JOIV : Int. J. Inform. Visualization, 6(4) - December 2022 921-930

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Software Defect Prediction Framework Using Hybrid Software Metric

Amirul Zaim [a], Johanna Ahmad [a,*], Noor Hidayah Zakaria [a], Goh Eg Su [a], Hidra Amnur [b]

[a] *School of Computing, Universiti Teknologi Malaysia, Johor, Malaysia*
[b] *Department of Information Technology, Politeknik Negeri Padang, Limau Manis, Padang, 25164, Indonesia*
*Corresponding author: *johanna@utm.my*

***Abstract*—Software fault prediction is widely used in the software development industry. Moreover, software development has accelerated significantly during this epidemic. However, the main problem is that most fault prediction models disregard object-oriented metrics, and even academician researcher concentrate on predicting software problems early in the development process. This research highlights a procedure that includes an object-oriented metric to predict the software fault at the class level and feature selection techniques to assess the effectiveness of the machine learning algorithm to predict the software fault. This research aims to assess the effectiveness of software fault prediction using feature selection techniques. In the present work, software metric has been used in defect prediction. Feature selection techniques were included for selecting the best feature from the dataset. The results show that process metric had slightly better accuracy than the code metric.**

***Keywords*— Software fault prediction; machine learning; object-oriented metric.**

Manuscript received 7 Jan. 2022; revised 28 Aug. 2022; accepted 19 Nov. 2022. Date of publication 31 Dec. 2022.

I. Introduction

Software development has accelerated significantly during this epidemic. Due to the COVID-19's significance, several software packages have been presented globally at various levels to upgrade the guide's format. The complexity and size of the software program influence the amount of time spent testing and introducing novel insects or unforeseen flaws. This significantly burdens the software industry to increase disease-free software, particularly for mission-critical software. However, software program failures are frequent, and the security segment of software program tasks should emerge as unquestionably painful for customers and luxurious for businesses. Occasionally, the exact insects mentioned are in motion during the protection section. Consequently, a software fault prediction model is desired to anticipate malfunctioning modules in software to avoid an increase in the rate of basic task completion [1].

The error prediction feature enables the development team to perform additional checks on modules with a high probability of failure [2]. Numerous error prediction methods, such as Bayesian networks [3] and K Means clustering [4], have been developed to improve the efficiency of application error prediction. However, selecting software metrics is the most difficult component of designing an error prediction model [5]. Selection measures evolved into a critical component that aided in constructing the default prediction model [6]. To assess product quality, software metrics are required, and a variety of aspects will be considered, including architectural design, computational complexity, software validity, and efficiency.

Generally, metric software property has method-level, class-level, components-level, file-level, process-level, and quantitative-level categories. Method-level metrics are widely used for software fault prediction problems [7]. Many past researchers use numerous software metrics such as CK metrics [8], Briand Metrics [9], and design metrics [10]. This metric was categorized into method and class level metric, an object-oriented design.

Metrics for software are divided into two main categories: process metrics and code metrics [5]. Process metric can be defined as a change of information in software or code [11], while code metric or product metric can be defined as an indicative measurement of faults that provides a quantitative description of certain characteristics of software products and processes [11]. Object-oriented metrics are a subset of code metrics and are critical in the defect prediction process. However, the main problem is that most fault prediction models disregard object-oriented metrics. In addition to object-oriented metrics, the usage percentage of component levels and process level metrics are very low [10].

Different evaluations were performed, including metric-type process metric and code metric comparisons, category-wise comparisons, and individual metric comparisons [11]. The selected performance metric which is Mean Square Error (MSE), Root Relative Square Error (RRSE), and Average Absolute Error (AAE). The result shows that the selected hybrid set obtained using wrapper subset selection performed very well compared to all metric sets for predicting the number of faults, and the process metric performed better than the code metric [11]. There were limitations in this research, such as the imbalance class distribution within the dataset.

Furthermore, the imbalance class distribution within the dataset resulted from the huge dataset used in the feature selection process [12]. If the number of specimens in one class is greater than that in another class, then a dataset is said to be highly divergent. The major class is used to detect unbalanced datasets with a greater number of samples, whereas the minor class expresses the samples as positive. [14]. By class ratios of 100 to 1 and 1,000 to the quantity of majority class specimens outnumbers minority class specimens. Both binary and multi-class datasets contain imbalanced data [15]. Unbalanced data is particularly common in machine learning and data mining applications, as it occurs in several actual prediction jobs. However, the approaches and notion of data balancing prior to model development are relatively new to many academics in information systems [16]. In several fields, such as medical diagnosis [17], classifiers for database marketing, property refinancing prediction [18], and classification of weld faults, numerous balancing strategies have been used to data sets.

Even though object-oriented paradigm is extensively employed in the business world, the utilisation rate of class level metrics is still beyond acceptable limits [6]. Important is the prediction model that uses class-level measurements to predict errors during the design process; this sort of prediction is known as early prediction. Due to realizing the importance of software defect prediction and the importance of object-oriented concepts in the software development phase, this research aims to propose a software defect prediction model using the object-oriented metric that focuses on the early stage of software development. Fault prediction in an early stage is important because it can reduce the cost of maintenance of the software product and reduce any fatal error that can cause the software product to be disabled [19]. This research will examine software metrics linked to dependability that are relevant throughout the early phases of software development. Additionally, the impact of the fault prediction model with different feature selection methods and machine learning techniques will be examined thoroughly.

A. Software Fault Prediction

Predicting software failures is critical for software quality assurance [20]. Because current software development is so sophisticated, errors are unavoidable. Error-prone software projects will have unanticipated repercussions throughout the implementation phase, causing significant harm to enterprises and even jeopardizing the safety of people's lives [20]. More than 80% of software development and maintenance expenditures are currently spent on bug fixes [20]. Expenses may be significantly reduced if these errors can be caught early in the software development cycle. As a result, several research has attempted to construct predictive models for defect prediction to assist developers in detecting potential faults in advance.

