = w0 + w1h1(x) + w2h2(x) + ... + wnhn(x) \quad (11)$$

where hi(x) is the output decision of the ith individual tree. The weights, wi, applied on the individual decisions are optimized by minimizing a differentiable loss function.

4.4.6 XGBoost

XGBoost (Extreme Gradient Boosting) is an efficient implementation of gradient boosting, renowned for its high performance and unique features. It improves upon traditional gradient boosting with algorithmic enhancements such as regularized objectives, parallelization, and advanced pruning to combat overfitting. XGBoost handles missing data, incorporates L1 (Lasso) and L2 (Ridge) regularization, and employs an exact greedy algorithm for precise tree construction. Its scalability and speed make it a popular choice. While XGBoost is a standout implementation, gradient boosting encompasses other variations like LightGBM and CatBoost, each tailored to specific needs. The choice depends on factors like dataset size and problem requirements.

5. RESULT AND DISCUSSION

5.1 Conducting the Experiment

Our experimental design comprehensively addresses various performance factors, including the effectiveness of feature selection techniques and the ensemble learning model introduced in this study. All the algorithms under evaluation were implemented using the Python programming language for consistency and ease of comparison.

We employed the AUC (Area Under the Curve) score as the primary metric for assessing classification performance. The AUC score provides a robust measure of the model's discriminative power and ability to distinguish between classes.

To benchmark the performance of our proposed model, we compared it against the classifiers, XGBoost and random forests. These classifiers, previously utilized in our earlier work [18], have exhibited promising results, serving as strong reference points for our evaluation.

Furthermore, this study assesses feature selection techniques to investigate Hypothesis 1 and determine their impact on classification performance.

5.2 Analysis and discussion

In this section, we will be presenting the analysis of the experiment and the discussion. To verify the hypothesis stated in section 2 we will evaluate first the impact of feature selection on our model, then we will also be evaluating the performance of ensemble learning model.

5.2.1 Feature Selection Performance Analysis

Figure 2 illustrates the impact of three proposed feature selection techniques, Correlation based selection (Point-Biserial correlation), Recursive Feature Elimination (RFE), and Forward selection, on defect classification performance, as evaluated using the AUC measure.

Figure 2 uncovers a noteworthy trend in the classification performance achieved by the Recursive Feature Elimination (RFE) technique. Remarkably, RFE not only surpasses both the PBC and Forward selection methods in our datasets but also achieves the highest AUC values while utilizing the fewest number of features to attain optimal classification accuracy (only 8 features).

The list of features resulted by RFC technique is presented in section 3.3. this is the adopted method that we used in pre-processing our dataset that we be used to compare the performance of our model. Which indicates, that by employing a compact yet highly informative feature subset, our classifier achieved notable AUC scores of 0.84.

These results, providing empirical validation for Hypothesis 1. As anticipated, the strategic application of feature selection, specifically the Recursive Feature Elimination (RFE) technique in our study, not only substantially enhanced classification performance but also effectively reduced the requisite number of features to attain these enhanced results. Consequently, we find Hypothesis 1 to be supported by our findings.

5.2.2 Proposed model Performance Analysis

We present result of our six classifiers and our model in Figure 3. As previously mentioned, the ensemble learning model we have introduced comprises six base classifiers. These classifiers collectively contribute to the final classification decision by averaging their output probabilities. we choose to compare firstly the classifiers to our model by accuracy.

Overall, the results obtained are very interesting, we can observe that our model acquires the best result, and KNN classifier obtained the weaker outcome. The proposed model has outperformed all classifiers even the strong one of them like XGBoost and SVM.

In the context of boosting ensemble learning, the weighted combination of other learners has the effect of adjusting decision boundaries to accommodate minority class samples more effectively. This weighted combination is a result of each learner attempting to correct errors introduced by the others in the ensemble. Beyond addressing imbalanced data, boosting ensemble methods can also improve their robustness when faced with the presence of redundant or irrelevant features. These characteristics make boosting ensembles a robust choice for tasks like software defect classification.

In conclusion, the performance outcomes, as presented in Figure 3, provide confirmation for the hypothesis 2 advanced in this study. As hypothesized, ensemble learning exhibits superior classification accuracy when compared to individual classifiers, particularly when the selection of base classifiers is grounded in their established performance.

As a way to use our model with confidence, the model performance should be evaluated.

Confusion matrices that include information about actual and predicted model outputs are shown in Table 2.

