IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON NEW TRENDS IN BRAIN SIGNAL PROCESSING AND ANALYSIS

Received April 1, 2019, accepted April 21, 2019, date of publication April 25, 2019, date of current version May 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2913349

Deep Learning Approach for Software Maintainability Metrics Prediction

SUDAN JHA[1], RAGHVENDRA KUMAR[2], LE HOANG SON[3], MOHAMED ABDEL-BASSET[4], ISHAANI PRIYADARSHINI[5], ROHIT SHARMA[6], AND HOANG VIET LONG[7,8]

[1]School of Computer Engineering, KIIT University, Bhubaneswar, India
[2]Computer Science and Engineering Department, LNCT College, Bhopal, India
[3]VNU Information Technology Institute, Vietnam National University, Hanoi, Vietnam
[4]Department of Operations Research, Zagazig University, Zagazig, Egypt
[5]Department of Electrical and Computer Engineering, University of Delaware, Newark, DE, USA
[6]Department of Electronics and Communication Engineering, SRM Institute of Science and Technology, NCR Campus, Ghaziabad, India
[7]Division of Computational Mathematics and Engineering, Institute for Computational Science, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam
[8]Faculty of Mathematics and Statistics, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam

Corresponding author: Hoang Viet Long (hoangvietlong@tdtu.edu.vn)

ABSTRACT Software maintainability predicts changes or failures that may occur in software after it has been deployed. Since it deals with the degree to which an application may be understood, repaired, or enhanced, it also takes into account the overall cost of the project. In the past, several measures have been taken into account for predicting metrics that influence software maintainability. However, deep learning is yet to be explored for the same. In this paper, we perform deep learning for software maintainability metrics' prediction on a large number of datasets. Unlike the previous research works, we have relied on large datasets from 299 software and subsequently applied various metrics and functions to the same; 29 object-oriented metrics have been considered along with their impact on software maintainability of open source software. Several metrics have been analyzed and descriptive statistics of these metrics have been pointed out. The proposed long short term memory has been evaluated using measures, such as mean absolute error, root mean square error and accuracy. Five machine learning algorithms, namely, ridge regression with variable selection, decision tree, quantile regression forest, support vector machine, and principal component analysis have been applied to the original datasets, as well as, to the refined datasets. It was found that this paper provides results in the form of metrics that may be used in the prediction of software maintenance and the proposed deep learning model outperforms all of the other methods that were considered. Furthermore, the results of experiment affirm the efficiency of the proposed deep learning model for software maintainability prediction.

INDEX TERMS Deep learning, machine learning, software metrics, software maintainability, prediction.

I. INTRODUCTION

Nearly 70% of the time in software development is for maintenance and this involves huge costs for the developers [1]. New software these days are very complex and the size of software has increased considerably, thereby making the software increasingly difficult to maintain [2]. Software maintainability is directly related with the economic performance and success of a product or an organization [3]. Using maintainability, we can predict what changes or failures that may occur in software after it has been deployed. Maintainability of the software is therefore a quality attribute for software that helps to determine the performance of the software [4]. Developers are assisted to determine probable amount of change that might occur in software modules during the maintenance phase [5], [6]. Owing to the correctness of the maintainability predictions, the design of software can be improved to determine changes that need to be made for development of software modules in the future [7].

Software maintainability is a big challenge that the computer industry is facing [8]. The reason behind is to make the system automated. There have been many machine learning and artificial intelligence approaches that have been used till date [9]. In majority of the approaches, only a small number of software has been used for testing [10]. It has been observed from the previous works that there is no research

The associate editor coordinating the review of this manuscript and approving it for publication was Victor Hugo Albuquerque.

for Software Maintainability Metrics Prediction using Deep Learning. Besides, little enhancements can be seen in the software maintenance predictability from those works. Indeed, datasets are not sufficient enough to give conclusive results which provide information regarding the input data of the system [11]. The related work regarding Software Maintainability Metrics Prediction in [12]–[29] are mainly based on applying different machine learning algorithms like ridge regression and artificial neural networks on two to three software models and providing a comparative study between the different software metrics and machine learning algorithms.

It has been observed from the previous works that there is no research for Software Maintainability Metrics Prediction using Deep Learning [30]. A review of the previous published works in this area shows that very few enhancements have been made in the study of the software maintenance predictability. Most of the previous works in this area have utilized artificial intelligence techniques such as machine learning, soft computing, fuzzy networks and evolutionary algorithms [31], [32]. Moreover, the datasets used in these studies are small scale with a very small number of software. Therefore, the testing process cannot be considered to be accurate, and the results obtained may not necessarily reflect actual systems in which a large number of software of different sizes interacts with one another [33]. The existing works in this area have also not addressed the issue of selecting the most relevant metrics in a systematic manner using methods such as the feature subset selection (FSS) algorithms. This is a major flaw in the existing works as the selection of relevant metrics for the computation process is an important phase in the decision making process and has the potential to significantly impact the results. This deficiency led us to use to an analysis of a sufficiently large number of open source software using deep learning algorithms [34].

Deep learning is one of the many techniques of machine learning that are available in literature. Deep learning uses data representation rather than task specific learning methods [35]. This method uses multiple layers of nonlinear processing units which will be used for feature extraction. The inputs from the earlier layers are used as the output for each of the successive layers. The learning process can be fully supervised (e.g. classification), semi-supervised or unsupervised (e.g. pattern analysis), and this method uses a hierarchy concept where each layer learns the multiple levels of representation that correspond to different levels of abstraction [36]. The network learns and remembers the sequence of input data and in each layer, and subsequently determines which data to remember and which data to forget. In this way, at the end of the process, we are able to determine which metrics will be added to software maintainability to make accurate prediction.

In this paper, we propose a new Deep Learning based method for Software Maintainability Metrics Prediction and apply this method on a large number of datasets. Twenty-nine object-oriented metrics have been considered and their impact on software maintainability of open source software has also been analyzed. The results based on the study highlight several metrics that should be used in the prediction of the software maintenance. Unlike the previous research works that mostly use a small number of datasets, we have chosen to use large datasets from 299 software to which we have applied the selected metrics that was determined using the deep learning algorithm. It has been proven that our predictions are more accurate as it is validated through experiments and systematic computation processes.