Unfortunately, current error prediction models continue to have issues [20], such as insufficient classifier performance. To address these issues, numerous machine learning approaches such as Naive Bayes (NB), Logistic Regression (LR), Random Forest (RF), and Support Vector Machine have been suggested to forecast software failures [20]. (SVM). These models, however, are far from acceptable.

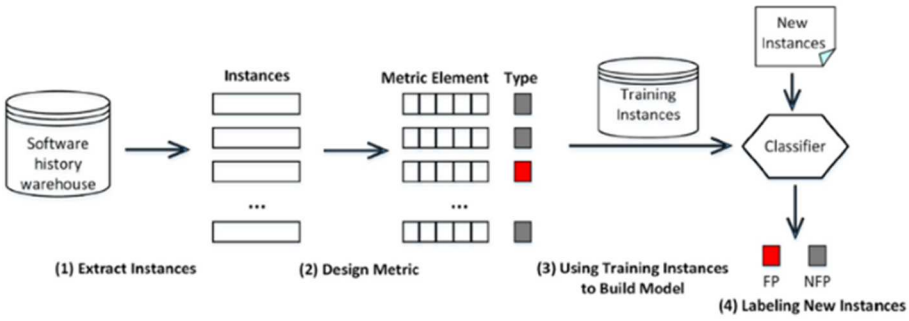

Fig. 1 The Process of Software Fault Prediction [20]

Predicting software failures is a significant area of study in software engineering [20]. The method of software failure prediction is shown in Fig.1, which consists of four phases. The initial stage is extracting the program modules/files/classes via exploration of historical software repositories and then categorizing the modules/files/program classes as error-prone or not. The second stage is extracting software problems' characteristics via an analysis of the program's source code or development process. Then, certain features, such as the Halstead trait [20], the McCabe trait, and the CK trait [9], were utilized to assess illness propensity. The grey colour represents the non-faulty bias module (NFP) in Fig.1, while the red represents the defective bias module (FP).

Training cases can create an error prediction model with relevant features [20]. Several machine learning algorithms are employed in this stage, including Naive Bayes, Support Vector Machine [21], Random Forest [23], and Logistic Regression [24]. The Naives Bayes algorithm has been developed to forecast the quantity of residual faults that may be discovered during independent inspection or operational usage. The suggested support vector machine optimizes the SVM model's prediction performance. Centroid approach.

These techniques use an error prediction model to forecast and categorize untagged software modules/files/classes.

B. Software Fault Prediction

Several evaluation metrics for machine learning classifiers include precision, recall, f-measurement, and Area under the receiver operating characteristic curve. AUC [20]. Among these measures, AUC is often used in software engineering research [20]. AUC refers to the region beneath the receiver operating characteristic. The x coordinate represents the false positive scale, whereas the y coordinate represents the genuine positive (recall) scale. The Receiver Operator Characteristic (ROC) curve is an assessment measure for binary classification issues. It is a probability curve that shows the TPR vs FPR at various threshold levels and, in essence, separates the "signal" from the "noise." As a summary of the ROC curve, the Area Under the Curve (AUC) shown in Fig. 2 quantifies the ability of a classifier to discriminate between classes. The greater the AUC, the greater the model's ability to differentiate between positive and negative classifications. A higher X-axis value in a ROC curve suggests a greater frequency of False positives than True negatives. A greater Y-axis value implies a greater proportion of True positives than False negatives. Therefore, the threshold selection is contingent on the capacity to strike a balance between False positives and False negatives. The AUC has been chosen as the performance statistic for three reasons. Unlike other indices, which need a value greater than the predicted likelihood of default susceptibility, the AUC is a threshold-independent measure [20]. The word 'threshold' refers to the probability threshold used to classify a situation as positive or negative. Other performance indicators (such as accuracy, recall, precision, and f-index) are calculated relative to the threshold specified and are commonly set to 0.5.

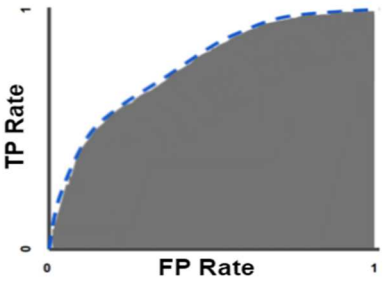

Fig. 2 Area Under Curve (AUC)

C. Machine Learning and Software Fault Prediction

With the fast advancement of software and computer technology, software has become ubiquitous in practically every facet of social life [25]. Due to the widespread use of invisible software that simplifies and expedites our lives. Meanwhile, technology is confronting a significant dilemma as the expense of software testing and maintenance and multifunctional and highly sophisticated software systems become more crucial. Software failure occurs when software is unable to execute its intended purpose owing to its own defect.

However, failures may be anticipated using software models that use past data from the organization and software features to predict problems [26]. Software failure prediction models are advantageous for decision-making in module verification and validation, such as resource allocation. We need to locate the fault and verify and repair the programme using more precise approaches. Thus, anybody may create a state where software is highly reliable, maintainable, and performant, with a short development cycle and cheap development and maintenance expenses [24].

