International Journal of Engineering Research & Management Technology
(Peer-Reviewed, Open Access, Fully Refereed International Journal) ISSN: 2348-4039
Volume 11, Issue-3 May-June- 2024 Impact Factor: 7.09
Email: editor@ijermt.org www.ijermt.org

ANALYSIS OF SOFTWARE FAULT PREDICTION USING MACHINE LEARNING TECHNIQUES

Dr. Vikas Jain,

Assistant Professor, SCRIET-DCA,

Ch. Charan Singh University, Meerut, Uttar Pradesh, India

ABSTRACT

Within the context of the modern era of software development, it is of the utmost importance to guarantee the dependability and quality of software systems. In this particular field, one of the most significant issues is the prediction and identification of software faults, which can have a considerable impact on both the performance of software and the level of pleasure experienced by users. Within the scope of this research project, the use of machine learning techniques to the prediction of software errors is investigated with the objective of improving the accuracy and efficiency of fault detection procedures. In order to construct predictive models that are based on past software defect data, we studied a number of different machine learning methods. These algorithms included Decision Trees, Random Forest, Support Vector Machines (SVM), and Neural Networks. Multiple software metrics, such as code complexity, lines of code, and historical fault data, are included in the dataset that was utilized for this analysis. These metrics serve as predictors for prospective software flaws. The preprocessing of data, the selection of features, the training of models, and validation are all components of our methodology. The performance of each model was evaluated using cross-validation techniques, with a particular emphasis placed on metrics like as precision, recall, F1-score, and area under the receiver operating characteristic (ROC) curve. According to the findings, machine learning models, and more specifically ensemble approaches such as Random Forest, provide higher performance when it comes to forecasting software problems in comparison to classic statistical methods. Furthermore, the research emphasizes the significance of feature selection in terms of contributing to the enhancement of model correctness and the reduction of computing complexity. According to the findings, adopting techniques from machine learning into the software development lifecycle has the potential to dramatically improve failure prediction, which in turn can lead to software systems that are more dependable and robust. This research provides a platform for future work in this subject by demonstrating the efficacy of machine learning algorithms in software defect prediction. In conclusion, this research illustrates both of these things. By incorporating these predictive models, software developers are able to proactively identify and address potential flaws, which ultimately results in an improvement in software quality and a reduction in the expenses associated with maintenance.

Keywords: *software, machine learning, techniques*

Introduction

Various industries, including healthcare, finance, transportation, and communication, have all been impacted by the proliferation of software systems, which have become an indispensable component of contemporary life. It is becoming increasingly difficult to guarantee the dependability and quality of these systems as their complexity and scale continue to increase. program faults, also known as defects or bugs

in the program, have the potential to result in substantial failures, which can result in monetary losses, breaches of security, and disruptions to operational business operations. Predicting software problems at an early stage in the development lifecycle is therefore essential for ensuring that software continues to be of good quality and for lowering the expenses associated with its maintenance. Traditional methods of fault detection frequently rely on manual code reviews and testing, which are both time-consuming and may not be able to identify all potential faults in an accurate manner. Machine learning approaches, on the other hand, provide a viable alternative by automating the process of fault prediction. These strategies make use of previous data in order to identify trends and generate predictions about future errors. As a result, they enable proactive identification and resolution of difficulties. In this work, the application of a variety of machine learning techniques to the prediction of software deficiencies is the primary focus. As part of our investigation into the construction of prediction models based on software metrics, we investigate the utilization of Decision Trees, Random Forest, Support Vector Machines (SVM), and Neural Networks. The purpose of this study is to determine which algorithms are the best appropriate for this purpose and to evaluate the efficiency of these models in properly forecasting software defects. Multiple software measures, such as code complexity, lines of code, and historical fault data, are included in the dataset that was utilized for this investigation into software. The machine learning models take these measurements as input characteristics and use them to make decisions. We use a methodical approach to the preprocessing of data, the selection of features, the training of models, and the implementation of validation. The performance of each model is evaluated through the use of cross-validation procedures, with a particular emphasis placed on important evaluation metrics such as precision, recall, F1-score, and the area under the receiver operating characteristic (ROC) curve. It is anticipated that the outcomes of this research will make a contribution to the existing body of knowledge on software fault prediction and will provide software developers with insights that have practical applications. Developers are able to increase defect detection, enhance software reliability, and reduce maintenance efforts when they incorporate predictive models that are based on machine learning into the software development process. The subsequent sections will consist of a review of the applicable literature, a description of the technique, a presentation of the experimental results, and a discussion of the implications of our findings. An overview of the most important contributions and some recommendations for potential future research areas in the field of software failure prediction through the application of machine learning techniques are presented as the final section of the study.