The rest of the paper is organized as follows. Section 2 introduces the related works and provides a background of the topic of study. In Section 3, the research methodologies that are considered are explained and demonstrated. Section 4 provides the results obtained through the experimentation process, followed by a comprehensive discussion of the experimental results. Finally, a summary of the overall work and the issues learned in the implementation of the investigated work are presented in Section 5. Concluding remarks are presented in Section 6.

II. RELATED WORK

Ahmed and Al-Jamimi [8] proposed a procedure for predicting software maintainability using fuzzy rationale based straightforward quality forecast models. Li and Smidts [37] dealt with consistently screen the software worked by end clients, naturally gathering issues and prescribing conceivable fixes to designers. A review of the various artificial intelligence techniques used in the modeling of machining processes [9] found that fuzzy logic is the most commonly used artificial intelligence technique used for this purpose. The review covered the abilities, limitations, and effectual modifications of fuzzy logic in the modeling of machining purposes. Kaur and Kaur [18] proposed statistical examination of displaying techniques for software viability forecast. Ionescu *et al.* [16] presented a computerized answer for software advancement exertion estimation based on content depictions of errands and exercises, joined with accessible measurements. Malhotra [34] examined machine learning techniques for blame forecast. Elish *et al.* [11] assessed diverse homogenous and heterogeneous outfit strategies in anticipating software upkeep exertion and change inclination. Radjenović *et al.* [26] attempted to recognize software measurements and to evaluate their pertinence in software blame forecast and researched the impact of setting on measurements' determination and execution.

Malhotra [25] presented a survey for machine learning methods for software blame expectation. Lin *et al.* [21] proposed an improved Support Vector Machines (SVM) based method by considering the complex literary and semantic connection highlights based on a past SVM-based discriminative plan (SVM-54). Yang *et al.* [29] proposed a two-layer ensemble learning approach for just-in-time imperfection expectation. Iyer *et al.* [6] introduced a completely data driven approach that uses end-to-end neural consideration for deriving source codes. Francese *et al.* [13] exhibited an instrument based on static examination of source code.

Rahimi and Zargham [27] proposed vulnerabilities crying for defenselessness disclosure expectation based on code properties. Menzies [23] explored software ecosystems and challenges of their maintenance and evolution.

Jamshidi *et al.* [17] applied exchange learning on four prevalent software frameworks, shifting software designs and natural conditions. Chhabra [10] proposed a multi objective advancement to deal with enhancement of a protest arranged framework with least conceivable development of classes between existing packages of unique software modularization. Rattan *et al.* [38] distinguished 213 examinations and displayed the outcomes in various measurements like order of clone inquire about, code clone administration as cross cutting space, sorts of clones, clone recognition instruments, clone discovery approaches, inward portrayals, subject frameworks, semantic clones and model clones. Finlay *et al.* [12] studied the idea of using stream mining techniques to determine the most useful metrics for predicting build success or failure. The authors found that the Hoeffding Tree method is more stable and robust compared to other traditional data mining methods. Huo *et al.* [14] proposed a convolution neural system which uses lexical data and program structure data to learn together the highlights from natural language and source code in programming language for consequently finding the potential buggy source code as indicated by bug report. Luo *et al.* [22] proposed an arrangement for naturally discovering execution bottlenecks. Tavakoli *et al.* [28] presented a fixation expectation and saliency displaying system, whereas Haije *et al.* [15] investigated viability of a machine translation approach to automatic comment generation.

In view of the increasing popularity of deep learning methods, there have been many review papers related to the application of deep learning methods in various fields and its superiority. LeCun *et al.* [39] presented a thorough review of deep learning methods including the improvements made through the application of deep learning methods in speech recognition, visual object recognition, drug discovery and genomics. Druzhkov and Kustikova [40] presented an overview of deep learning methods such as autoencoders, restricted Boltzmann machines and convolutional neural networks in the study of image classification and object detection. The duo also presented a comparison of the existing software packages for deep learning. Guo et al. [3] presented a comprehensive state-of-the-art review of more than 210 research papers related to the application of deep learning algorithms in computer vision, image classification, object detection, and image retrieval, as well as the future trends and challenges in the study of deep learning methods. Gulshan *et al.* [41] used a type of neural network called the deep convolutional neural network which is optimized for image classification for the detection of diabetic retinopathy in retinal fundus photographs. Recognizing the increasing use of deep learning algorithms in the analysis of medical images, Li and Smidts [37] presented a review of 300 research papers related to the application of deep learning methods in image classification and segmentation. Liu *et al.* [42] presented an overview of four deep learning architectures, and its application in speech recognition, pattern recognition and computer vision. Min, *et al.* [43] reviewed the progress made in the application of deep learning methods in bioinformatics, whereas Miotto *et al.* [44] reviewed the recent literature related to the application of deep learning methods in the domain of healthcare and biomedicine. Sun *et al.* [45] studied the application of the deep neural network method in solving problems related to fault diagnosis for induction motors.

Cheng *et al.* [46] presented a system for indicating, preparing, and assessing and conveying machine learning methods focused on streamlining bleeding edge machine learning for practitioners with a specific end goal to bring such advances into creation. Nam *et al.* [47] proposed heterogeneous defect prediction (HDP) to anticipate deserts crosswise over activities with heterogeneous metric sets. Tavakoli et al. [28] presented a novel obsession forecast and saliency demonstrating system in light of inter-image similitude and ensemble of Extreme Learning Machines (ELM). Singh and Chug [48] analyzed the most popular & widely used machine learning methods of Artificial Neural Network (ANN), Particle Swarm Optimization (PSO), Decision Trees (DT), Naïve Bayes (NB) & Linear Classifier (LC) namely using KEEL tool & validated using k-fold cross validation techniques. The authors found that the linear classifier method proved to be superior to the other algorithms in terms of defect prediction accuracy. Litjens *et al.* [49] presented CCLEARNER, the principal exclusively token-based clone detection approach which utilizes deep learning.

Fonton *et al.* [50] was in fact due to the specific dataset employed rather than the actual capabilities of machine-learning techniques for code smell detection. Liu *et al.* [51] proposed to utilize the Historical Version Sequence of Metrics (HVSM) in consistent software versions as imperfection predictors. Chen *et al.* [52] proposed a multi-target optimization based supervised strategy MULTI to construct JIT-SDP models. Miholca *et al.* [53] built up a novel supervised classification strategy called HyGRAR for software imperfection forecast. Krishna and Menzies [54] have undertaken a detailed study of transfer learners known as ''bellwethers'' which is a method that can be used to reduce conclusion instability in the building of quality prediction models for software projects.