Several model performance evaluation metrics can be generated from the confusion matrix as sensitivity, specificity, precision. . .

We also used Receiver Operating Curve (ROC to measure the classification effectiveness of our predictive model run with training and testing data.

ROC plots the sensitivity (true positive) versus specificity (false positive) of the model.

Table 2. Confusion matrix for proposed model

		Predicted output	
		+	-
Actual Output	+	TP= 680	FN=5
	-	FP=25	TN=2678

The enhancement in performance resulting from the feature selection process, which retains only relevant software metrics, reveals an intriguing finding deserving of analysis.

6. CONCLUSIONS AND FUTURE WORK

Machine learning techniques have demonstrated significant promise in addressing such problem-solving tasks, particularly considering the continual expansion of data within organizations. This paper

explored a dataset obtained from real industrial projects within the automotive industry, based on standard software development processes. The outcome was an unbalanced dataset that required adaptation. Initially, data treatment techniques were employed, followed by feature selection methods, aiming to enhance the performance of machine learning algorithms. Our study focused on various feature selection methods for software defect prediction, showing that the selection of a quality features significantly improves the AUC, this result was obtained by comparing the accuracy and the AUC result of two algorithms on the raw dataset (before applying any features selection techniques) and the obtained dataset. Subsequently, this dataset was used to train an ensemble model containing robust classifiers (both basic and ensemble). The obtained result was highly satisfactory, our solution demonstrated impressive performance, reaching nearly 94% accuracy.

The model's performance was evaluated using multiple metrics on test data. Our observations from the confusion matrix and the ROC curve instill confidence in the model's accuracy and effectiveness. However, further analysis is imperative as this domain lacks extensive studies on this topic to our best knowledge. For future investigations, we aim to explore larger datasets to validate the performance of our model. We also want to experiment cross-organizations projects to test our model.

