International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Machine Learning-Based Defect Prediction for Software Efficiency

[1]Jayanti Goyal [2]Dr. Ripu Ranjan Sinha

Submitted: 10/02/2023 **Revised:** 17/04/2023 **Accepted:** 08/05/2023

Abstract- Software engineering research is centered on defect prediction. Successful software development requires better communication between data mining and software engineering. Software defect prediction is a pre-testing technique that estimates where bugs will show up in the code. The purpose of software defect prediction research is to identify potentially flawed parts of a programme before it reaches the testing phase. The primary benefit of these prediction models is that they need more testing time and money. may be directed to the modules most prone to errors. However, only a few mobile app-specific software defect prediction algorithms currently exist. It is common practise to utilise defect prediction algorithms to probe the impact domain in software (clustering, neural networks, statistical methods, and machine learning models). This research aims to examine and compared various ML (machine learning) algorithms for software bug prediction. Despite the widespread availability of failure prediction methods, no one strategy is appropriate for every data collection. Support Vector Machine, Random Forest, Naive Bayes, Logistic Regression, and Artificial Neural Network, were only some of ML methods utilised to find biggest possible subset of faults. The goal of this study is to utilize 5 data sets (JM1, KC1, KC2, PC1, and DS1) to identify flaws. As compared to other methods, ANN has been demonstrated to have the highest accuracy (93.8%).

Keywords—Software Metrics, Software Defect Prediction, Software Failure Factors, Software Quality Assurance, Machine Learning

1. Introduction

Since modern software systems are more intricate, developing bug-free code is becoming increasingly challenging. Finding and fixing flaws in software design should be an ongoing process. As 100% fault-free software production seems unlikely, it's important to work on defect minimization instead [1]. It's not simple to do this. Saving money requires finding software flaws as early as possible in the development process. Additionally, early identification will guarantee that high-quality software is being given to clients at the lowest possible cost. Prediction models for identifying flawed classes in software may be developed using several Object Oriented (OO) software metrics that measure structural properties of a programme like inheritance, cohesion, etc. These measurements may be gathered from previously released software and then used to inform prediction models for future updates [2].

As a result, the quality of software systems may be enhanced by more precise predictions of whether or not software entities include design faults. A software metric is a quantitative or qualitative evaluation of some aspect of software. It's used to assess software quality when still in its formative stages of development, such design, and coding [3]. A software flaw is any aspect of a piece of software that fails to function as intended or satisfy the needs of the intended audience. In other terms, a defect is a flaw in the program's code or logic that leads to unintended behaviour. Predicting where bugs will appear in code is known as software defect prediction. High-quality software development results in a finished product with minimal bugs [4]. Finding software bugs early on potentially saves time and money during development and makes the final product more stable. As a result, predicting when a bug may appear is crucial for improving software quality. Defect prediction metrics are backbone of any statistical prediction model development process.

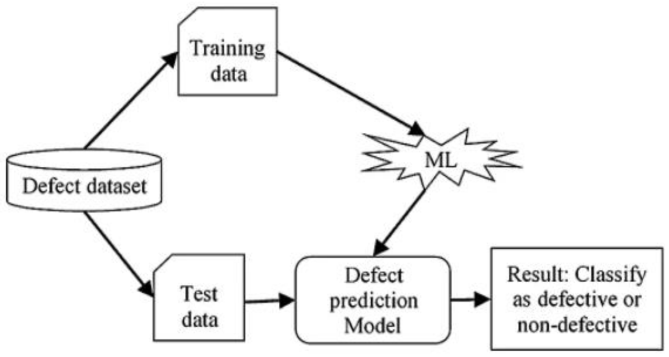

Fig. 1: Software defect prediction process

Hence, predicting defects is crucial for improving software quality. While developing a statistical prediction model, defect prediction metrics are crucial. There are two main types of metrics used for defect prediction: code metrics and process metrics. By developing these prediction models, software companies may utilise them throughout the earliest stages of development to locate problematic

1Computer Science Department
Rajasthan Technical University (RTU)
Kota
goyal.jayanti@gmail.com
2Computer Science Department
Rajasthan Technical University (RTU)
Kota
drsinhacs@gmail.com

code. Software companies may utilise these measures to narrow down the larger pool of software metrics they have at their disposal [5]. The measurements may be used to create models that forecast defects. Researchers have utilised a wide variety of approaches to determine the correlation between static code metrics and defect forecasting. LR (Logistic regression) is an example of a classic statistical technique, whereas SVM (Support Vector Machines), NB (Naive Bayes), ANN (Artificial Neural Networks), and Decision trees, are examples of ML techniques [6]. The objective of a neural network is to represent data in a manner which minimizes discrepancy between actual class labels of data tuples and predicted class of network. In order to discover a hyperplane for data separation utilising crucial training tuples, SVM projects the original data into a higher dimension. The process is carried out in a tree-like fashion via a decision tree. Each node in the tree that isn't a leaf is evaluated on a set of attributes using an attribute selection measure.