The application of software failure prediction methods is extremely advantageous to the software development process because it results in a highly reliable software system with fewer error-prone modules, resource allocation based on prediction results, and maximizing the use of limited resources to enhance the quality of software products [27]. In recent years, several methodologies and models for forecasting software failures have been published. Examples of these techniques include statistical approaches, machine learning techniques, parametric modelling, and mixed algorithms [28]. The most prevalent software fault prediction models are Markov models, classification and regression tree models, artificial neural network models, linear discriminant analysis models, LSTSA, and classification tree models. However, these techniques pose problems that cannot be adequately overcome. For example, the Markov model must assume the classification tree and regression models have limited generalization capacity; and the artificial neural network model does not yet have a coherent guiding theory for network structure selection. Logistic regression (LR) further strengthens the difficulties of resolving a potentially incorrect value outside of the range of 0 and 1. In this case, LR performed a logarithmic modification to unconstraint the value. restricted to the range 0 to 1, which may be any value between positive and negative infinity. LR developed a quadratic discriminant that increased computing speed and recognition rate, resulting in its extensive application in modelling software defect prediction. Although it is often employed in big samples, the results are less than optimal for tiny samples [24].

D. Deep Forest

Recently, a deep forest model termed gcForest [24] was introduced as an alternative to deep neural networks. Similar to deep neural networks, gcForest features a multi-layered structure with many forests on each layer. Fig. 3 depicts the basic structure of a forest. The input is a feature vector X, and the output is a rank vector for X based on the forest's decision tree. The red line in Fig. 3 depicts the decision-making process of the decision tree. The functioning of deep neural networks prompted the design of gcForest, a complete component. The first is multi-grain analysis, which uses a sliding window structure to evaluate the immediate environment from a one-dimensional perspective in order to develop random forest representations of input data. The second is a cascading multi-layer population of random forests that, under the supervision of an input representation supervisor for each class, learns more discriminating representations, resulting in more accurate predictions. Constructed upon a set of randomly generated forests. Unlike conventional deep neural networks, which need a substantial amount of computer resources to train, deep forests use far fewer resources. Moreover, Deep Forest can perform several tasks with its default settings [24].

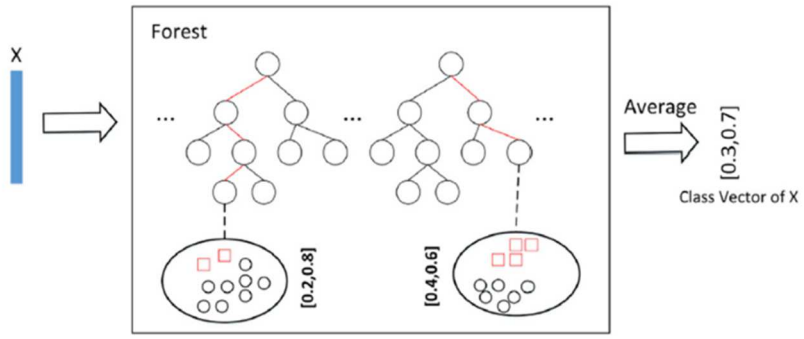

Fig. 3 The Basic Structure of Deep Forest [24]

E. Support Vector Machine Learning (SVM)

When applied to binary classification, SVM's central notion is to find the hyperplane in a high-dimensional space that serves as the plane of separation for the two aspects, ensuring the lowest possible error rate. SVM turns the classification issue to a constraint-based quadratic programming problem [29]. If the sample sets are linear and separable, it is assumed that (x1; y1) Rn; y+1; 1 indicates the number of samples and the category, such as a hyperplane separating these two sample kinds fully. This strategy is denoted by equation (1) below:

$$wix + b = 0 \tag{1}$$

The symbol denotes the intersection of two vectors. The parameters w and b denote the hyperplane's normal and deviation vectors, respectively. The above issue may be stated in the following broad terms:

$$f(x) = wix + b \tag{2}$$

If f(x) is greater than 0, the category label is +1; otherwise, it is -1. We hope that the training data will be accurately dissected during vector classification and that the gap between the nearest and hyperplane data will be as great as feasible. This is a quadratic programming issue, similar to equation (3).

$$\min(\frac{1}{2} \parallel w \parallel^2) \tag{3}$$

The optimum function has a quadratic shape, and the constraints suggest that the issue described above is a classic quadratic programming problem. It is possible to resolve it by first introducing the Lagrange operator and then converting it to its double version.

$$min\frac{1}{2}\sum_{i=1}^{L}\sum_{j=1}^{L} y_i y_j (x_i x_j) a_i a_j - \sum_{i=1}^{L} a_j \tag{4}$$

F. K-Means Algorithm

In 1967, James MacQueen used the k-means time period for the first time [4]. The k-means algorithm may be thought of as a reduced basic set of rules for resolving well-known clustering situations. The set of rules is implemented via the usage of exact records. Clustering operations are carried out using predefined clustering centroids. Centroids must be chosen as far apart as possible in order to establish a legal clustering. On the other hand, each factor associated with a record set is connected with the nearest centroid. The preceding stages are repeated until stable centroids are obtained. The k-means set of rules consists of the following: Select the okay records factor as the preliminary centroid and then assign all factors to the centroid that is closest to it. Assign centroids in accordance with the newly formed clusters. Rep the second and third steps until the centroids are steady.

Due to its simplicity, the k-means set of rules is quite favourable [4]. The recommend clustering randomly produced centroids and factors. Hexagons are used to symbolise centroids. The computed bounds of the clusters are displayed using a set of k-means rules applied to factors. Establishes the first centroids. Arrows depict the gadget's dynamic form, in which centroids shift in response to external variables, causing boundaries to shift. Later, new centroids and cluster boundaries are established.

G. Process Metric

The selection of software metrics for use in constructing prediction models of software quality is a software engineering activity based on search [20]. Due to limited project resources, a thorough search for such metrics is sometimes impractical, especially when the number of available metrics is vast. Abreu MOOD metric suite, Bieman and Kang, Briand, Halstead, Henderson-sellers, Li and Henry, McCabe, Lorenz and Kidd, and CK metric suite are the most often employed metric suites that indicate the quality of the programme. PROMISE repository source code metrics were evaluated for developing a defect class prediction model in this work.