III. METHODOLOGY

In this section, we present the steps to apply Deep Learning for Software Maintainability Metrics prediction. Firstly, the data collection process including software metrics is described in Section 3.1. Subsequently, the Deep Learning architecture and evaluation metrics are shown in Sections 3.2 & 3.3, respectively.

A. DATASETS

Software metrics compute various aspects of software that describe the functional and physical characteristics of a

TABLE 1. Software metrics.

No.	Metric	Metric Suite	Definition
1.	LOC	-	The total lines of code which is basically the size of a software program
2.	SLOP-P	-	Physical source lines of code and comprises of text of the program's code along with comment lines and blank lines in the code
3.	WMC	C&K	Weighted methods per class basically indicate the summation of McCabe's complexities of all local methods
4.	CBO	C&K	Coupling between object classes gives a count of the efferent and afferent couplings to a specific class. High CBO values is undesirable for efficient class maintainability
5.	RFC	C&K	It stands for response of a class. It calculates the cardinality of the response set of a class wherein the response set includes all the local methods along with all the methods called by the local methods
6.	NChange	-	Normalized Change is the dependent variable which indicates the number of changed lines of code per class. A Change could be an addition, deletion or a modified line which is considered as an addition and a deletion. The value of Change has been normalized using the formula: where max_t – maximum desired target value i.e. 100, and min_t – minimum desired target value i.e. 0 max_a – maximum value of actual variable min_a – minimum value of actual variable V – Change

TABLE 2. Details of open source software systems.

Software Used	Version	No. of Classes	Number of Common Classes	Number of Classes Changed	% of Changed Classes	LOC of Common Classes	LOC of Changed Classes
antlr4	3.6.0	2506	2189	886	40.47	598976	352503
mcMMO	3.7.0	3599	2187	885	40.14	598974	352507
Junit	7.0.30	1734	1407	728	51.74	463780	336784
Mct	7.0.40	2447	1247	736	51.47	468522	336685
Mapdb	2.16.4	2376	1920	1085	56.51	217443	174263
Oryx	2.17.4	2604	1852	1078	52.78	217534	174532

component, process or project. The different metrics that have been used for this empirical study are described in Table 1.

Different software's have different numbers of classes. The number of common classes is 2189, and this is the number of classes that are common to all 29 open source OO projects. Change in a common class is the total count of added, deleted and modified lines. A modification may be counted as a deletion and an addition. Out of 2189, 886 classes were altered and 1303 were unaltered. Version 7.0.30 of Antlr4 was released on 5th September, 2011 and its source code had 1734 classes, whereas version 7.0.40 was released on 9th May, 2013 and consisted of 2447 classes.

Two versions of Antlr4 had 1407 common classes and out of these 1407 classes, 728 classes were changed and the remaining 679 were not changed. Version 2.16.4 of Junit was released on 18th September, 2016 and its source code consisted of 2376 classes. Similarly, version 2.17.4 was released on 24th November, 2016 and its source code consisted of 2604 classes. Two versions of Junit had 1920 common classes. Out of 1920, 1085 classes were altered and 835 were unaltered. These have been summarized in Table 2.

Table 3 describes the error prediction of Change on the basis of subset of metrics acquired using FSS algorithms. The results obtained after examining with smallest subset that controls and determines the requirement of newer versions of a project to enhance the structural quality were identified. The descriptive statistics of the dataset obtained from Mapdb, McMMO and Mct are summarized in Tables 4, 5 and 6 respectively. The statistics describe the correlation of features (software metrics) with NChange for Mapdb, McMMO and Mct datasets, respectively. The dataset comprises of twenty-nine class-level metrics with 28 dependent variables and one normalized dependent variable NChange.

The following observations were noted from the descriptive statistics of the dataset and the following measures were taken accordingly: C & SLOC, HCLOC and HCWORD were removed from all the three dataset because their values were 0 for all the classes. The mean value of NChange is the

TABLE 3. Analysis of various metrics and the comparison results.

Metrics – I	Metrics – II	Metrics – III	Metrics – IV	Metrics – V	Metrics - VI	Classification accuracy of Neural Network (NN) after using these metrics
WMC	CBO	CBOI	NII	NOI	RFC	0.82
CBO	CBOI	NII	NOI	RFC	AD	0.79
CBOI	NII	NOI	RFC	AD	CD	0.73
NII	NOI	RFC	AD	CD	CLOC	0.89
NOI	RFC	AD	CD	CLOC	DLOC	0.94
RFC	AD	CD	CLOC	DLOC	PDA	0.73
AD	CD	CLOC	DLOC	PDA	PUA	0.83

TABLE 4. Descriptive statistics of the mapdb dataset.

Metric	Minimum	Maximum	Mean	Median	Standard Deviation	Variance
LOC	1	12548	273.63	153	586.16	343574.90
SLOC-P	1	11468	195.51	90	508.15	258207.90
SLOC-L	0	6753	124.10	63	305.49	93321.67
MVG	0	944	22.53	10	50.62	2562.07
BLOC	0	2443	74.52	39	125.34	15708.74
C&SLOC	0	0	0	0	0	0
CLOC	0	414	3.61	0	14.73	216.98
CWORD	0	552	4.82	0	19.64	385.73
HCLOC	0	0	0	0	0	0
HCWORD	0	0	0	0	0	0
WMC	0	1078	25.16	8	66.36	4403.61

TABLE 5. Descriptive statistics of the Antlr4 dataset.

Metric	Minimum	Maximum	Mean	Median	Standard Deviation	Variance
LOC	1	4849	270.65	124	446.37	199250.26
SLOC-P	1	3077	153.45	51	290.18	84206.49
SLOC-L	0	2361	106.48	36	207.36	42997.09
MVG	0	455	24.04	5	52.20	2724.56
BLOC	0	1772	117.08	57	177.23	31408.92
C&SLOC	0	0	0	0	0	0

lowest for the Mapdb dataset amongst all the three datasets which suggests that it is easy to maintain it as compared to Antlr4 and Mct datasets. Cohesion is a measure of how well the lines of source code within a module work together. The mean value of LCOM is 2387.765 in Mapdb dataset, which is undesirable as high LCOM value implies that the classes are less cohesive and thus, are not efficiently encapsulated.