The outline for the rest of the paper appears below. Section II elaborates the concept and overview of software defects. Section III discusses the software metrics and the software quality assurance is described in section IV. Section V provides the detail of software failure factors. Section VI describes the available literature on approaches for software defect prediction. Section VII covers a variety of ML algorithms. The comparison of several ML methods is presented in Section VIII. The last thoughts on the subject are presented in Section IX.

2. Concept and Overview of Software Defects

A. Concept of Software Defects

Incorrect or unexpected results can be generated by software due to errors, bugs, flaws, faults, malfunctions, or other types of defects. Inherent to any system are flaws. They manifest as a result of the process of making something or the materials used. Software defects are coding mistakes that result in unexpected behaviour. Most bugs originate in the code or design, while others are the result of faulty compiler output.

Software bugs provide a serious security risk for both developers and users. Defects in software don't only slow down development; they also raise costs and lower quality. Predicting software faults is presented as a means to address such issues. SDP may help maximise software testing's efficiency and guide the use of scarce resources. Finding and fixing software bugs early in the software development life cycle (SDLC) is essential for producing high-quality code [7]The description of the concepts that are easily mistaken with defects is as follows:

- **Fault:** The programme operates in an improper internal state and fails to provide the expected results for the client. Considered to be a kind of dynamic behaviour, this flaw may be seen as a cause of software failures.
- **Failure:** It describes the results of the software's execution that the customer does not want to see. Example: if the client's capabilities aren't met and the framework loses its execution capacity, it won't be able to meet fixed asset's execution needs.
- **Error:** Humans are responsible for its introduction, and it may then be perverted into mistakes. Inconsistencies in software design, requirements analysis, data structure, code, and other carriers provide a trail of evidence that follows the project from start to finish [8].

The defect count is a key indicator of software quality. Client satisfaction drops, resources are depleted, and the testing process drags on when there are too many flaws. Improving test efficiency is crucial to controlling problems and saving money.

B. Main Research Direction of Software Defects

1) Software Defect Management

The primary goals of defect management are data gathering, statistical analysis, and practical defect recording. Many robotized defect management systems have been developed by engineers to boost management productivity. The two most popular programmes in use today are Bugzilla, an open-source bug-tracking platform supplied by Mozilla, and JIRA, which is distributed by Atlassian. None of these systems allows for a more thorough study or explicit categorization of faults beyond recording their transactions, characteristics, and statistical information. Important parts of defect management include analysing and categorising defects. Thus, additional investigation of the data collected in JIRA and Bugzilla is required for the analysis and arranging of deformities [9].

2) Software Defect Analysis

Defect analysis is a common tool used by software engineers and designers to evaluate the caliber of their code and the finished product. Defect analysis is a technique used to categorise errors and discover their root causes in software. Finding, locating, evaluating, and bettering test efficiency are all goals of software defect analysis. Defects analysis techniques may be broken down into three broad categories: qualitative, quantitative, and attribute [10]. Root Cause Analysis (RCA) and Software Fault Tree Analysis are two common qualitative methods of investigation (SFTA). Single-attribute analysis and multi-attribute analysis are two typical ways of categorising attributes.

3) Software Defect Classification

The causes of software flaws are unique and convoluted. Better aggregation and categorization of defects may boost analyzers' efficiency, make it easier for programmers to evaluate code quality, and reduce analysis overhead.

Classification also helps in recommending fixes and reusing test cases. It's able to understand defect distribution based on categorization and analysis findings, stops common software bugs in their tracks, drastically improves the SDLC, and ultimately boosts software quality. Software defect analysis relies heavily on the categorization of software flaws. Defect categorization is crucial since the results have a direct impact on the next step of defect investigation. Until date, software fault categorization has been divided into two camps: those that rely on humans and those that use computers.

a) Manual classification of software defects

Examiners must rely on their own experience and judgement to manually classify software faults into categories. To begin, the team established a benchmark for defect categorization. The problem is identified, and a defect category is matched based on previous experience. Nevertheless, this strategy's categorization cycle is labor-intensive and needs a large group of experts. Data analysis requires a lot of time and resources since humans have limited energy and memory compared to computers. This means that humans will have a slower categorization speed.

b) Automatic classification of software defects

More and more people are turning to computers to automatically detect flaws in an effort to save development expenses and boost development efficiency. Experts are looking for an easy way to categorise problems, and with the rise of ML &AI, this has become an active area of study in business world.

4) Software Defect Prediction (SDP)

Software engineering is a potential area for future advancement. In the context of software development projects, Project Defect Detection is a technique for accumulating data about software flaws (WPDP). While the damage data prediction model has been utilised in other projects before, the authors claim that their cross-project defect detection (CPDP) programme is the first of its kind. Some recent research indicates promise for CPDP [11]. To be consistent objects across index sets, however, CPDP Indicators must originate from the same block. As modern technology is so adaptable, horizontal CPDP project applications and indicators [12] may be better suited to the situation. Although most previous approaches are supplemental project defect detection, the serial forecasting model uses up-to-date information to determine failure trends in newly developed software modules within the same project (WPDP). However, studies have shown that other programmes require sufficient historical data in order to function, regardless of whether they support statistical data or not.