This study focuses on applying software metrics, such as code metrics, process metrics, and a mix of the two. Using a hybrid metric dataset, the software metrics were categorised by performing feature selection during the data pre-processing step. Process metric may be defined as a change in software or code [10] whereas Code metric or product metric can be defined as an indicative measurement of errors that gives a quantitative description of specific aspects of software products and process [12]. As a result of the accessibility of code metric-based datasets, code metrics were predominantly employed for fault prediction research [11].

H. Code or Product Metric

Code metrics were mostly used for fault prediction studies because of easy availability of code metrics-based datasets. Most common datasets used in related studies belong to PROMISE repository which can be accessed publicly worldwide. Following are some papers that experimented

using code metrics and are related to our study. Following are some papers using code metric. Li and Henry examined CK metric suite for fault prediction for the first time. Except for LCOM, all measures properly predicted fault proneness, it was discovered. However, the correlation between the evaluated measures and error propensity was not studied.

Ohlsson assessed the fault prediction capabilities of several design metrics. All used measures have a substantial link with fault proneness. Nonetheless, this review was conducted on a single software project. Insufficient evaluation of fault prediction models. Shanthi and Duraisamy assessed the set of MOOD measures for fault prediction. We discovered a substantial link between each parameter and mistake propensity. However, it lacked a thorough review of fault prediction models and did not analyse the connection between the metrics utilised for fault prediction. Shatnawi and Li analysed the severity of software error prediction metrics. It was determined that CTA, CTM, and NOA measures accurately predicted class-error probability across all mistake severity categories. The gathering of defect datasets was performed using commercial software, resulting in questionable precision.

Code metrics were mostly used for fault prediction studies because of easy availability of code metrics-based datasets. Most common datasets used in related studies belong to PROMISE repository which can be accessed publicly worldwide. Following are some papers that experimented using code metrics and are related to our study. Following are some papers using code metric. Li and Henry]examined CK metric suite for fault prediction for the first time. Except for LCOM, all measures properly predicted fault proneness, it was discovered. However, the correlation between the evaluated measures and error propensity was not studied.

Ohlsson assessed the fault prediction capabilities of several design metrics. All used measures have a substantial link with fault proneness. Nonetheless, this review was conducted on a single software project. Insufficient evaluation of fault prediction models. Shanthi and Duraisamy assessed the set of MOOD measures for fault prediction. We discovered a substantial link between each parameter and mistake propensity. However, it lacked a thorough review of fault prediction models and did not analyse the connection between the metrics utilised for fault prediction. Shatnawi and Li analysed the severity of software error prediction metrics. It was determined that CTA, CTM, and NOA measures accurately predicted class-error probability across all mistake severity categories. The gathering of defect datasets was performed using commercial software, resulting in questionable precision.

I. Object Oriented Metric

One cost-effective approach to resolving software faults is to use fault prediction models relying on object-oriented (OO) design metrics to identify software defects in created systems' classes prior to delivery. Such models may be used to assist software development managers in generating quality software on time and within budget , and certainly, during the past three decades, several software fault prediction models have been suggested. However, software errors vary significantly in severity, including some potentially devastating effects and others being just cosmetic. Most of it were related to find the effectiveness and analysing the defect prediction model using object oriented metric source code only. Most of the source code metric were taken from PROMISE repositories which is widely used in research studies.

Previous research on object-oriented design metrics has demonstrated that some of them are effective in predicting the fault-proneness of classes but has also demonstrated that their usefulness is contingent on their being valid regardless of the severity of the fault. However, these investigations did not differentiate between problems based on their severity.

TABLE I
REVIEW ON IMPLEMENTATION OF PROCESS METRIC AND CODE METRIC

Process Metric	Code Metric
Code Delta: Delta of LOC, Delta of Changes Code Churn: Total LOC Churned, LOC Deleted LOC, File Count, weeks churn, churn cost and files churn etc. Change Metric: Revisions, Refactoring, Bug fixes , ashes, etc. Developer Based: Personal commit Sequence, No. of corrections etc Requirement of Metric: Action, Conditional, Continuance, Imperative etc.	CK Metric Suits: CBO, LCOM, DIT, NOC, RFC and WMC Yacoub Metric Suit: Export Object Coupling (EOC), Import Object Coupling (IOC) Airdelm Metric suit: IC_OD, IC_OM, IC_OC, IC_CD, IC_CM, IC_CC,EC_OD,EC_OM, EC_OC, EC_CD, EC_CM, EC_CC, Michel Metric Suits: Dynamic CBO for a class, Degree of Dynamic Coupling between two classes at runtime, Degree of Dynamic Coupling within a given set of classes, R1,R2,RD1,RD2 Moods metric suit: MHF,AHF, MIF,AF,PF,CF

Table I show the explanation of implementation of process metric and code metric. Code metric utilise various capabilities of the finalised software, product metrics are calculated. These metrics verify compliance with certain standardisation requirements, such as ISO-9126. There are three broad categories of product metric: traditional metric, object-oriented metric, and dynamic metric. Traditional metrics consist of software metrics that were developed during the discipline of software engineering's infancy and gradual emergence. Chidamber and Kemerer [9] presented the CK metric package as a software metrics suite for OO applications. Dynamic metrics are metrics that depend on the characteristics acquired from a running application. Their significance stems from the fact that they are real-time and based on how the gathered software components react at the time of actual programme execution. They evaluate particular runtime properties of programmes, system components, and

system. Process metrics are derived from the features gathered across the whole software development life cycle. They aid in the formulation of improved strategies for future software development processes. They are particularly helpful in facilitating the standardisation of a collection of process metrics, which leads to the long-term enhancement of the software development process.