On the other hand, McMMO and Mct have lower values of LCOM of 117.55 and 111.87, respectively which implies comparatively good cohesion. Coupling is a measure of interaction between two classes indicated by the Ca, Ce, CBO, IC and CBM metrics. The three datasets Mapdb, McMMO and Mct have very low values of 0.09, 0.10 and 0.03, respectively, which suggests that the new methods are not coupled with the inherited methods, thus making it a good software design. The median of NOC for the Mapdb and McMMO datasets is 0 which indicates that 50% or more classes do not have child classes. Table 4 gives the descriptive statistics of the Mapdb dataset.

Low values of NOC are undesirable since it does not promote reusability which is one of the key features of object-oriented systems. Since the mean value of NOC is highest for the Mct dataset with a value of 0.52, it indicates that the code has been reused in it. It is preferred to have minimum NPM as public as it means that the software can be accessed by everyone which may lead to security issues. It can be observed that the mean value of NPM is the highest for Mapdb dataset with a value of 23.53769 and the lowest for the Mct dataset with a value of 7.8 among the three-open source software datasets.

Table 5 depicts the abrupt change in the behavior of the variances where LOC has the highest value of 199250.26, whereas MVG has a recorded variance of 2724.56, which implies comparatively good coupling between the

TABLE 6. Descriptive statistics of the junit dataset.

Metric	Minimum	Maximum	Mean	Median	Standard Deviation	Variance
LOC	1	3120	113.25	67	166.44	27703.33
SLOC-P	1	1788	71.63	37	109.35	11957.01
SLOC-L	0	1253	49.03	25	74.81	5596.81
MVG	0	301	10.54	4	21.18	448.44
BLOC	0	1670	41.39	29	63.39	4017.72
C&SLOC	0	0	0	0	0	0
CLOC	0	93	0.23	0	2.79	7.79
CWORD	0	441	0.65	0	11.87	140.86
HCLOC	0	0	0	0	0	0
HCWORD	0	0	0	0	0	0
WMC	0	198	9.50	5	13.66	186.56
DIT	0	4	0.57	1	0.63	0.40
NOC	0	71	0.52	0	3.02	9.11
CBO	0	531	11.75	7	25.24	636.98
RFC	0	444	23.13	14	30.23	913.68
LCOM	0	19407	111.87	4	766.46	587454.72
Ca	0	529	5.83	2	24.05	578.46
Ce	0	108	6.42	4	7.79	60.76
NPM	0	184	7.87	4	12.30	151.41
LCOM3	0	2	1.12	0.85	0.72	0.51
LCO	0	4921	134	59	239.47	57343.67
DAM	0	1	0.60	1	0.48	0.23
MOA	0	26	0.66	0	1.47	2.16
MFA	0	1	0.03	0	0.15	0.02
CAM	0	1	0.49	0.44	0.27	0.07
IC	0	2	0.02	0	0.14	0.02
CBM	0	7	0.03	0	0.26	0.07
AMC	0	158.67	11.26	8.30	12.51	156.53
NCHANGE	0	100	38.25	36.44	5.85	34.18

various classes. Here coupling refers to the measure of interaction between two classes.

Table 6 depicts the abrupt change in the behavior of variances for the Junit database. The simple Junit test operations in Table 5 yields a very narrow range of variations in the variances. This is because of simply "Junit test dropping" and recreating the database between two operations is done through the code where each instances of a class varies as soon as an operation occurs. Secondly, data fetched from the database makes the instance forcibly change its behavior because it assumes the same class, but in reality the class belongs to some other projects too. Hence, there is an abrupt change in behavior. Coupling and cohesion helps in segregating two or more classes but we can see narrow variations in this case in comparison to the values in Table 4.

B. DEEP LEARNING

In the Deep Learning method, we have taken three layers of long short-term memory (LSTM) deep neural network and one dense layer for getting the cumulative output. We have divided it into two classes namely Defect or Not-Defect. We chose to use the Recurrent Neural Network as it does not only predict on the basis of current metrics but is also able to use the metrics in previous states so that we can predict how the inter-combination of metrics affects each other. Except calculating the hidden layer output, all the high-level operations of LSTM are the same as those in RNN. In the training phase, a sequence of inputs is represented by Eq. (1):

$$X = \{x_{1,}x_{2,} \ldots \ldots \ldots x_T\} \tag{1}$$

The above equation depicts different input nodes i.e. x_1, x_2, ... which represent various metrics that we have considered, and X_T is the input of the current time step. In short, this equation gives the input nodes to the neural network. From the given sequence above, we feed an input x_t to the hidden layers and calculate the hidden layer activations using Eq. (1):

$$h_t = \{h_{t,}^0 h_{t,}^1 \ldots \ldots \ldots h_{t,}^N\} \tag{2}$$

In Eq. (1), h_t is the activation function which is coupled with the input nodes as explained in Eq. (1). Once h_t is acquired, we calculate the output y_t and loss L_t in the prediction. Then, we back propagate "the loss" through the network.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{3}$$

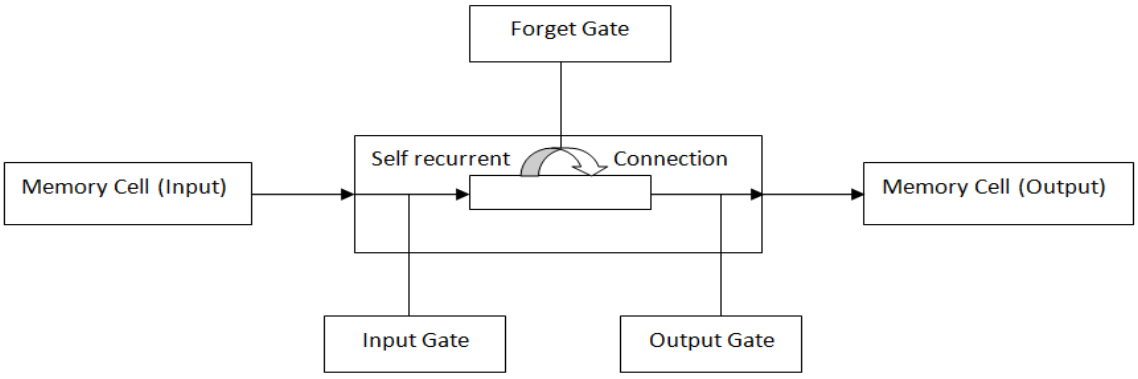

FIGURE 1. The forget gate.