3. Software Metrics

A software meter quantifies an attribute or characteristic of the code or its requirements. The effectiveness of software in accomplishing a certain task is typically measured using software metrics. A software metric is a quantitative or qualitative evaluation of an aspect of a programme. Metrics pertaining to complexity, coupling, and cohesion (CCC) may be assessed and utilised to assess software quality throughout its development in stages like design and coding. The most crucial part of developing a statistical prediction model is the use of defect prediction metrics. There are two main types of metrics used for defect prediction: code metrics and process metrics. Size, McCabe, Hastead, CK, and OO metrics are only few examples of most often used code metrics, which have been demonstrated to be more popular than process metrics. [13]. Several code metrics are given below:

C. Cyclomatic Complexity

It evaluates the code's structural complexity. It is generated by tallying the possible branches in the program's logic. More tests are needed to obtain high code coverage in a programme with complicated control flow, and the software will be less maintainable as a result.

D. Halsteads Product Metrics

In order to quantify complexity, the late Maurice Halstead developed measures, which take into account not only number of operators and operands in a module but also its vocabulary size and running time.

E. Product Metrics

The lines of code (LOC) count in Product Metrics offers an approximation of total number of lines code. While the count is taken from code, it may not accurately represent entire amount of code lines. If number is too high, it might indicate that a single type or function is attempting to take on too much. Pre-implementation design metrics are calculated from a requirements or design document. Object-oriented metrics are useful for debugging and providing insight into where and how classes and objects might be simplified.

4. Software Quality Assurance

Software quality has always been a focal point for IT and software companies. The ability to accurately forecast software defects has gained a lot of attention in the last several years because of the direct influence it may have on programme quality. The cost, delivery time, and maintenance expenses associated with software all go up significantly when flawed modules are present. Not only must you be able to meet deadlines and perhaps accomplish them quickly, but you must also be capable to provide code of excellent quality or immensely greater quality.[14].

As a result, there is a great deal of investigation into how to further improve the product quality within the mandated early phases of complete SDLC improvements to product's general concept may be accomplished in a number of ways, including via enhanced testing procedures, comprehensively programmed testing, and early deformity prediction[15]. Therefore, it can be useful to improve software quality by predicting, utilizing a software module, whether a software entity contains defects. Therefore, quality is an important factor in determining whether or not the software is suitable for a customer's needs or process, and pre-processing techniques, including KNN, multilayer perceptron, and many others, are applied to data to retrieve information of defective data. In short, happy customers are essential to a successful project, thus we'll be doing early-stage research to identify any issues.

The practise of predicting software bugs has gained significant traction in recent years. Software quality is directly impacted by software defect prediction. There is a strong correlation between poor-quality software modules and software cost, software completion time, and software maintenance expenses[16]. When it comes to SQA, there are two primary methods: defect detection and defect prevention. Preventing defects requires taking proactive measures to forestall the occurrence of future errors.

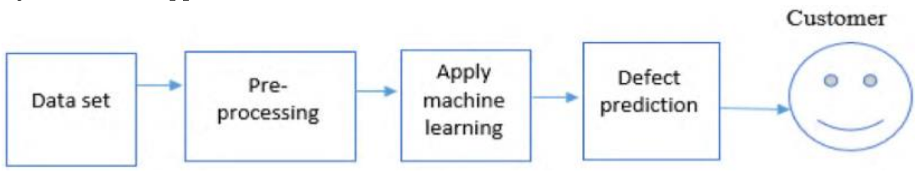

Fig. 2: Generic Process of SDP

Issues with existing faults are the focus of defect prediction. Defect prevention is a way to increasing software quality [13], and the focus of our study is on doing just that via the use of defect prediction. Software defect prevention encompasses elimination of errors in algorithm design, software requirement planning, and review [16]. The goal of defect prediction is to make informed decisions about the quality of a software product's delivery and amount of work required to keep it running smoothly before it is released to the public. Improving software quality is the method for preventing defects. The main goal of this study is to compare and contrast several ML methods in an effort to achieve the highest possible level of accuracy in feature selection for SDP. Finding and analysing which component of the programme is more prone to faults and delivering quality software is key to this study, as is the ability to foresee performance issues without going over the budgeted cost.

5. Major Factor of Software Failure

Finance, insurance, healthcare, manufacturing, e-commerce, aviation, social networking, and other commercial areas [14] are all examples of software systems. Software system development and design need resources such as money, domain-specific human specialists, time, tools, and infrastructures. Even if a software organisation has years of expertise in project design and development, software failures are on the rise (as shown in Table II), resulting in wasted resources including money, time, and effort. Every SDLC has its share of bugs, and sometimes customers don't give you the information you need because they aren't familiar with IT projects or the ramifications of politics and culture. Challenges in completing a project were another topic included in the study. Lack of user interaction [14], unclear goals [15], insufficient requirement definition, lack of resources[16], inadequate project planning and scheduling [17], poor communication among Team members [18], and poor Testing are most prevalent recognized causes of software failures. According to Table-1 below, the most important elements for a project's success are a lack of user involvement and an insufficient user demand definition. The leading causes of software failure are shown in Table I below.