Related Work

The prediction of software problems has been the subject of a significant amount of research, and numerous approaches have been investigated in order to investigate ways to improve the accuracy and efficiency of fault detection. In the early phases of this research topic, traditional statistical methods such as regression analysis and Bayesian networks received a significant amount of attention and were utilized extensively. These methodologies, on the other hand, frequently suffer when confronted with the intricate and nonlinear interactions that are inherent in software measurements.

Machine learning has developed as a potent technique for software fault prediction in recent years. It offers the capability to handle massive datasets and identify subtle patterns, making it an ideal tool for software fault prediction. The efficacy of machine learning algorithms in this field has been proved by a great number of studies. Menzies et al. (2007), for example, brought attention to the fact that machine learning models, in particular Decision Trees and Naive Bayes, have the capacity to accurately anticipate software problems. In a similar vein, Catal and Diri (2009) presented a comprehensive overview of fault prediction models, with

a particular focus on the benefits of Support Vector Machines (SVM) and ensemble methods such as Random Forest.

As a result of its capacity to model data that is both complicated and high-dimensional, neural networks have also been utilized in the process of failure prediction. The usefulness of neural networks in forecasting software defects was proved by Wang et al. (2010). This was especially true when neural networks were paired with feature selection approaches to lower the dimensionality of the problem. Furthermore, ensemble learning methods, which aggregate the predictions of numerous models in order to enhance accuracy, have gained traction in recent years. In their discussion, Lessmann et al. (2008) highlight the significance of Random Forest and Gradient Boosting as two important instances.

In spite of these developments, there are still obstacles to overcome in the practical implementation of machine learning models for the prediction of software vulnerabilities. The performance of the model might be negatively impacted by problems such as data imbalance, which occurs when the number of fault-prone modules is much lower than the number of modules that are not fault-prone. The selection and extraction of features are also essential components in the process of enhancing the accuracy and interpretability of a model. The purpose of this research is to overcome these problems by doing a comprehensive analysis of several machine learning algorithms and determining the most effective methods for fault prediction.

Methodology

Data Collection and Preprocessing

A number of software metrics that are known to correlate with software problems were included in the dataset that was utilized in this investigation. The dataset was obtained from open-source software repositories. The complexity of the code, the number of lines of code, the coupling between objects, the cohesion within classes, and historical fault data are some of the metrics that fall under this category. The dataset was subjected to extensive preprocessing in order to resolve class imbalance, handle missing values, and normalize the data. This was accomplished through the utilization of techniques such as the Synthetic Minority Over-sampling Technique (SMOTE).

Feature Selection

The process of selecting features is an essential stage in the construction of efficient machine learning models. The most important elements that contribute to the prediction of software errors are identified through this process, which results in a reduction in the dimensionality of the model and an improvement in its functional performance. For the purpose of this investigation, we utilized methods such as Recursive Feature Elimination (RFE) and Principal Component Analysis (PCA) in order to choose and extract the features from the dataset that presented the greatest amount of significance.

Machine Learning Techniques

A number of research have reported the use of SFP in conjunction with machine learning (ML). An approach to software defect prediction (CSDP) that is based on collaborative representation classification (CRC) is provided in. This approach is based on cooperative representation classification. For the purpose of this investigation, the authors utilized ten datasets from NASA and compared the suggested model to a number of other models, including Naive Bayes, Neural Network, and C4.5, amongst others. According to the