Here, i_t is the sigmoid function which holds W_{xi} which is a set of random weights and is coupled with $x_{t,}$, which is the input of the current time step. The term h_{t-1} is the output of the previous state i.e. h_t along with b_i which is i^{th} bias of the couple. In short, Eq. (3) takes the input to the hidden layer which is a function of the previous state or the sequence.

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{4}$$

After getting sigmoid function, the f_t which is a forget gate is calculated to control the self-loop.

$$c_t = f_t c_{t-1} + \mathrm{i}_{\tanh}(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{5}$$

The *forget gate* is also sigmoid in nature which looks at x_t and x_{t-1} and gives output between 0 and 1 and for each number in the cell state between c_{t-1}. Here '1' will completely represent '*keep this*(DO NOT STORE)', whereas '0' completely represents '*reject this*(STORE)'. Since our objective is to extract out the valuable parameters that are actually affecting the prediction, we used the tangent hyperbolic function (*tanh*). Here, *tanh*takes data of previous h_{t-1} and the current input x_t, multiplied by the corresponding weights and added by the bias. The tangent hyperbolic function is used in order to achieve decisions from the characteristics curves in general. Subsequently, i_t as explained in Eq. (3) decides the sigmoid layer. This layer decides which prediction value to be stored and which not to be stored. Finally, c_t creates a vector derived from the values which was decided by i_t.

At this stage (Fig. 1), we come to the state of finding out the value of c_t which is the vector that goes on rejecting the c_{t-1} and thus adding the new values i.e. i_t and c_t. Eq. (6) represents the internal state of a LSTM cell where the conditional self -loop gate is included.

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{6}$$

Firstly, we run a sigmoid layer o_t which decides what parts of the cell state being outputs. Then we put the cell state through *tanh* (to push the values to be between -1 to 1 and multiply it by the output of the sigmoid gate. It should be noted that Eq. (6) is the *output of the external input gate.*

$$h_t = o_t \tanh(c_t) \tag{7}$$

In Eq. (7), we put the cell state through *tanh*, and the output of Eq. (7) is the *output of the hidden layers.*

$$y_t = Softmax\left(W_{hN_y}h_t^N + b_h^N\right) \tag{8}$$

Eq. (8) is the output of the dense layer and is the classification output. All the outputs from the hidden layers are combined in a dense fully connected layer. The binary output that is obtained will determine if it is defective or not. The proposed LSTM algorithm is summarized in Table 7.

C. EVALUATION

The accuracy measures considered in this research for evaluating error in prediction of 'NChange' are described as follows:

(i) *Pearson's Correlation Coefficient:* The correlation coefficient (r) indicates the degree of association and is commonly referred to as Pearson's correlation. The values of r may lie in the interval $[-1, 1]$. Values between $[-1, 0)$ indicate a negative correlation, values between $(0, 1]$ indicate a positive correlation, whereas a value of 0 indicates the absence of any correlation. The formula for the correlation coefficient is given in Eq. (9).

$$r = \frac{\sum (P_j - P)(A_j - \overline{A})}{\sqrt[2]{\sum (P_j - \overline{P})^2 (A_j - \overline{A})^2}} \tag{9}$$

where P_j and A_j denotes the predicted value and the actual value, respectively.

(ii) *Mean Absolute Error (MAE):* The MAE is a measure that calculates the closeness between the predicted value and the eventual outcomes, and is defined as given in Eq. (10).

$$MAE = \frac{1}{N}\sum_{j=1}^{N} |P_j - A_j| \tag{10}$$

where N is the total number of instances.

(iii) *Root Mean Square Error (RMSE):* The RMSE indicates the spread of data around the regression line i.e. the

TABLE 7. The proposed LSTM algorithm.

Input: $X= \{X_1, X_2, \ldots\ldots, X_t\}$, W_t, b_t, h_t, x_t, b_f ***Output:*** *Classify into Defect and Not Defect (Change and No Change)*	
1	$X= \{X_1, X_2, \ldots\ldots, X_t\}$ // *Enter the metric data under consideration*
2	*The data under consideration are computed as* $((W_t * X_t) + b_t)$ *and becomes the input to the first hidden layer*
3	***Internal process****:* f_t= *Sigmoid* $(W_f * [h_t, x_t] + b_f)$ // *In the hidden layer the forget gate is applied to forget the metrics that have been stored and are not required.* i_t = *Sigmoid* $(W_i * [h_{t-1}, x_t] + b_i)$ // *Decide the metrics that are to be added to the memory* $C_t^{'}$ = *tanh* $(W_f * [h_t, x_t] + b_f)$ '// *The decisions taken by* i_t *are projected converted into a vector values* $C_t = C_{t-1} * f_t + i_t * C'$ // *In the new hidden layer, the memory is updated*
4	***External process****:* O_t= *Sigmoid* $(W_o * [h_{t-1}, x_t] + b_o)$ $H_t = O_t * \tanh(C_t)$
5	*Apply steps 3 and 4 to 3 hidden layers*
6	*Apply steps 3 and 4 to the fully connected dense layer*
7	*Use the softmax function for the output of the dense layer*
8	*Classify the achieved results into Defect or Not-Defect*
9	*Back propagation*
10	*End*

standard deviation of errors, and is defined by Eq. (11).

$$RMSE = \sqrt[2]{\frac{1}{N}\sum_{j=1}^{N}\left(P_j - A_j\right)^2} \tag{11}$$

(iv) *Accuracy:* This function is an indication of how close the measured value is to the true value, and is defined in Eq. (12).

$$Accuracy = \left(\frac{1}{N}\sum_{j=1}^{N}\left|P_j - A_j\right|\right) * 100 \tag{12}$$

where $\left|P_j - A_j\right| \leq 1$.

a. **The novelty of the proposed approach**

Most of the previous works that have been mentioned in this paper has referred to Software Maintainability as ''the ability to modify''. However, in our methodology, we have defined Software Maintainability in terms of the following criteria/features:

(i) The method to detect and rectify the faults of any Software Development Life Cycle (SDLC). Corrective maintenance of software is only restricted to discovering defects. It also takes into account reactive modification of software product after the defect has been discovered [55]. This is done so as to ensure that software is still usable in a changed or changing environment. The idea is to reduce the software corrective maintenance effort. Therefore proper enforcement of standards during each phase of SDLC is mandatory.