Table I: Major factor for software failures [17]

Project Challenges Factor	**% of Responses**
Incomplete requirements and specifications	12.3%%
Changing requirements and specifications	11.8%
New technology	3.7%
Lack of User Input	12.8%
Technology incompetence	7.0%
Lack of resource	6.4%
Unrealistic expectations	5.9%
Lack of executive support	7.6%
Unrealistic time frames	4.3%
Unclear objectives	5.3%
Other	23.0%

According to CHAOS MANIFESTO's 2013 report, a survey was conducted between 2004 and 2012 on software projects, with results subdivided by category in Table II below.

Table II: Percentage detail from 2004 – 2012 [18]

	2004	2006	2008	2010	2012
Failed	18%	19%	24%	21%	18%
Successful	29%	35%	32%	37%	39%
Challenged	53%	46%	44%	42%	43%

F. Recommendation to Address Software Failures

In section V, we covered what tends to go wrong with software projects, but since every problem has a solution, we've included some suggestions for moving forward successfully.

- Put clear objectives and goal
- Monitor project to ensure that estimate is accurate.
- Do not depend on a single cost or schedule estimate.
- Fully satisfied the user requirements.

6. Literature Review

Important studies that use ML, neural networks and deep learning to the problem of predicting software faults are discussed below. We further elaborate on the considerations that led us to propose specific deep-learning techniques for Software Fault Prediction (SFP).

Khalid et al., (2023) The ultimate purpose of the research was to improve the model's accuracy and precision relative to previous investigations. Previous research indicates that there is room for even greater precision gains. K-means clustering was used to organise the classes for this purpose. Also, we used classification models to sort out particular characteristics. To get there, ML models are fine-tuned using Particle Swarm Optimization. To evaluate the models, we employed the f-measure, the confusion matrix, the performance error metrics, and the precision and accuracy measurements. All of the ML and enhanced ML models perform well, but SVM and optimized SVM models are most accurate (at 99 percent and 99.80 percent, respectively). Accuracy is 93.80% for NB, 93.80% for Optimized NB, 98.70% for RF, 99.50% for Optimised RF, and 97.60 for an ensemble method. Our ultimate objective was to improve upon the accuracy of past investigations, and we believe we have done so here [19]

In this paper, Yao et al. (2023) propose a programme semantics feature mining (PSFM) approach to software defect prediction. Specifically, the grammatical structure information and the text information in the code are parsed first, and then the semantic information is retrieved. The semantic data is then mined for the faulty feature. Finally, the mined defect features are used to make predictions about software defects. The experimental results show that the suggested technique in this work (PSFM method) improved the F-measure more than the state-of-the-art approaches to software fault prediction[20].

In this study, Jorayeva et al., (2022) posed nine research topics, for which 47 papers were culled from scholarly databases. The majority of research (48%) centered on Android apps; 92% employed supervised ML; and the majority of chosen metrics were object-oriented. decision trees, NB, SVMs, LR, and neural networks are the five most popular ML methods. In the academic world, object-oriented metrics were the norm. Deep learning techniques such as DBN (Deep Belief Networks), LSTM (Long Short-Term Memory), and Deep Neural Networks have only been used in a small number of research projects thus far (DNN). For mobile applications, this is the first systematic literature study on topic of software defect prediction. It will aid practitioners and academics alike in their pursuit of defect prediction in mobile software [21].

Using fine-tuned tree-based ensembles, Alazba and Aljamaan (2022) explore the usage of a stacking ensemble for defect prediction. Boosted trees, histogram-based gradient boosting, AdaBoost, gradient boosting, Random forest, XGBoost, and CatBoost were among the tree-based ensembles whose hyperparameters were optimized using a grid search. We then stacked the optimized tree-based ensembles we had built. Twenty-one publicly available defect datasets were utilized to evaluate the ensembles. Empirical results demonstrate that hyperparameter optimization significantly affects ensembles of extra trees and random forest models. When compared to our optimized tree-based ensembles, the stacking ensemble likewise outperformed them [22]

Tadapaneni et al., (2022) The proposed research uses the binary prediction dataset PROMISE. Due to the inherent binary nature of software fault prediction, a classification model was used for this investigation. This is why we use the NB on ML model and evaluate the performance of DNN and LSTM. The DNN algorithm performed better than competing methods in a controlled experiment designed to identify software bugs [23].

A hybrid heterogeneous ensemble strategy is suggested in this research by Alsawalqah et al., (2020) to forecast software defects. Classifiers in heterogeneous ensembles come from a variety of learning base approaches, each with its own set of advantages and disadvantages. The suggested method's central aim is to create high-quality heterogeneous categorization models maintained by specialists. The suggested method is developed in two different iterations and tested. The first uses conventional classifiers, while the second employs ensemble classifiers. The proposed method is evaluated by performing experiments on 21 publicly available benchmark datasets. The ensemble version was shown better than other well-regarded basic and ensemble classifiers in head-to-head assessments [24].

The software metrics that exist for defect prediction in software using ML are reviewed in depth by Meiliana et al. (2017). Our research shows that this is the first study of its kind to use the datasets available in the PROMISE repository to predict software faults. Several experiments from the PROMISE repository dataset are compared in attempt to develop an agreement on what constitutes successful software metrics and ML approach in software failure prediction [25].