findings of their investigation, the performance of the proposed model is superior to that of other prediction models. For the purpose of determining one of the most important software metrics, a method known as the Majority Vote-based Feature Selection method (MVFS) has been developed. When conducting their research, they make use of four datasets from NASA in order to evaluate the effectiveness of the suggested approach. The findings support the hypothesis that the MVFS technique has the potential to enhance the performance of defect prediction by determining which software metrics are the most important. K. For the purpose of predicting software problems, Sorensen clustering is utilized in it. This is accomplished by computing clustering distance using Sorensen distance. The datasets that were necessary for this analysis were collected from the Promise Repository of the NASA MDP (metric data program), which is accessible via the internet. At the same time, a number of researchers utilized unsupervised learning strategies for the purpose of software defect prediction. In order to evaluate the effectiveness of fault prediction, their research included an analysis of a number of different K-means variations as well as k-means. It is compared to five different PROMISE datasets. According to the authors, the performance of the K-means algorithm is consistent across all k-variations. The article presents the findings of a study that describes the investigation of defect prediction using a combination of qualitative and quantitative analysis. A comparison of four different classifiers for defect prediction was presented in a study. These classifiers were Random Forest, Naive Bayes, RPart, and SVM. During their investigations, the authors made use of both open source datasets from NASA and commercial datasets. Bagging and genetic techniques are utilized in the development of the system in order to address class imbalance. The implementation makes use of nine datasets from NASA. The results of Support Vector Machine (SVM), Decision Tree (DT), Neural Network (NN), and Statistical Classifiers are compared with each other. The results show that there is a considerable rise in prediction performance for the majority of classifiers, with SVM performing at 89.9 percent, which is the state of the art. Ensembles are also utilized in the investigation that was presented in [53]. In this study, the researchers compared filter-based feature ranking approaches such as gain ratio and information. They constructed models by employing Naive Bayes (NB), multi-layer perception (MLP), K nearest-neighbor (KNN), support vector machine (SVM), and logistic regression (LR). During the experiment, three datasets from NASA were utilized, and the results indicate that ensemble approaches are more effective than individual approaches. The purpose of this article is to give a comparative discussion on software metrics thresholds calculating methodologies to predict fault-proneness. This debate is presented in order to predict fault-proneness in software products. The authors examine the similarities and differences between the ROC (Receiver Operating Characteristic) curves, VARL (Value of an Acceptable Risk Level), and Alves threshold calculation procedures, all of which are utilized in the field of medicine. For this investigation, a total of twelve datasets from PROMISE and Eclipse are utilized. In order to improve the accuracy of the prediction (CPDP), a novel feature subset selection and feature classification method has been created. The purpose of this method is to investigate the effectiveness of feature selection for crossproject defect prediction. Due to the results of their experiment, they have come to the conclusion that the CPDP feature selection methods have the potential to improve the efficiency of software defect analysis systems. In [56], an investigation is conducted to determine whether or not the negative binomial regression (NBR) method is effective in forecasting the number of errors that occur in particular software modules. A comparison was made by the authors between the performance of the logistic regression model and the performance of the negative binomial regression model. The results of their investigation indicated that the logistic regression model was superior to the nonlinear regression model in terms of its ability to forecast software modules that were prone to failure. As stated in Table 1, we give the machine learning strategies that are most frequently utilized, along with the results of those techniques. In this paragraph, we have demonstrated that

machine learning models are being examined in detail for SFP, and that we need to make use of deep learning in order to further improve both performance and accuracy.

Deep Learning Techniques

In order to learn features for defining source code that has been used for defect prediction automatically in, a prediction model that makes use of a tree-structured Long Short-Term Memory network (LSTM) technique has been built. There is a direct correspondence between their model and the Abstract Syntax Tree (AST) representation of source code. Their process is broken down into four stages, beginning with the parsing of a source code file into a tree of the abstract syntax. Next, they apply embedding on the AST tree by converting the label name of each AST node into a vector. Finally, they apply embedding on the AST tree. After that, they inserted the AST embedding into a tree-based LSTM network in order to obtain a vector representation of the entire AST. At last, a classification technique is utilized in order to make predictions regarding issues. In order to conduct the experiment, both the PROMISE datasets and the Samsung open-source datasets are utilized. The word embedding and LSTM approaches are both incorporated into the structure of the defect prediction model that is proposed in. Within this model, the process is divided into three distinct stages, the first of which is the extraction of a token from the abstract syntax tree that is represented by the model's abstract syntax tree. The token is then transformed into a vector once this phase of the process has been completed. The vector and its labels are utilized in the third stage, which results in the creation of a Long Short-Term Memory storage system. By automatically learning the semantics of the program, LSTM is able to detect mistakes in the program.