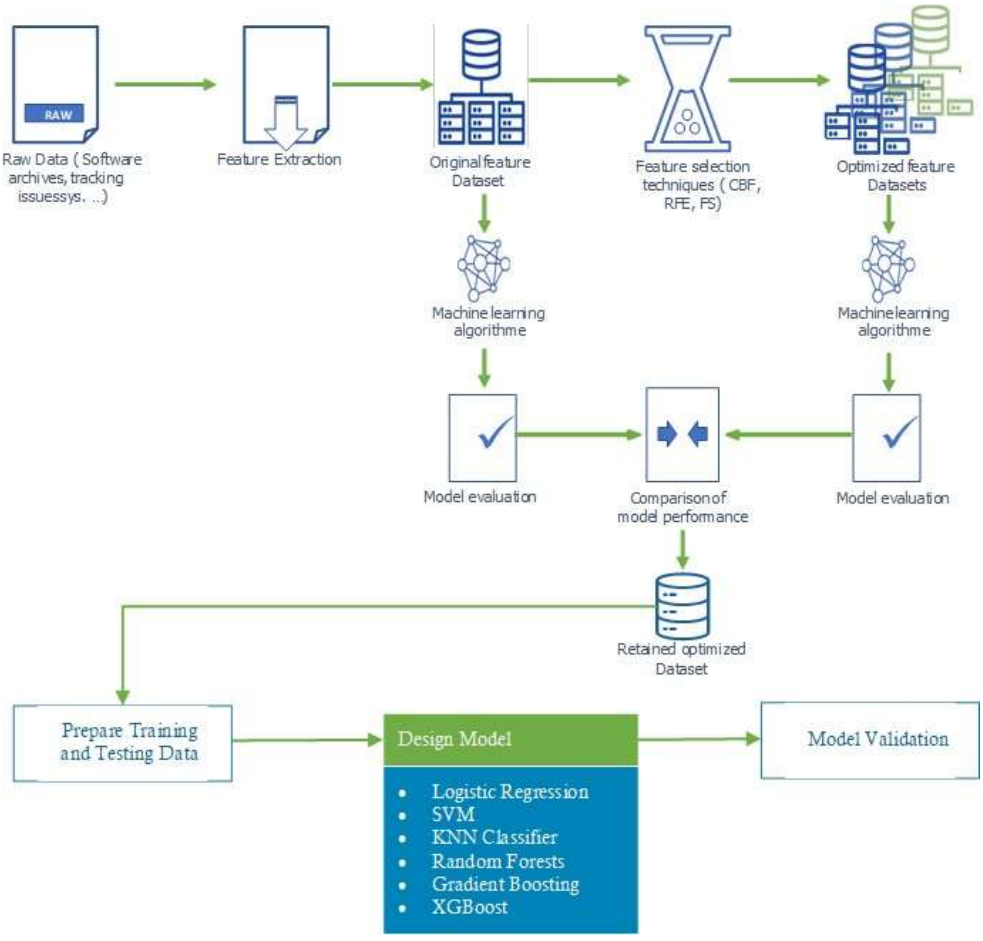

Figure 1: Software defect prediction model design

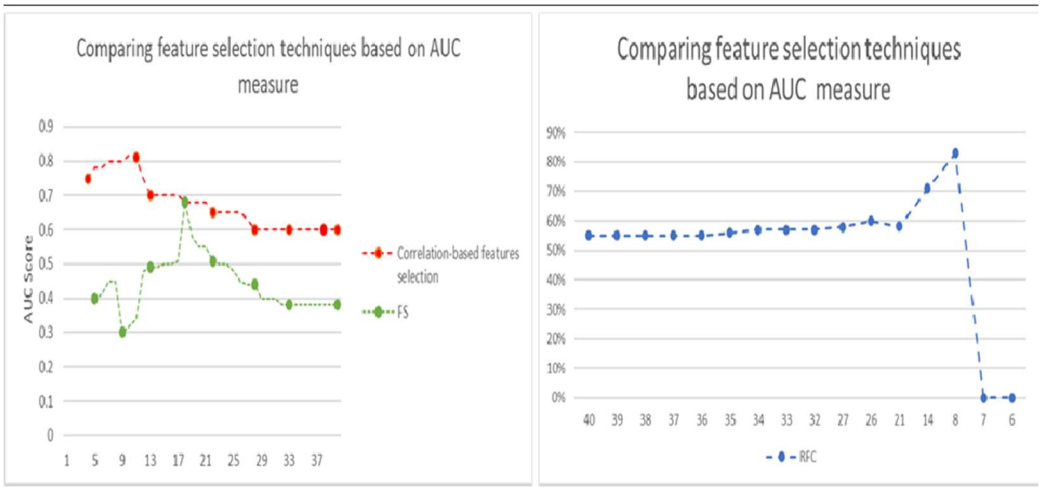

Figure 2 : Comparing Feature Selection Techniques Based On AUC Score

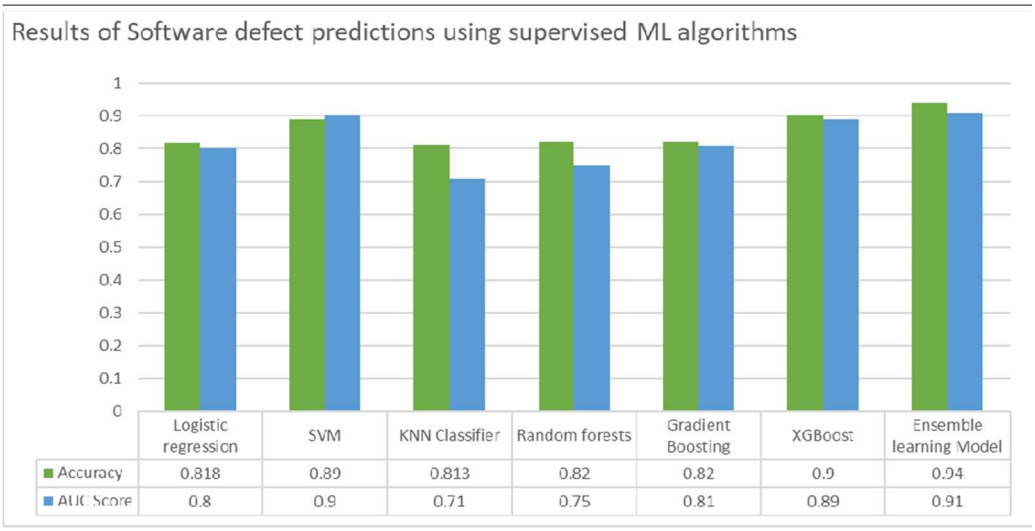

Figure 3: Results Of Software Defect Predictions Using Supervised ML Algorithms

REFERENCES:

[1] (2006) Challenges in automotive software engineering. In: Proceeding of the 28th international conference on Software engineering. ACM, pp 33–42