J. The Combination of Metric

Numerous academics have utilised various metrics to analyse and enhance fault prediction. Some academics have also conducted a comparison of the various measures employed for software defect prediction. In order to evaluate the efficacy of a defect prediction model, several studies incorporated process and code metrics. Following is a summary and analysis of the research publications that addressed the integration of process and code metrics.

Rahman and Devanbu evaluated the combined capacity of process and code metrics for failure prediction and found that the process measure nearly always exceeded the code metre. Due to the usage of commercial tools to compute code, the correctness cannot be determined with certainty. The results are not the product of extensive analysis.

Ma employed requirement metrics for the first time to assess failure prediction. When the requirement metric was paired with the design metre, the fault prediction skills improved significantly, according to the findings. Considers software metrics evaluation studies utilised a restricted number of datasets, such as two or three. No study of data distribution for statistical testing was performed.

Wu analyse the impact of developer quality on software failure prediction. The integration of developer, process, and product data improves the outcomes of fault prediction. To validate the process, more datasets from various areas must be utilised for evaluation.

Selection strategy for useful software metrics for fault prediction was proposed by Xia. CM, MMSLCM, and HM metrics demonstrated a considerable effect on fault prediction, relative to the amount of code and process metrics evaluated. With only a number of fault datasets utilised to validate the results, the suggested selection method is insufficiently validated to show its applicability.

As a conclusion, the previous works by previous researchers related to the combination metric were discussed above. Rahman and Devanbu stated that process metric has better accuracy compared to the code metric although the accuracy is not well ascertained. There are several potential factors that affect the accuracy of fault prediction such as the type of machine learning (supervised or unsupervised learning), data pre-processing, number of features and feature selection technique. Although previous researchers were not highlighted the object-oriented metric as their main purpose, it can be concluding the usage of object-oriented metric in software fault prediction are capable to predict the defect at the class level of the system.

K. Feature Selection

For feature selection process, there were many techniques used in previous research such as Filtering Features selection methods proposed by . This technique had better performance prediction compared to the old previous studies. The best feature selection method will be selected in data pre-processing process to select the best metric for training the machine learning model.

II. MATERIAL AND METHOD

A. Proposed Experiment Procedure

This study's experimental methodology will consist of four phases and will be conducted using a Python-based experiment workbench. The first phase is data preparation, which involves applying a variety of data pre-processing procedures to the study dataset. Test/train segmentation, data cleaning, feature engineering, and feature selection are the data preparation procedures. In the second stage, several Random Forest models are trained on the pre-processed dataset. Random Forest Classifiers have demonstrated significant efficacy in predicting software problems in the past and are thus great candidates for comparison analysis utilising a number of software metrics [10]. Model evaluation is the third phase, which involves determining the accuracy and f1 score values for each model. The fourth and last phase, Result Analysis, will evaluate the relative importance of each character in the pre-processed dataset for software failure prediction by analysing the results. In this research, each machine learning model will undergo a sensitivity analysis. The subsequent sections will examine the execution of the four steps of the experimental technique on the experimental workbench.

B. Dataset

This research used a single dataset including a count of faults/bugs in our research. The datasets are open to the public, and the derived class is fault count. These datasets are part of the PROMISE open-source initiatives. These datasets are referred to as CM1/software defect prediction by NASA. Process and code metrics are included as features in the datasets. Each dataset has fifteen process metrics and seventeen code metrics. In the features section, all of these measures are explained in great detail. These datasets are class-level datasets, with the target class having a specific number of faults/bugs. The total number of dataset instances is 484.

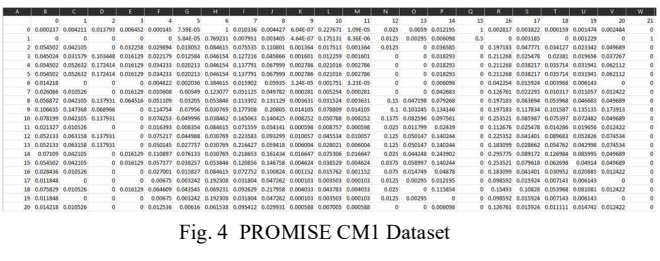

Fig. 4 PROMISE CM1 Dataset

C. Experiment Setup

This study uses a two-stage design. The external "View layer" presents the experimental design's input, processing, and output, as well as the method for loading the investigated datasets into the experimentation, in a black-box manner. The "Application layer" of the experimental workbench analyses the research dataset to calculate each model's accuracy and F1 scores. Python workbench obtains research dataset from "View Layer." The workbench hosts experiments. The experiment workbench in the "Application Layer" prepares

the dataset before using it. The dataset is split 70-30 between testing and training. The workbench removes unnecessary characteristics from the training dataset. Feature engineering—data binning, missing value imputation, categorical value encoding, and numeric data standardization—follows on the training set. "Feature engineering" is this.

The workbench then allows data set selection. Wrappers subset selection chooses the data set's procedure and code metrics' greatest attributes. Our hybrid measurements predicted defect numbers. All metrics subgroups were studied. The workbench pre-processes the testing set after pre-processing the training set. Workbench-cleaned training sets are used to train Random Forest Classification models. The workbench evaluates models after training using process and code metrics. The workbench categorises metrics and evaluates models for each category and single metrics from process and code subsets.

The experiment workbench begins by creating a Python virtual environment and installing the essential Python libraries and packages. Anaconda does this. Free and open-source Anaconda lets you construct and manage Python virtual environments. This utility can install different Python packages and their dependencies in various Python virtual environments. This inquiry uses Python 3.8.3 and Anaconda. With Anaconda and the terminal, a python virtual environment named "software fault_ prediction" is created. The "software fault prediction" virtual environment instals the "numpy," "pandas," "seaborn," "matplotlib," and "scikit-learn" python libraries and their dependencies.