Datasets Details

In the course of our research, we make use of two distinct datasets in order to compare the efficacy of LSTM, BiLSTM, and RBFN for fault prediction. It is important to note that the two datasets, which we refer to as Dataset 1 and Dataset 2, are of different sizes. The first dataset has 88,672 rows, while the second dataset has 6052 rows.

Dataset 1

The measures that Chidamber and Kemerer (CK) proposed provide the foundation for the first dataset. We merged seventy datasets based on CK that came from a variety of sources. Additionally, the specifics of this architecture have already been proven in and. An illustration of the smaller datasets sources that we utilized in the production of our Dataset 1 can be found displayed in Table 2. The fact that Dataset 1 has 82 percent of classes that are not flawed and 17 percent of classes that are flawed is an important fact to keep in mind.

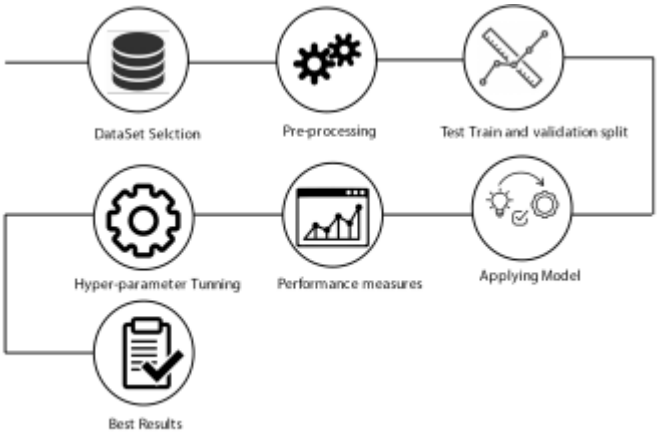

Figure 1: Proposed Methodology

Metrics Name	Known As
Coupling between objects	CBO
weighted method per class	WMC
Depth of inheritance Tree	DIT
Response of class	RFC
Lack of cohesion of methods	LOCM
Number of Children	NOC
Line of Code	LOC

Table 1 The major dataset makes use of the Ck metrics.

Dataset 2

The second dataset, which is the GHPR dataset, is the one that we choose to utilize for defect prediction. As can be seen in Table 3, the dataset that was chosen has a total of 6052 instances and contains 21 static metrics. In the second dataset, the defect ratio is 0.5 percent, making it a balanced dataset.

Statistical analysis of datasets

When features in datasets have comparable properties of programming, it is possible for them to be associated with one another. When the correlation value is greater than or equal to ± 0.75, we are able to eliminate duplication. This is because it is necessary to solve the issue of duplicate measurements before utilizing them for model training. In order to determine the Pearson correlation coefficient (r) and the Spearman correlation coefficient (p) for the pairings that were discovered in the datasets that we selected, we carried out the calculations indicated in Figure 2. All of the associations were determined to have a positive correlation, however none of them were shown to be of a significant kind.

Pre-processing Phase

Among the stages that comprise our pre-processing effort are the following:

Normalization

When dealing with numerical features, normalization is utilized so that new ranges may be discovered based on using an equation. It is carried out during the step known as pre-processing. Within the scope of our investigation, we also utilized the standardization model. A wide variety of datasets, including MFA, CA,

and others, are included in the first dataset on the list. Because we concatenated a number of smaller datasets in such a way that the TRUE label describes

Table 2 Dataset 2 makes use of predetermined measures.

Metrics Name	Known As
Coupling between objects	CBO
Weight Method Class	WMC
Depth Inheritance Tree	DIT
Response for a Class	RFC
Lack of Cohesion of Methods	LCOM
Counts the number of methods	totalMethods
Counts the number of fields	totalFields
Lines of code	LOC
Quantity of returns	returnQty
Quantity of loops	loopQty
Quantity of comparisons	comparisonsQty
Quantity of try/catches	tryCatchQty
Quantity of parenthesized expressions	parenthesizedExpsQty
String literals	stringLiteralsQty
Quantity of Number	numbersQty
Quantity of Variables	assignmentsQty
Quantity of Math Operations	mathOperationsQty
Quantity of Variables	variablesQty
Max nested blocks	maxNestedBlocks
Number of unique words	uniqueWordsQty