(ii) The performance can be improved after analyzing the precision and accuracy. Evaluation metrics like precision, accuracy, Area under Curve etc. have been popularly associated with machine learning (Kaur *et al*, 2014). Since our study takes it a step further to Deep Learning, which is concerned with better accuracy to perform exceptionally well, it is important that the metrics exhibit good results. Improvising precision and accuracy is one way of enhancing the performance of software maintainability.

(iii) Various attributes can be well adapted if a particular SDLC is converted into a new environment. In SDLC, as a software progresses into different phases one after the other, the environment may keep on changing. With the changing environment, there may be new attributes and they may have issues with adaptability due to faults in various stages. Software maintenance discovers and corrects faults, which may evade this issue (Asghar *et al*, 2011).

(iv) No variations in the result even though data sets are migrated from one to the other domain. Software maintainability resolves conflicts across various platforms. With defects being identified and corrected, data sets across different domains become more compatible.

On the other hand, maintainability also depends on how the software is being used. This paper well describes the parameters such as programming languages, frameworks, coding rules, and design patterns. As the majority of the time is spent in the maintenance phase, efforts spent in increasing its maintainability will affect the cost of the software in a negative or positive manner. It is obvious to state that if the software is customizable and maintainable, it will be very much easier for the developers to modify the bugged module at any time within a short period of time; thus enabling

TABLE 8. Metrics subset obtained by FSS algorithms for the mapdb dataset.

Linear Correlation	Rank Correlation	OneR	Relief	Consistency
SLOC-P	LCO	CBM	WMC	LOC
SLOC-L	DAM	CLOC	CBO	SLOC-P
LOC	AFC	CWORD	NPM	SLOC-L
RFC	RFC	MFA	SLOC-P	MVG
LCO	LCOM3	MOA	NOC	WMC
MVG	DIT	CBO	Ca	DIT
Ce	Ca	DAM	LCOM	CBO
WMC	WMC	DIT	MFA	RFC
NPM	BLOC	Ce	IC	LCOM
CBO	NPM	WMC	CBM	Ca
LCOM	CAM	Ca	DAM	Ce
BLOC	Ce	SLOC-L	LOC	NPM
CAM	MOA	NPM	MOA	LCOM3
DAM	SLOC-L	SLOC-P	LOC	LCO
CLOC	CWORD	LCOM	SCOC-L	DAM

TABLE 9. Metrics subset obtained by FSS algorithms for the Antlr4 dataset.

Linear Correlation	Rank Correlation	OneR	Relief	Consistency
RFC	LCO	DIT	Ce	WMC
LOC	RFC	NOC	WMC	CBO
SLOC-L	MVG	IC	NPM	RFC
BLOC	SLOC-P	CBM	DAM	LCOM
SLOC-P	SLOC-L	MFA	CBO	Ca
LCO	LOC	Ce	CAM	Ce
MVG	CAM	SLOC-P	LCO	NPM
WMC	AMC	MOA	Ca	LCOM3
LCOM	Ca	DAM	LCOM3	LCO
NPM	BLOC	LOC	BLOC	DAM
Ce	Ce	LCOM	RFC	MOA
DAM	CBO	CBO	MOA	CAM
CAM	WMC	SLOC-L	MVG	AMC
CBO	MOA	BLOC	LCOM	LOC
LCOM3	DIT	SLOC-P	SLOC-L	SLOC-P

more efficient software management. The previous related works indicate that the maintenance effort in most of the SDLC ranges from 65% to 75% of total software development time. In our proposed method and discussion, we have clearly proven that the effort reduces to 45% to 50% i.e. the maintainability feature of a software increases the quality of it.

How does software maintainability prediction help in the maintenance process and what would be the appropriate associated metrics to be used for software maintainability prediction?

Software maintainability predicts changes or failures that may occur in software after it has been deployed. Therefore, it is necessary to predict software maintainability in quantitative terms as early as the design phase. It also helps in identifying those areas that may be the cause of poor maintenance. Software maintenance is mainly dependent on failure rate and repair time of the software. The goal of the software maintainability process is to determine the amount of time required for repairing system and maintaining tasks, which may lead to improved performance of the software. Moreover, early assessment of characteristics maintainability in a system design progresses with different phases of the software. Since tracking the maintenance behavior of software is quite difficult, predicting cost and risk associated becomes hard. Therefore it is important that

TABLE 10. Metrics subset obtained by FSS algorithms for the junit dataset.

Linear Correlation	Rank Correlation	OneR	Relief	Consistency
RFC	SLOC-L	NOC	WMC	WMC
LCO	SLOC-P	Ca	NPM	DIT
LOC	LCO	MFA	DIT	CBO
SLOC-L	LOC	IC	BLOC	RFC
Ce	MVG	CBM	MOA	LCOM
SLOC-P	RFC	CBO	LOC	Ce
MVG	BLOC	DAM	NOC	NPM
BLOC	WMC	MOA	LCOM3	LCOM3
WMC	AMC	DIT	MVG	LCO
NPM	NPM	NPM	RFC	DAM
AMC	CAM	WMC	SLOC-P	MOA
CAM	LCOM	LOC	LCOM	CAM
LCOM	Ce	Ce	Ca	AMC
DAM	CBO	MVG	Ce	LOC
CBO	MOA	SLOC-L	SCOL-P	SLOC-P

software maintainability is predicted and evaluated. This may be done using machine learning algorithms. With the help of the deep learning the process of software maintainability is eased significantly. Early prediction of faults is quite useful in improving software quality. Deep learning also gives a high level of efficiency, which further plays an important role in the maintenance of the software.