[2] Vinoth Kannan, K. (2021). Model-Based Automotive Software Development. In: Kathiresh, M., Neelaveni, R. (eds) Automotive Embedded Systems. EAI/Springer Innovations in Communication and Computing. Springer, Cham. https://doi.org/10.1007/978-3-030-59897-6_4

[3] Aoyama M (2011) A Design Methodology for Real-Time Distributed Software Architecture Based on the Behavioral Properties and Its Application to Advanced Automotive Software. 18th Asia-Pacific Software Engineering Conference

[4] Basili VR, Briand LC, Melo WL (1996) A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Transactions on Software Engineering 22(10):751–761

[5] Bock F, Homm D, Bock, F., Homm, D., Siegl, S., & German, R. (2016). A taxonomy for tools, processes and languages in automotive software engineering. arXiv preprint arXiv:1601.03528.

[6] Breiman L (2001) Random forests. Mach Learn 45:5–32

[7] Broy M (2006) Challenges in automotive software engineering. Proceedings of the 28th international conference on Software engineering pp 33–42

[8] Broy M, Kruger I, Pretschner A, et al (2007) Engineering automotive software.
Proceedings of the IEEE 95(2):18–9219

[9] C (2011) Software fault prediction: A literature review and current trends / Expert Systems with. Applications 38:4626–4636

[10] Catal C (2011) Software fault prediction: a literature review and current trends. Expert Syst Appl 38(4):4626–4636

[11] Charette RN (2009) This car runs on code. IEEE Spectrum 46(3):3–3

[12] Chidamber SR, Kemerer CF (1991) Towards a metrics suite for object-oriented design. Oopsla 91 Conference Proceedings: Object-Oriented Programming Systems, Languages, and Applications

[13] Chidamber SR, Kemerer CF (1994) A Metrics Suite for Object-Oriented Design.
IEEE Transactions on Software Engineering 20(6):476–493

[14] Chidamber SR, Kemerer CF (1994) A metrics suite for object-oriented design. IEEE Transactions on Software Engineering 20:476–493

[15] Dieterle W (2005) Mechatronic systems: Automotive applications and modern design methodologies. Annual Reviews in Control 29(2):273–277

[16] DspaceURL https://www.dspace.com/fr/gmb/home/products/sw/ pcgs/targetli.cfm

[17] Dvorak, D. (2009, April). NASA study on flight software complexity. In AIAA infotech@ aerospace conference and AIAA unmanned... unlimited conference (p. 1882).

[18] Fathi RT, Fathi MT, Ammari M, et al (2023) Machine Learning Algorithms for Automotive Software Defect Prediction. In: Kacprzyk, J., Ezziyyani, et al (eds) International Conference on Advanced Intelligent Systems for Sustainable Development. AI2SD 2022, vol 637. Springer

[19] Friedman J (2002) Stochastic gradient boosting. Comput Stat Data Anal 38:367– 378

[20] Ghosh S, Majumdar S, Saha RK (2017) An Empirical Validation of Halstead Metrics for Predicting Program Effort. Procedia Computer Science 115:586–593

[21] Mark A. Hall, "Correlation-based Feature Selection for Machine Learning" by Mark A. Hall, a thesis submitted at The University of Waikato.

[22] Halstead, M. H. (1977). Elements of Software Science (Operating and programming systems series). Elsevier Science Inc..

[23] Laradji IH (2014) Software defect prediction using ensemble learning on selected features. Inform Softw Technol

[24] Lessmann S (2008) Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering 34(4):485–496

[25] Mark, Hall A, W. Dieterle, "Mechatronic systems: Automotive applications and modern design methodologies," Annual Reviews in Control, vol. 29, no. 2, pp. 273–277, 2005.

[26] Mathworks T, Simulink URL http://fr.mathworks.com/products/simulink

[27] Mccabe TJ (1976) A Complexity Measure. MITRE Corporation Technical Report (185)

[28] Pretschner A (2007) Software Engineering for Automotive Systems: A Roadmap. Proc ICSE pp 57–71

[29] Radjenovic´D, Hericvko M, Torkar R, et al (2013) Software fault prediction metrics: A systematic literature review. Information and Software Technology 55(8):1397–1418

[30] Rattan D, Kumar R (2017) Evaluating Software Complexity Metrics for Object-Oriented Software Systems. Procedia Computer Science 115:356–363