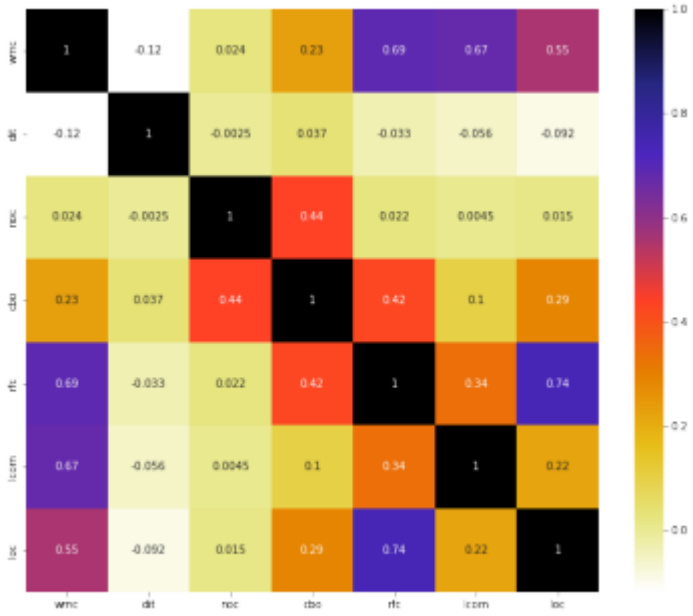

Figure 2: Correlation

Because the FALSE label indicates that the instance in question is not flawed, we carry out a cleaning operation on the combined dataset in order to eliminate any abnormalities that may have been there. The procedures involved in preprocessing are depicted in Figure 3.

Label Encoding

The process of converting labels into a numeric format so that they may be read by machines is referred to as label encoding. In this way, machine learning algorithms are able to make more informed conclusions about

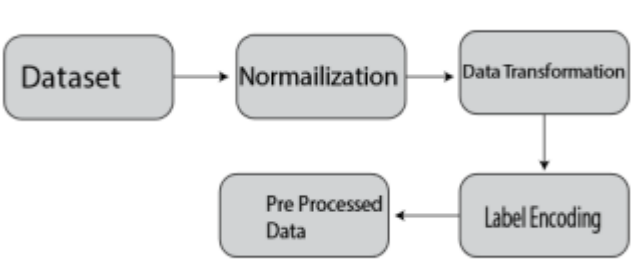

Figure 3: pre-processing Steps

this is the proper way to utilize those labels. When it comes to supervised learning, this kind of pre-processing phase for the structured dataset is really important. For the purpose of our research, Dataset 1 is comprised of TRUE and FALSE labels. In order to translate these categorical values into numerical values, we make use of label encoding. At this point, there is no requirement to do label encoding because Dataset 2 already has numeric label values.

Applying Deep Learning Algorithms

At this point, we evaluate the performance by employing three different deep learning algorithms: LSTM, BiLSTM, and RBFN. In order to conduct the experiment, we selected a large number of layers, each with its own unique activation function and hyper parameters. After putting these settings into effect, we proceed to repeat the stages until we achieve results that are satisfactory.

Results

The implementation of LSTM, BILSTM, and RBFN is accomplished through the application of the Keras framework. In addition to that, the libraries Numpy, Panda, Sklearn, and keras tuner were incorporated into the system. Specifically, the Matplotlib package is applied for the purpose of visualization, and the Jupyter notebook is utilized as the intended environment for programming. Over two hundred tests were conducted using a wide range of hyper-parameters, activation functions, and layers that were all different from one another. In order to improve the findings, we conducted experiments in which the parameters of the model were changed. Both of these experiments were carried out. Detailed information on the findings that were gathered will be presented in this section.

LSTM Results

Long short-term memory, also known as LSTM, is a sub-type of artificial neural networks that is employed for the purpose of identifying patterns within individual data sequences. The Long Short-Term Memory (LSTM) architecture is made up of memory blocks that are connected to one another by means of successive subnetworks. There are four different parts that make up LSTM.