Table III: Summary of Software Defect Prediction Using ML Algorithms from Various Researchers

Author	**Used techniques**	**Key findings**
Cai, X., Niu, Y., Geng, et al. [26]	SVM	G-Mean is used as a performance metric and a technique for SDP's state-of-the-art performance.
Akour, M., Alsghaier, H., et.al [27]	NB, Bayes net, PART, Random Forest, IB1, VFI, decision table, and NB tree base learners	According to findings, Random Forest Classifier is the most effective method.
C. W.	8 classifiers: NB,	combination of RF with
Yohannese and T. Li [28]	NN, SVM, RF, KNN, DTr, DTa, and RTr	Information Gain (IG) FS yields the highest Receiver Operating Characteristic (ROC) curve value.
Lov Kumar, Santanu Rath et al. [29]	Majority Voting Ensemble (MVE) method	The MVE method yields the highest efficiency. The MVE technique for fault prediction results in a model with lower overall fault removal costs than competing methods.

7. Machine Learning Algorithms

The purpose of this research is to investigate and evaluate ML algorithms. The research demonstrates the performance accuracy and capacity of ML algorithms for predicting software bugs and gives a comparative study of selected ML methods. In order to anticipate output values for novel input data, supervised ML algorithms attempt to create an inferring function by drawing conclusions about links and dependencies between known inputs and outputs of labeled training data. The chosen supervised ML algorithms and their brief descriptions are as follows. [30]:

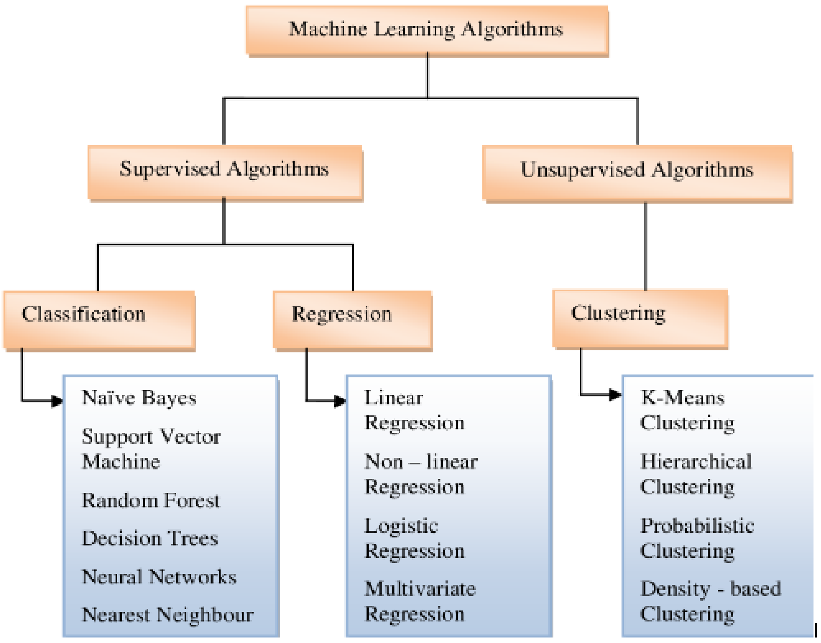

Fig. 3: Machine learning Algorithms

These are a few algorithms utilized for comparison purposes and are detailed below:

G. Logistic Regression

A method based on LR is utilized to make the forecast. It is used to establish the likelihood of a particular category, such as success or failure, victory or defeat, survival or annihilation. This may be expanded to show other categories of events, such as determining whether or not a given image has a cat, dog, lion, etc. Each item in the photo would be assigned a probability between 0 and 1, and the total would be totaled up to 1. Although many increasingly complex enhancements exist, this technique is primarily used in logistics and statistics to display a dependent variable in a binary (0 or 1) form [31].

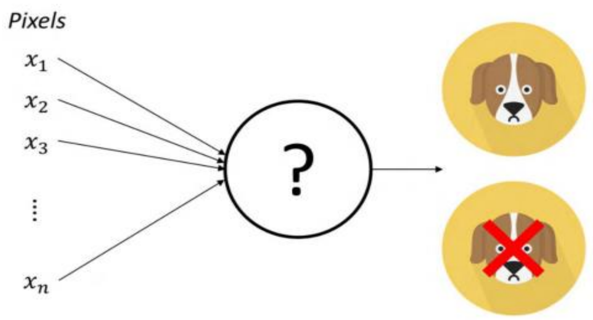

Fig. 4: Logistic Regression

H. *Random Forest*

Random forests, a popular ensemble learning approach built on the Bagging framework, take in unlabeled samples and spit out the classification results decided on by individual trees. This makes random forests useful for a wide variety of tasks, including classification, regression, and more. The random forest is an improvement over the decision tree in terms of performance since it incorporates the bagging approach into the decision tree. The use of randomization improves scalability and parallelism in the classification of high-dimensional data, while also optimizing anti-noise abilities and lowering the danger of over-fitting. The input dataset for a random forest does not need to be standardized, hence it may be used for both continuous and discrete situations requiring regression.