Maintainability may also take into account addition of new features or functionalities or modifying the software so as to meet new requirements. Software maintenance is concerned with sustaining the software product throughout its life cycle [56]. It is required when customer demands new features and new functions in the software. It may be also required when hardware of the system undergoes changes. Since software maintenance deals with modification of software product after delivery so as to correct faults, or to improve performance, we may conclude that sustenance of software depends on its flawlessness, which may be achieved once the software is defect free. Software maintainability prediction ensures defect free software's and thus leads to software maintenance. The maintenance phase in SDLC usually is responsible for identifying defects and fixing errors in software. The data may be provided from either the customers or testers who may identify defects or issues with the software. Once the data is with the developers, they can work on fixing the errors to ensure defect free software. Some of the metrics for Software Maintainability Prediction are recall, precision and Area under curve (AUC) of ROC curve [18].

IV. RESULTS AND DISCUSSION

Firstly, we identify the set of metrics that can best predict the value of NChange to enhance the structural quality of the software. Tables 8-10 give the set of the most significant metrics obtained by applying five Feature Subset Selection (FSS) algorithms (Linear Correlation, Rank Correlation, OneR, Relief and Consistency) for the Mapdb, McMMO and Mct datasets.

According to Linear Correlation, SLOC-P, SLOC-L, LOC, RFC, LCO, MVG, Ce, WMC, NPM, CBO, LCOM, BLOC, CAM, DAM and CLOC are the most significant metrics for Mapdb, RFC, LOC, SLOC-L, BLOC, SLOC-P, LCO, MVG, WMC, LCOM, NPM, Ce, DAM, CAM, CBO and LCOM3 for Antlr4. After applying Linear Correlation for Junit RFC, LCO, LOC, SLOC-L, Ce, SLOC-P, MVG, BLOC, WMC, NPM, AMC, CAM and LCOM are found to be more significant than others.

Rank Correlation gives LCO, DAM, AMC, RFC, LCOM3, DIT, Ca, WMC, BLOC, NPM, CAM, Ce, MOA and SLOC-L as the most relevant metrics for Mapdb, whereas LCO, RFC, MVG, SLOC-P, SLOC-L, LOC,CAM, AMC, Ca, BLOC, Ce,

TABLE 11. Comparison (red values denote the results of the proposed method).

ML Algorithm	FSS Algorithm	Mapdb				McMMO				Junit			
		r	MAE	RMSE	Acc.	r	MAE	RMSE	Acc.	r	MAE	RMSE	Acc.
Ridge Regression with Variable Selection	Original Dataset	0.50	1.59	5.56	66.75	0.15	1.98	5.22	69.74	0.20	1.96	4.05	70.02
	Linear Correlation	0.55	1.35	3.94	72.45	0.33	1.64	4.63	73.76	0.37	1.54	3.08	72.10
	Rank Correlation	0.58	1.22	3.48	73.67	0.21	1.54	4.28	76.83	0.31	1.54	3.73	72.10
	OneR	0.56	1.33	3.16	69.10	0.20	1.53	3.76	75.89	0.35	1.63	3.59	74.52
	Relief	0.63	1.30	3.62	69.86	0.41	1.36	3.02	77.81	0.42	1.50	3.10	75.14
	Consistency	0.60	1.32	3.12	75.36	0.36	1.45	3.85	76.36	0.32	1.76	3.85	69.32
Decision Tree	Original Dataset	0.48	1.53	4.41	82.65	0.28	1.94	5.04	70.21	0.27	1.93	4.57	70.88
	Linear Correlation	0.61	1.24	3.40	82.95	0.44	1.40	3.61	77.78	0.45	1.23	2.64	76.08
	Rank Correlation	0.60	1.20	3.30	84.47	0.42	1.54	3.74	77.54	0.35	1.40	3.08	75.74
	OneR	0.55	1.27	3.64	84.63	0.56	1.61	3.73	77.78	0.39	1.54	3.10	75.04
	Relief	0.54	1.41	4.05	85.93	0.49	1.20	3.68	79.20	0.40	1.42	2.72	75.91
	Consistency	0.65	1.23	2.84	87.93	0.58	1.16	3.40	82.38	0.42	1.23	2.46	78.08
Quantile Regression Forest	Original Dataset	0.69	1.00	5.67	88.89	0.13	1.22	3.85	86.76	-0.01	1.36	4.28	77.30
	Linear Correlation	0.70	0.99	5.65	88.58	0.17	1.16	4.16	88.42	0.01	1.42	4.30	77.12
	Rank Correlation	0.69	0.99	5.65	88.89	0.30	1.07	3.28	87.94	0.02	1.37	4.26	77.82
	OneR	0.69	1.01	5.66	88.74	0.23	1.21	3.37	86.05	0.03	1.20	3.96	79.68
	Relief	0.69	1.03	5.65	88.13	0.19	1.12	3.74	88.18	0.01	1.40	4.32	77.99
	Consistency	0.70	0.99	5.63	89.74	0.31	1.04	3.25	88.99	0.02	1.37	4.24	78.64
Support Vector Machine	Original Dataset	0.43	1.29	5.11	80.37	0.34	1.18	4.56	82.98	0.23	1.67	4.89	74.35
	Linear Correlation	0.61	1.17	4.17	85.54	0.65	0.79	2.82	87.00	0.38	1.16	3.45	78.34
	Rank Correlation	0.62	1.11	4.19	86.91	0.69	0.72	2.78	85.34	0.35	1.12	3.74	77.12
	OneR	0.69	1.08	3.00	87.96	0.70	0.73	2.29	88.42	0.42	1.05	3.45	79.65
	Relief	0.60	1.17	3.96	83.71	0.60	0.89	3.13	88.18	0.31	1.17	3.84	78.51
	Consistency	0.63	1.09	3.62	85.84	0.48	1.01	3.89	86.05	0.32	1.09	3.43	79.03
Principal Component Analysis	Original Dataset	0.27	1.43	4.70	73.06	0.23	1.92	4.98	70.21	0.32	1.78	4.29	71.23
	Linear Correlation	0.60	1.32	3.48	76.56	0.35	1.48	4.00	78.25	0.41	1.41	2.82	74.35
	Rank Correlation	0.56	1.32	3.32	76.86	0.28	1.70	4.69	75.65	0.38	1.37	2.58	72.96
	OneR	0.45	1.36	3.38	68.04	0.38	1.37	3.30	78.96	0.38	1.41	3.03	74.35
	Relief	0.43	1.32	3.49	68.34	0.33	1.33	3.49	78.01	0.42	1.39	2.80	76.70
	Consistency	0.58	1.31	3.10	79.71	0.29	1.29	3.09	79.07	0.43	1.30	2.22	75.39
LSTM	Original Dataset	0.68	1.3	5.70	90.06	0.35	1.92	4.98	93.21	0.42	1.68	3.29	89.23
	Linear Correlation	0.60	1.33	5.48	92.56	0.38	1.48	4.00	92.25	0.45	1.51	3.82	92.35
	Rank Correlation	0.56	1.32	5.89	91.86	0.32	1.80	4.69	91.65	0.48	1.47	3.58	90.96
	OneR	0.45	1.36	5.87	92.04	0.31	1.77	4.30	90.96	0.46	1.51	3.33	93.35
	Relief	0.43	1.38	5.56	91.34	0.35	1.83	4.49	93.01	0.41	1.49	3.80	92.70
	Consistency	0.58	1.36	5.76	94.00	0.37	1.89	4.09	94.07	0.45	1.40	3.22	93.76