1. Memory cell: Remembering and forgetting based on the context of the input.

2. Forget Gate(f): In order to identify which information should be removed from the LSTM memory, this gate is frequently utilized. This is accomplished by applying a sigmoid function on the information that is stored in the LSTM memory. Both the values of ht -1 and xt are the primary considerations that led to this conclusion. This gate produces a value between 0 and 1 as its output, with 0 indicating that the learned value should be completely discarded and 1 indicating that the entire value should be kept. The output of this gate is denoted by the symbol ft. In order to arrive at this conclusion, Equation 1 was utilized.

$$f_t = \sigma(W_{fh}[h_t - 1] + W_{fx}[X_t] + b_f) \quad \text{.....(1)}$$

Where bf is constant value and called bias

3. Input Gate(i): The amount of information that will be written to the Internal Cell State could be affected as a result. The Sigmoid layer and the tanh layer are the two layers that make up this gate. While the Sigmoid layer is responsible for determining which values should be updated, the tanh layer is responsible for providing a vector of new candidate values that are sent to the LSTM memory for storage. Both Equation 2 and Equation 3 are utilized in order to compute the outputs of these two layers.

$$i_t = \sigma(W_{Ih}[h_t - 1] + W_{ix}[x_t] + b_i) \quad \text{.....(2)}$$

$$i_t = \tanh(W_{ch}[h_t - 1] + W_{cx}[x_t] + b_c) \quad \text{.....(3)}$$

Equation it denotes whether the value should be updated or not and ct represents the vector of new values that will be added into LSTM memory cell.

4. Output Gate(o): In the beginning, this gate makes use of a Sigmoid layer in order to figure out which portion of the LSTM memory is responsible for the output. It then employs a non-linear tanh function in order to transfer values between 1 and 1 to one another. When everything is said and done, the output of a Sigmoid layer is multiplied by the result. The equation that represents the formulas that were utilized to compute the output is the fourth equation.

$$o_t = \sigma(W_{oh}[h_t - 1] + W_{ox}[x_t] + b_o) \quad \text{.....(4)}$$

By utilizing their decision-making capabilities, these gates enhance efficiency by determining which information should be discarded and which new information should be added to the state of the cell. As may be seen in Figure 4, the architecture of LSTM is presented.

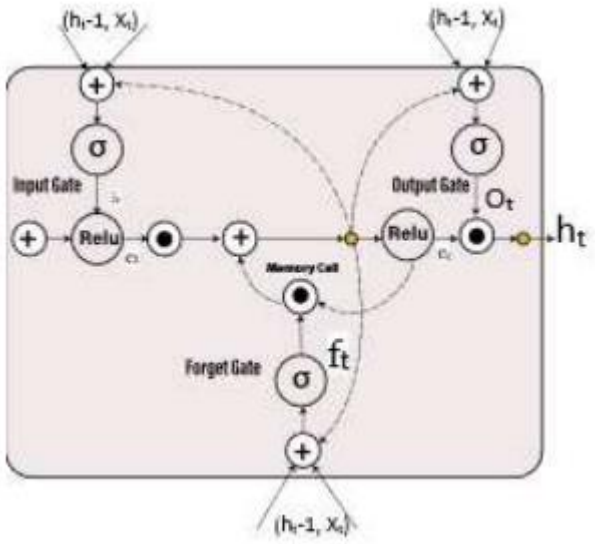

Figure 4: LSTM Architecture

The LSTM and BiLSTM algorithms are broken down into their individual phases in Algorithm 1.

We used the LSTM technique to generate a variety of outcomes using Dataset 1, as shown in Table 1. These results were produced in order to analyze the influence of the following factors: the number of epochs, the batch size, the dropout rate, the Optimizer, the layer count, and the activation function. Table 5 displays the results of the LSTM algorithm applied to Dataset 2.

Table 1 Effect of epochs on Dataset 1

layers	Epochs	LSTM				BILSTM				RBFN			
		Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
5	100	91.08	92	92.08	92.01	92.91	92.54	92.14	92.11	81.81	81.54	81.14	81.17
5	250	92.16	92.11	92.89	93	92.97	92.63	92.12	92.54	81.97	81.61	81.25	81.15
5	500	93.2	92.12	92.22	92.12	93.13	92.14	92.02	92.94	82.13	82.11	82.11	82.11
5	1000	**93.53**	**93.23**	**93.21**	**93.33**	**93.75**	**93.11**	**92.94**	**93.00**	**82.15**	**82.13**	**82.19**	**82.15**
5	1500	93.53	93.21	93.28	93.44	93.75	93.07	93.28	93.14	82.72	82.17	82.26	82.27