I. *Decision Tree*

As an example of a supervised learning algorithm, the decision tree is shown here. Each leaf node in a decision tree stands for a class label, whereas each interior node stands for a "test" on an attribute. Together, these nodes make up structure of a decision tree. Its framework may be used to reconstruct and comprehend previous decisions on a subject of interest. ID3 and C4.5 are examples of classic decision tree algorithms that can find optimal solutions to problems and resolve multi-stage decision problems [32].

J. *Support Vector Machine (SVM)*

SVM is a type of supervised learning which can be utilized for tasks other than straightforward regression and classification. It sets up according to how one thinks or what one values. Data used for relapse analysis and cluster analysis may be kept segregated with the use of supervised learning models (SVMs) in ML. Given a large number of training models, each of which is considered as possibly belonging to both classes, a SVM training algorithm builds a model that propagates new advisers for one class or another. Models in SVMs are shown as spotlights in space and are meant to hold examples of the various classes a normal distribution. Then, depending on which side of the gap new models are projected to fit into, plans are made to include them in the similar area [33].

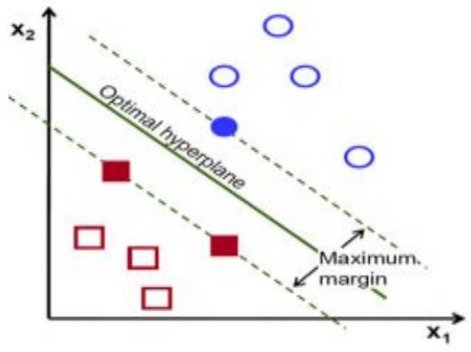

Fig. 5: Support Vector Machines

K. *Artificial Neural Network (ANN)*

ANN was first developed as oversimplified representations of the human nervous system, with the goal of replicating some of the latter's cognitive skills in a computer program. Better cognitive processes, including learning and memory, are a result of the enhanced cognitive abilities of ANNs, which not only learn from experience but also retrain themselves to adapt to new circumstances [34].

L. *Naive Bayes*

Probabilistic classifiers belonging to NB family. It assumes that the elements of the model are unrelated to one another and are grounded on Bayes' Theorem. In most cases, the presumption of autonomy is a false one. In contrast to other, more sophisticated classifier models, NB has been shown to be effective.

$$p(C \backslash x) = \frac{p(C)p(x \backslash C)}{p(x)} \tag{1}$$

8. Comparative Analysis of Machine Learning Algorithms in Sdp

We experimented with several ML models, including LR, ANN, RF SVM, and NB. This review is done using five different defect prediction datasets named KC2, PC1, JM1, DS1, and KC1. The accuracy score of the ML algorithms for different datasets is given below in table and figure.

Table IV: Comparison Results of ML Algorithms

Source	Datasets	Algorithms	Accuracy
[35]	KC2	Logistic Regression	83.7
[36]	PC1	Random Forest	91
[37]	JM1	Support Vector Machine	81.73
[38]	DS1	Artificial Neural Network	93.8
[39]	KC1	Naïve Bayes	82.10

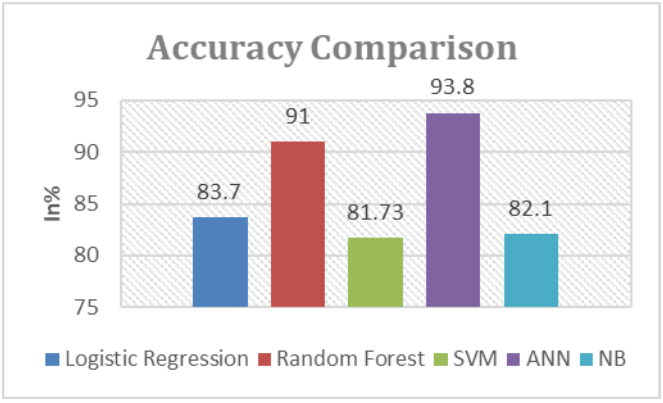

Fig. 6: Accuracy Comparison of ML Algorithms for Different Datasets

The above table IV and figure 6 shows comparative results of ML algorithms i.e., SVM, Random Forest (RF), ANNs (ANN), Naïve Bayes (NB), and LR, for variety of datasets like KC2, PC1, JM1, DS1, and KC1. The x-axis of graph represents the number of algorithms, while the y-axis represents number of accuracy scores in percent. From these results, we can see that the LR obtains the highest 83.7% accuracy for KC2 dataset, 91% accuracy for random forest for PC1 dataset, 81.73% accuracy for SVM for JM1 dataset, 93.8% accuracy for ANN for DS1 dataset, and 82.1% accuracy of naïve bayes for KC1 dataset, respectively. It is clearly shown that the ANN algorithm achieves the highest accuracy value for DS1 dataset compared to other algorithms.