Installed libraries have different capabilities. Only Python's "numpy" library handles arrays. Pandas is essential for data analysis. Data visualisation requires Matplotlib. The "Seaborn" package, built on "Matplotlib," helps visualise and comprehend data. The machine learning pipeline requires Scikit-learn for data preparation, ML model training, and performance assessment. Scikit-learn (previously scikits.learn and sklearn) is a free Python machine learning package. It supports support-vector machines, random forests, gradient boosting, k-means, DBSCAN, and NumPy and SciPy numerical and scientific libraries. S. Hassan [10] describe scikit-machine learn's learning toolset.

The experiment workbench is programmed in Jupyter Notebook. Jupyter notebook, an open-source online programme, supports numerical simulation, statistical modelling, data visualisation, and machine learning. Data scientists prefer it. Jupyter Notebook can link to multiple kernels for multi-language programming. Jupyter kernels respond to code execution, completion, and inspection requests. Kernels communicate with other Jupyter components using ZeroMQ. Jupyter kernels may connect to several clients and are unaware that they are associated to a document, unlike many Notebook-like interfaces. Kernels usually support one language.

D. Data Pre-processing

Data pre-processing is necessary to turn the NASA PROMISE repository's PROMISE dataset into a machine learning (ML)-ready format and to maximise the dataset's productive potential. Python is used for data pre-processing, and the different data pre-processing processes, including data cleaning, feature engineering, and feature selection, are programmed into the experiment workbench to automate them. The first PROMISE dataset had 2998 occurrences and 22 date-time, ordinal, and numeric properties. After pre-processing, 11 characteristics and 484 cases remain in the dataset. Table II provides a summary of the final pre-processed dataset.

E. Test/Train Split

A random 70-30 split is used to divide the dataset into a training set and a testing set before any preparatory processes are carried out. This is done to prevent the model from being overfit by evaluating it with the same data it was trained on. It will not be an accurate depiction of the actual situation, as the data will be utterly novel and unheard of in the real world. Consequently, it is likely that the generated model will perform badly in real-world circumstances.

TABLE II
SUMMARY OF THE FINAL PRE-PROCESSED DATASET

No.	Feature Selected
1	Line of Count code
2	Cyclomatic complexity
3	Essential complexity
4	Design complexity
5	Total operators and operand
6	Volume
7	intelligence
8	effort
9	Count of line of comments
10	Unique operators
11	Total Operands

The training set is pre-processed first, followed by the application of the pipeline used to pre-process the training set to the testing set. Separately preparing the testing and training sets guarantees that no data is lost, i.e., the pre-processing in the training set is not impacted by information from outside the training set, in this case the testing set. This prevents any bias by keeping the data in the testing set fresh and unknown during model assessment.

F. Data Cleaning

In this phase of preparation, the PROMISE NASA dataset is cleaned with the use of "pandas" library functions. Pandas has been one of the most popular and favoured data science tools for data manipulation and analysis in the Python programming language. In addition to Pandas, numpy is frequently utilised by data scientists for scientific computing in Python. Pandas is highly effective with tiny datasets (typically 100MB to 1GB) and speed is rarely an issue. The Pandas DataFrame is a structure containing two-dimensional data and its associated labels. Pandas DataFrames are utilised extensively in data science, machine learning, and other data-intensive domains. Initially, unnecessary characteristics are removed from the original dataset. For instance, the original dataset included information on the job title, date of employment, and date of separation; however, this information is unrelated to the current study and has been removed. Second, certain recurring characteristics are eliminated. The dataset has separate columns of test results for each of the domain-specific topics, despite the inclusion

of a "Domain" column indicating each candidate's domain-specific subject test results.

G. Feature Engineering

This phase involves altering the dataset so that it is more suitable for machine learning. Initially, missing values in columns are determined. Second, the "Defect" column, which denoted the module that reported the defect, was modified using data binning to represent two classification values to conduct a first inquiry for software fault prediction using the PROMISE NASA dataset. The values within the Defect column are converted into a new column titled "Defect" that has two categories: "1" and "0."

Since ML algorithms in Scikit-learn cannot work on categorical values, the fourth step is to convert all category values in the dataset to numeric values. According to Scikit-learn documentation, the "OneHotEncoder" function from Scikit-learn preprocessing is used to convert the categorical values for the independent variable "Defect" to numerical values, while the "LabelEncoder" function does the same operation for the dependent variable (predicted class). The "LabelEncoder" assigns number labels to each of the column's category values. This meets the criteria for the intended class. ML algorithms view a higher number as more significant or significant than a smaller number.

Several numerically based features have been standardised. It is necessary to rescale the data so that the mean is 0 and the standard deviation is 1. This is owing to the broad range of values for a variety of numerical attributes. As an illustration of this, the "Line of Count" column, which shows line counts, has figures ranging from 0.008155 to 0.99991. This is a concern since some machine learning algorithms that employ gradient descent as an optimization strategy are sensitive to feature range differences because each feature has a specific step size and features with bigger magnitudes may be given more weight. Since this study will employ AutoML, which applies many machine learning techniques to a dataset, the numeric features must be normalised. MinMaxScaler, a Scikit-learn preprocessing function, is used to normalise the values in the numeric columns.

H. Feature Selection

This phase will summarise the investigated qualities. In this study, two distinct types of software metrics were employed: code metrics and process measurements. Each of these domains is evaluated using a variety of criteria. Code metrics are a type of software statistic that are calculated based on the code structure of a programme. They offer information about the code's characteristics, size, and structure. In addition, process metrics based on software revisions, upgrades, and tweaks were evaluated.