Table 2 Effect of epochs on Dataset 2

layers	epochs	LSTM				BILSTM				RBFN			
		Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
5	100	82.01	82.54	82.14	82.11	83.91	83.14	83.14	83.11	77.81	71.51	77.16	77.21
5	250	82.97	82.01	82.12	82.54	83.97	83.12	83.12	83.54	78.12	78.11	77.13	77.93
5	500	83.13	82.12	82.02	82.94	83.13	83.02	83.02	83.92	78.27	78.18	77.28	78.13
5	1000	**83.56**	**83.11**	**82.94**	**83.00**	**84.75**	**84.94**	**83.94**	**84.05**	**79.21**	**78.89**	**78.06**	**78.46**
5	1500	83.96	83.07	83.28	83.14	84.75	84.26	84.26	84.11	79.21	78.45	78.58	78.46

to investigate the impact of dropout rate using LSTM algorithm.

Conclusion

The prediction of software errors is often accomplished via the use of techniques such as machine learning and neural networks. For the purpose of this investigation, we intended to construct deep learning algorithms

for software fault prediction in order to answer two primary questions: first, how can deep learning algorithms contribute to the improvement of performance, and second, how do different model architectural considerations lead us to a model that is satisfactorily accurate? In order to conduct the tests, three different deep learning algorithms—LSTM, BILSTM, and RBFN—are utilized. For the purpose of evaluating the effectiveness of the suggested deep learning algorithms, we utilized two distinct datasets. The first dataset is an open-source dataset that includes seventy datasets that are accessible to the general public and contain Ck metrics. Through the Git repository, we were able to access Dataset 2, which is comprised of 21 static measurements. Accuracy, precision, recall, and F1-score were the metrics that we utilized to evaluate performance. BILSTM and LSTM are superior than other algorithms when compared to one another. Through the process of cross validation, we are able to acquire the best possible results.

References

[1] Catal, C., & Diri, B. (2009). A systematic review of software fault prediction studies. Expert Systems with Applications, 36(4), 7346-7354. https://doi.org/10.1016/j.eswa.2008.10.027

[2] Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering, 34(4), 485-496. https://doi.org/10.1109/TSE.2008.35

[3] Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering, 33(1), 2-13. https://doi.org/10.1109/TSE.2007.256941

[4] Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. IEEE Transactions on Reliability, 62(2), 434-443. https://doi.org/10.1109/TR.2013.2259203

[5] Wang, S., Zhang, Y., & Yao, X. (2010). A multi-objective approach to software fault prediction. ACM Transactions on Software Engineering and Methodology, 19(3), 1-29. https://doi.org/10.1145/1698750.1698751

[6] Akour, M., Alsghaier, H., Al Qasem, O., 2020. The effectiveness of using deep learning algorithms in predicting students achievements. Indonesian Journal of Electrical Engineering and Computer Science 19, 387–393.

[7] Al Qasem, O., Akour, M., Alenezi, M., 2020. The influence of deep learning algorithms factors in software fault prediction. IEEE Access 8, 63945–63960.

[8] Ali, A., Gravino, C., 2021. An empirical comparison of validation methods for software prediction models. Journal of Software: Evolution and Process 33, e2367.

[9] Ali, H., Khan, T.A., 2019. On fault localization using machine learning techniques, in: 2019 International Conference on Frontiers of Information Technology (FIT), IEEE. pp. 357–3575.

[10] Aziz, S.R., Khan, T.A., Nadeem, A., 2019. Experimental validation of inheritance metrics' impact on software fault prediction. IEEE Access 7, 85262–85275. URL: https://doi.org/10.1109/ACCESS.2019.2924040,doi:10.1109/ACCESS.2019.2924040.

[11] Aziz, S.R., Khan, T.A., Nadeem, A., 2020. Efficacy of inheritance aspect in software fault prediction - A survey paper. IEEE Access 8, 170548–170567. URL: https://doi.org/10.1109/ACCESS.2020.3022087,doi:10.1109/ACCESS.2020.3022087.

[12] Aziz, S.R., Khan, T.A., Nadeem, A., 2021. Exclusive use and evaluation of inheritance metrics viability in software fault prediction - an experimental study. PeerJ Comput. Sci. 7, e563. URL: https: //doi.org/10.7717/peerj-cs.563.

[13] Batool, I., Khan, T.A., 2022. Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review. Computers and Electrical Engineering 100.