9. Conclusion and Future Work

Due to its advantages, the development of software-based systems has increased in recent years. But, before the system is supplied to end users, its quality must be ensured. Many quality measures, including software testing, CMM, & ISO standards, have been developed to improve software quality. Today, the importance of software testing to software stability is increasing. Software flaws are one of the leading reasons for software failure. Software defect prediction is a method that creates a prediction model based on previous data in order to forecast future software problems. Predicting software defects may enhance the effectiveness of software testing and influence resource allocation. For error-prone modules, we should allocate additional time and resources. This study's primary purpose was to evaluate past research on software defects using ML techniques and datasets. Several ML techniques, Random Forest, SVM, LR, ANNs, and NB, have been analyzed in this research utilizing the datasets JM1, KC1, PC1, DS1, and KC2. For DS1 dataset, the most precise outcome (93.8 percent accuracy) is obtained using the ANN method. In conclusion, this study's ML algorithms can be utilized to discover software problems.

In a future project, we may incorporate other ML approaches and give a comprehensive contrast between them. In addition, adding additional software measures to learning process is one way to improve prediction model's accuracy.

References

[1] M. Shepperd, D. Bowes, and T. Hall, “Researcher bias: The use of machine learning in software defect prediction,” *IEEE Trans. Softw. Eng.*, 2014, doi: 10.1109/TSE.2014.2322358.

[2] M. Shepperd, T. Hall, and D. Bowes, “Authors’ reply to ‘comments on “researcher bias: The use of machine learning in software defect prediction,”’” *IEEE Trans. Softw. Eng.*, 2018, doi: 10.1109/TSE.2017.2731308.

[3] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, “Comments on Researcher Bias: The Use of Machine Learning in Software Defect Prediction,” *IEEE Trans. Softw. Eng.*, 2016, doi: 10.1109/TSE.2016.2553030.

[4] G. Czibula, Z. Marian, and I. G. Czibula, “Software defect prediction using relational association rule mining,” *Inf. Sci. (Ny).*, 2014, doi: 10.1016/j.ins.2013.12.031.

[5] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: Some comments on the NASA software defect datasets,” *IEEE Trans. Softw. Eng.*, 2013, doi: 10.1109/TSE.2013.11.

[6] R. Malhotra, “Comparative analysis of statistical and machine learning methods for predicting faulty modules,” *Appl. Soft Comput. J.*, 2014, doi: 10.1016/j.asoc.2014.03.032.

[7] N. Kalaivani, R. Beena, and A. Professor, “Overview of Software Defect Prediction using Machine Learning Algorithms,” *Int. J. Pure Appl. Math.*, vol. 118, no. 20, pp. 3863–3873, 2018.

[8] X. Peng, “Research on Software Defect Prediction and Analysis Based on Machine Learning,” 2022. doi: 10.1088/1742-6596/2173/1/012043.

[9] B. Yalciner and M. Ozdes, “Software Defect Estimation Using Machine Learning Algorithms,” *UBMK 2019 - Proceedings, 4th Int. Conf. Comput. Sci. Eng.*, no. 01, pp. 487–491, 2019, doi: 10.1109/UBMK.2019.8907149.

[10] J. Gao, L. Zhang, F. Zhao, and Y. Zhai, “Research on Software Defect Classification,” in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2019, pp. 748–754. doi: 10.1109/ITNEC.2019.8729440.

[11] Z. Cai, L. Lu, and S. Qiu, “An Abstract Syntax Tree Encoding Method for Cross-Project Defect Prediction,” *IEEE Access*, 2019, doi: 10.1109/ACCESS.2019.2953696.

[12] Q. Yu, J. Qian, S. Jiang, Z. Wu, and G. Zhang, "An Empirical Study on the Effectiveness of Feature Selection for Cross-Project Defect Prediction," *IEEE Access*, 2019, doi: 10.1109/ACCESS.2019.2895614.

[13] P. Paramshetti and D. A. Phalke, "Survey on Software Defect Prediction Using Machine Learning Techniques," *Int. J. Sci. Res.*, 2014.

[14] H. K. Dam, J. Grundy, T. Kim, and C. Kim, "A deep tree-based model for software defect prediction".

[15] G. Esteves, E. Figueiredo, A. Veloso, M. Viggiato, and N. Ziviani, "Understanding machine learning software defect predictions," *Autom. Softw. Eng.*, 2020, doi: 10.1007/s10515-020-00277-4.

[16] E. A. Felix and S. P. Lee, "Integrated Approach to Software Defect Prediction," *IEEE Access*, 2017, doi: 10.1109/ACCESS.2017.2759180.

[17] T. Clancy, "The Standish Group Report CHAOS," *Proj. Smart*, pp. 1–16, 2014.

[18] V. Chomal and J. Saini, "Cataloguing Most Severe Causes that Lead Software Projects to Fail," 2014.

[19] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software Defect Prediction Analysis Using Machine Learning Techniques," *Sustainability*, vol. 15, no. 6, 2023, doi: 10.3390/su15065517.

[20] W. Yao, M. Shafiq, X. Lin, and X. Yu, "A Software Defect Prediction Method Based on Program Semantic Feature Mining," *Electronics*, vol. 12, no. 7, 2023, doi: 10.3390/electronics12071546.

[21] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Machine Learning-Based Software Defect Prediction for Mobile Applications: A Systematic Literature Review," *Sensors*, vol. 22, no. 7, 2022, doi: 10.3390/s22072551.