I. Model Training

The subsequent part of the experiment is to train machine learning models. The workbench is designed to train models for several types of measures, such as selected metrics and hybrid metrics, using a single ML method. Various machine learning methods were utilised to forecast software bugs. These models comprised decision trees, naive bayes, logistic regression, as well as several more. Due to Hassan's[10] experiment on predicting software failure using many software metrics and Random Forest regression, Random forest classifiers were used in this study. Random forest is a technique for machine learning that was initially described in 1995 and updated in 2001.

It may be used for both classification and regression learning. The structure-wise random forest methodology is an ensemble method that use many decision trees to represent distinct classes. The result of categorization learning is the mode of all trees, whereas the mean of all trees is calculated. Random Forest Classifier is a Scikit-learn tool used to train models using Decision trees. Utilizing a Randomized Search approach, the ideal hyperparameter values for the Decision Tree are determined. This is accomplished by providing a list of possible Decision Tree hyperparameter values to the Randomized Search algorithm, which then executes several times to determine the optimal hyperparameter values for the current training set. In regard to the conclusion of the Randomized search, the maximum depth of the decision tree is set to 100. The "criterion" hyperparameter is set to "entropy," which sets the measurement method used to evaluate the quality of the decision tree split. The "class weight" hyperparameter is afterwards changed to "balanced" to address the dataset's imbalance. Lastly, the Decision Tree with the best hyperparameter value is trained using the training data using the "fit()" function of scikit-learn.

J. Model Testing/Evaluation

The "metrics" tools included in the Scikit-learn package facilitate model testing during the experiment. Each model is tested using a sklearn-imported "metric" library. The import library from sklearn uses the "metrics" methods "metrics.accuracy score()" and "metrics.f1 score()" to automatically generate prediction accuracy and f1 score values. This technique requires two arguments: a training model and testing sets. The trained models, random forest classifiers, are compared to the testing set in order to determine the expected outcomes. Performance evaluation The metrics TP, TN, FP, and FN are used to evaluate specificity, sensitivity, and accuracy. True Positive, or TP, is the number of event values that were precisely anticipated. True Negatives, or TN, is the number of non-event values that were correctly anticipated. False Positive or FP is the number of event values that were mistakenly anticipated. False Negative, or FN, refers to the number of erroneously anticipated non-event values. The number of TF (true positive), FP (false positive), TN (true negative), and FN (false negative) instances for each model is calculated by comparing predicted and actual outcomes from the testing set. Finally, the accuracy and F1 score value of each model's prediction value are computed using the TP, TN, FP, and FN values.

K. Result Analysis and Discussion

In addition to the experiment conducted in the initial study, this investigation provides data and analysis to determine which input features have the most impact on forecasting software inaccuracy. In the experiment, the pre-processed dataset is altered such that it seems to lack a certain property. This is performed individually for each feature in the pre-processed dataset by converting the feature that makes all the values in each row constant, while keeping the other features

unchanged. When evaluating a numerical feature, the mean value of the feature is substituted for each number in each row, while all other qualities remain untouched. When a categorical characteristic is present, the category with the highest occurrence frequency is chosen to replace all rows. To analysing the outcome of performance software fault prediction, object-oriented metric features were compared with hybrid metric features.

When each feature is analysed, a new dataset including the modified feature and other intact features is generated. The experiment employs sensitivity datasets for each feature in order to train and assess the Random Forest algorithms, and the f1 scores for each scenario are recorded. Then, these values would be compared with the f1 scores generated during model evaluation of the two distinct software metrics. The outcome of each characteristic will be indicated by the size of the difference.

III. Result

When each feature is evaluated, a new dataset is constructed that contains the updated feature and other intact features. The experiment uses sensitivity datasets for each feature to train and evaluate the Random Forest algorithms, and the f1 scores values for each scenario are recorded. These values would then be compared to the f1 scores obtained during model assessment of the two separate software metrics. The result of each attribute will be represented in the magnitude of the difference.

A. Results Analysis and Discussion

The initial test result for this experiment is in Table III. Nevertheless, for the experiment at process metric shown that it was slightly higher than code metric.

TABLE III
THE RESULT OBTAIN FROM THE INITIAL EXPERIMENT

Iteration	Process Metric		Code Metric	
	Accuracy (%)	F1 Score	Accuracy	F1 Score
1	89.33	0.1866	88	0.104

The accuracy and F1 score of the process metric are superior to those of the code metric, as shown in the table III above. The procedure of feature selection influences the accuracy and f1 score ratings. In this preliminary work, we employ only one machine learning technique, the random forest classifier, because it outperforms other classifiers [10]. As a means of enhancing the accuracy of the prediction model, we may employ a bigger data set to improve the accuracy of the findings.

To differentiate between the types of metrics, the process and code metric were selected based on feature selection method and were train and test in the deep learning model. The result can be shown in Table III using the accuracy and F1 score formula to evaluate the prediction model.

$$F1 = 2 \times \frac{precission * Recall}{precission + Recall} \quad (5)$$

$$Accuracy = \frac{True\ Positive + True\ Negative}{\text{True Positive+True Negative+False Negative+False Positive}} \quad (6)$$

IV. Conclusion

The experimental data of software fault prediction is presented here. Further works that can be proposed based on the work and results presented in this study are Include additional large (higher number of instances) and varied (connected to other software) datasets into the testing and experimenting process in the future and compare the performance of the new neural network algorithm to our findings. Furthermore, we will explore the impact of class balancing on the prediction of software defects in the future. Lastly, we assess the influence of various issue severity classes and the impact of certain issue severity classes in fault prediction.

Acknowledgements

The authors humbly acknowledge the UTM Encouragement grant, Q.J130000.3851.19J69, funded by Universiti Teknologi Malaysia, Malaysia.

References

[1] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," Artif Intell Rev, vol. 51, no. 2, pp. 255–327, Feb. 2019, doi: 10.1007/s10462-017-9563-5.