[22] A. Alazba and H. Aljamaan, "Software Defect Prediction Using Stacking Generalization of Optimized Tree-Based Ensembles," *Appl. Sci.*, vol. 12, no. 9, 2022, doi: 10.3390/app12094577.

[23] P. Tadapaneni, N. C. Nadella, M. Divyanjali, and Y. Sangeetha, "Software Defect Prediction based on Machine Learning and Deep Learning," in *2022 International Conference on Inventive Computation Technologies (ICICT)*, 2022, pp. 116–122. doi: 10.1109/ICICT54344.2022.9850643.

[24] H. Alsawalqah *et al.*, "Software Defect Prediction Using Heterogeneous Ensemble Classification Based on Segmented Patterns," *Appl. Sci.*, vol. 10, no. 5, 2020, doi: 10.3390/app10051745.

[25] Meiliana, S. Karim, H. L. H. S. Warnars, F. L. Gaol, E. Abdurachman, and B. Soewito, "Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset," in *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, 2017, pp. 19–23. doi: 10.1109/CYBERNETICSCOM.2017.8311708.

[26] X. Cai *et al.*, "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," *Concurr. Comput. Pract. Exp.*, 2020, doi: 10.1002/cpe.5478.

[27] M. Akour, H. Al Sghaier, and O. Al Qasem, "The effectiveness of using deep learning algorithms in predicting students achievements," *Indones. J. Electr. Eng. Comput. Sci.*, 2020, doi: 10.11591/ijeecs.v19.i1.pp388-394.

[28] C. W. Yohannese and T. Li, "A Combined-Learning based framework for improved software fault prediction," *Int. J. Comput. Intell. Syst.*, 2017, doi: 10.2991/ijcis.2017.10.1.43.

[29] L. Kumar, S. Rath, and A. Sureka, "Using Source Code Metrics and Ensemble Methods for Fault Proneness Prediction," 2017.

[30] M. M. Moore, E. Slonimsky, A. D. Long, R. W. Sze, and R. S. Iyer, "Machine learning concepts, concerns and opportunities for a pediatric radiologist," *Pediatr. Radiol.*, 2019, doi: 10.1007/s00247-018-4277-7.

[31] S. MwanjeleMwagha, M. Muthoni, and P. Ochieng, "Comparison of Nearest Neighbor (ibk), Regression by Discretization and Isotonic Regression Classification Algorithms for Precipitation Classes Prediction," *Int. J. Comput. Appl.*, 2014, doi: 10.5120/16919-6729.

[32] P. Xu, "Review on Studies of Machine Learning Algorithms," 2019. doi: 10.1088/1742-6596/1187/5/052103.

[33] C. Pan, M. Lu, B. Xu, and H. Gao, "An improved CNN model for within-project software defect prediction," *Appl. Sci.*, 2019, doi: 10.3390/app9102138.

[34] S. Chen and D. Tan, "A SA-ANN-Based Modeling Method for Human Cognition Mechanism and the PSACO Cognition Algorithm," *Complexity*, 2018, doi: 10.1155/2018/6264124.

[35] M. Assim, Q. Obeidat, and M. Hammad, "Software Defects Prediction using Machine Learning Algorithms," *2020 Int. Conf. Data Anal. Bus. Ind. W. Towar. a Sustain. Econ. ICDABI 2020*, 2020, doi: 10.1109/ICDABI51230.2020.9325677.

[36] M. Azam, M. Nouman, and A. R. Gill, "Comparative analysis of machine learning techniques to improve

software defect prediction,” *KIET J. Comput. Inf. Sci. [KJCIS]*, vol. 5, no. 2, pp. 41–66, 2022.

[37] A. Alsaeedi and M. Z. Khan, “Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study,” *J. Softw. Eng. Appl.*, 2019, doi: 10.4236/jsea.2019.125007.

[38] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, “Software Bug Prediction using machine learning approach,” *Int. J. Adv. Comput. Sci. Appl.*, 2018, doi: 10.14569/IJACSA.2018.090212.

[39] S. Aleem, L. F. Capretz, and F. Ahmed, “Benchmarking Machine Learning Techniques for Software Defect Detection,” *Int. J. Softw. Eng. Appl.*, 2015, doi: 10.5121/ijsea.2015.6302.

[40] Alaria, S. K. "A.. Raj, V. Sharma, and V. Kumar.“Simulation and Analysis of Hand Gesture Recognition for Indian Sign Language Using CNN”." *International Journal on Recent and Innovation Trends in Computing and Communication* 10, no. 4 (2022): 10-14.

[41] Satish Kumar Alaria. Design & Analysis of Cost Estimation for New Mobile-COCOMO Tool for Mobile Application. *IJRITCC* 2019, *7*, 27-34.

[42] Najneen Qureshi, Manish Kumar Mukhija and Satish Kumar, "RAFI: Parallel Dynamic Test-suite Reduction for Software", New Frontiers in Communication and Intelligent Systems, SCRS, India, 2021, pp. 165-176. https://doi.org/10.52458/978-81-95502-00-4-20.