[2] S. S. Rathore and S. Kumar, "Towards an ensemble-based system for predicting the number of software faults," Expert Syst Appl, vol. 82, pp. 357–382, Oct. 2017, doi: 10.1016/j.eswa.2017.04.014.

[3] N. Fenton, M. Neil, W. Marsh, P. Hearty, Ł. Radliński, and P. Krause, "On the effectiveness of early life cycle defect prediction with Bayesian nets," Empir Softw Eng, vol. 13, no. 5, pp. 499–537, Oct. 2008, doi: 10.1007/s10664-008-9072-x.

[4] M. Öztürk, U. Cavusoglu, and A. Zengin, "A novel defect prediction method for web pages using k-means++," Expert Syst Appl, vol. 42, no. 19, pp. 6496–6506, May 2015, doi: 10.1016/j.eswa.2015.03.013.Kumar Pandey and M. Gupta, "Software Metrics Selection for Fault Prediction: A Review."

[5] Okutan and O. T. Yıldız, "Software defect prediction using Bayesian networks," Empir Softw Eng, vol. 19, no. 1, pp. 154–181, 2014, doi: 10.1007/s10664-012-9218-8.

[6] Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Systems with Applications, vol. 36, no. 4. pp. 7346–7354, May 2009. doi: 10.1016/j.eswa.2008.10.027.

[7] Alshayeb and W. Li, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes," IEEE Transactions on Software Engineering, vol. 29, no. 11, pp. 1043–1049, 2003, doi: 10.1109/TSE.2003.1245305.

[8] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, 1994, doi: 10.1109/32.295895.

[9] R. Shatnawi, "The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction," Innov Syst Softw Eng, vol. 13, no. 2–3, pp. 201–217, Sep. 2017, doi: 10.1007/s11334-017-0295-0.

[10] S. Hassan, "Predicting the Number of Software Faults Using Significant Process and Code Metrics," 2020.

[11] P. Musilek, A. Mahaweerawat, P. Sophatsathit, and C. Lursinsap, "Fault Prediction in Object-Oriented Software Using Neural Network Techniques." [Online]. Available: https://www.researchgate.net/publication/228824925

[12] Catal and B. Diri, "LNCS 4589 - Software Fault Prediction with Object-Oriented Metrics Based Artificial Immune Recognition System."

[13] T. M. Khoshgoftaar and K. Gao, "Feature selection with imbalanced data for software defect prediction," in 8th International Conference on Machine Learning and Applications, ICMLA 2009, 2009, pp. 235–240. doi: 10.1109/ICMLA.2009.18.

[14] Van Hulse, T. M. Khoshgoftaar, and A. Napolitano, "Experimental Perspectives on Learning from Imbalanced Data."

[15] Thammasiri, D. Delen, P. Meesad, and N. Kasap, "A critical assessment of imbalanced class distribution problem: The case of

predicting freshmen student attrition," Expert Syst Appl, vol. 41, no. 2, pp. 321–330, 2014, doi: 10.1016/j.eswa.2013.07.046.
[16] Y. Yang et al., "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models," Nov. 2016, pp. 157–168. doi: 10.1145/2950290.2950353.
[17] W. Li, W. Zhang, X. Jia, and Z. Huang, "Effort-Aware semi-Supervised just-in-Time defect prediction," Inf Softw Technol, vol. 126, Oct. 2020, doi: 10.1016/j.infsof.2020.106364.
[18] Catal, "Software fault prediction: A literature review and current trends," Expert Systems with Applications, vol. 38, no. 4. pp. 4626–4636, Apr. 2011. doi: 10.1016/j.eswa.2010.10.024.
[19] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," Inf Softw Technol, vol. 114, pp. 204–216, Oct. 2019, doi: 10.1016/j.infsof.2019.07.003.
[20] Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," Softw Pract Exp, vol. 41, no. 5, pp. 579–606, Apr. 2011, doi: 10.1002/spe.1043.
[21] Kaur and R. Malhotra, "Application of random forest in predicting fault-prone classes," in Proceedings - 2008 International Conference on Advanced Computer Theory and Engineering, ICACTE 2008, 2008, pp. 37–43. doi: 10.1109/ICACTE.2008.204.
[22] Sharma and P. Chandra, "Identification of latent variables using, factor analysis and multiple linear regression for software fault prediction," International Journal of System Assurance Engineering and Management, vol. 10, no. 6, pp. 1453–1473, Dec. 2019, doi: 10.1007/s13198-019-00896-5.
[23] H. Xiao, M. Cao, and R. Peng, "Artificial neural network based software fault detection and correction prediction models considering testing effort," Applied Soft Computing Journal, vol. 94, Sep. 2020, doi: 10.1016/j.asoc.2020.106491.
[24] Kumar Pandey and M. Gupta, "Software Metrics Selection for Fault Prediction: A Review."
[25] L. Kumar, S. Misra, and S. K. Rath, "An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes," Comput Stand Interfaces, vol. 53, pp. 1–32, Aug. 2017, doi: 10.1016/j.csi.2017.02.003.
[26] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," IEEE Transactions on Software Engineering, vol. 32, no. 10, pp. 771–789, Oct. 2006, doi: 10.1109/TSE.2006.102.
[27] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," Inf Softw Technol, vol. 114, pp. 204–216, Oct. 2019, doi: 10.1016/j.infsof.2019.07.003.
[28] Isa and R. Rajkumar, "Pipeline defect prediction using support vector machines," Applied Artificial Intelligence, vol. 23, no. 8, pp. 758–771, 2009, doi: 10.1080/08839510903210589.
[29] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," Softw Pract Exp, vol. 41, no. 5, pp. 579–606, Apr. 2011, doi: 10.1002/spe.1043.