CBO, WMC, MOA and DIT are the most relevant metrics for Antlr4, while SLOC-L, SLOC-P, LCO, LOC, MVG, RFC, BLOC, WMC, AMC, NPM, CAM, LCOM, Ce, CBO and MOA are determined to be the most relevant metrics for

TABLE 12. Comparative analysis.

No.	Authors	Advantages	Limitations
1	Cheng et.al. [46]	Presented a system for indicating, preparing, assessing and conveying machine learning methods	Suggested model utilizing heterogeneous datasets.
2	Nam et.al. [47]	Proposed heterogeneous defect prediction (HDP) to anticipate deserts crosswise over activities with heterogeneous metric sets.	Need to investigate the attainability of building different forecast.
3	Tavakoli et.al. [28]	Exhibited the handiness of scene closeness in anticipating the saliency roused by the impact of the nature of a scene on the spectator's eye developments.	Approval technique on the aftereffects of proposed consider.
4	Singh and Chug [48]	Performed detailed analysis & showed those Linear classifiers are the most accurate & reliable among defect prediction methods.	Results can be further refined by using more number of datasets and comparison can be done amongst more number of methods.
5	Li et.al. [17]	Proposed CCLEARNER, a deep learning-based clone detection method.	Need to try different methods with other machine learning systems to investigate whether deep learning is the best strategy to prepare a classifier for clone identification.
6	Nucci et.al. [58]	Exhibited a recreated investigation.	Need to address the issues in assessing the effect of dataset estimate
7	Liu et.al. [42]	Proposed to utilize the Historical Version Sequence of Metrics (HVSM) in constant programming renditions as imperfection indicators.	Need to apply distinctive procedures to HVSM or utilizing the data given by HVSM to enhance the execution of run of the mill strategies.
8	Chen et.al. [52]	Formalized JIT-SDP as a multi-target optimization issue and afterward propose a novel technique MULTI	Need to optimize the performance of proposed method by considering other MOAs
9	Miholca et.al. [53]	Proposed an advanced supervised learning method called HyGRAR to take care of the software deformity forecast issue.	Need to extend the HyGRAR show by considering other machine learning strategies for slow relations, for example, SVM.
10	Krishna and Menzies [54]	Undertook a detailed study of transfer learners called the “bellwethers”.	Lacking feature selection in choosing the relevant metrics for the analysis.
11.	**Proposed Work**	Proposed a model for software maintainability metrics prediction. A new measure of precision is introduced. The proposed Deep Learning model outperforms all of the other Machine Learning Techniques that were considered.	More datasets need to be used. A new measure of precision is introduced. More SDLC need to be considered.

the Junit dataset. Similarly, CBM, CLOC, CWORD, MFA, MOA, CBO, DAM, DIT, Ce, WMC, Ca, SLOC-L, NPM, SLOC-P and LCOM are found to be the more important metrics by OneR in the case of Mapdb dataset, whereas DIT, NOC, IC, CBM, MFA, Ce, SLOC-P, MOA, DAM, LOC, LCOM, CBO and SLOC-L are the most important metrics for the Antlr4 database, whereas NOC, Ca, MFA, IC, CBM, CBO, DAM, MOA, DIT, NPM, WMC, LOC and Ce are the most important metrics in the case of the Junit dataset.

The key metrics obtained by Relief for the Mapdb dataset are WMC, CBO, NPM, SLOC-P, NOC, Ca, LCOM, MFA, IC, CBM, DAM, LOC and MOA, the key metrics for the AntIr4 dataset are WMC, NPM, DAM, CBO, CAM, LCO, Ca, LCOM3, BLOC, RFC, MOA, MVG, LCOM and SLOC-L, whereas the key metrics for the Junit dataset are WMC, NPM, DIT, BLOC, MOA, LOC, NOC, LCOM3, MVG, RFC, SLOC-P, LCOM, Ca and Ce. According to Consistency, the most important metrics for the Mapdb dataset are the LOC, SLOC-P, SLOC-L, MVG, WMC, DIT, CBO, RFC, LCOM, Ca, Ce, NPM, LCOM3, LCO and DAM, and the most important metrics for the McMMO dataset are the WMC, CBO, RFC, LCOM, Ca, Ce, NPM, LCOM3, LCO, DAM, MOA, CAM, AMC, LOC, SLOC-P and MVG, whereas DIT, CBO, RFC, LCOM, Ce, NPM, LCOM3, LCO, DAM, MOA, CAM, AMC, LOC, SLOC-P and SLOC-L are the most essential metrics for the Mct dataset.

Secondly, we identify the most efficient algorithm on the basis of different prediction accuracy parameters. Five machine learning algorithms namely Ridge Regression with Variable Selection, Decision Tree, Quantile Regression Forest, Support Vector Machine and Principal Component Analysis were applied on the original datasets and those obtained by the five selected FSS algorithms. The evaluation results of different prediction accuracy measures consisting of values of r (correlation coefficient), MAE, RMSE and Accuracy of the original datasets (without any feature reduction), and the best combination of FSS and machine learning algorithms for the three datasets are presented in Table 11.

For the three open source datasets, the LSTM algorithm gave the best results with an accuracy of 94%. The FSS algorithm consistency gave the best results when applied on the LSTM deep neural network. The correlation coefficient r of the data had a range from 0.45 to 0.68, and the MAE was considerably less for the datasets on which it was applied. The RMSE was high for the Mapdb dataset but for the MCMMO and Junit databases the RMSE was comparably lower. The LSTM algorithm when applied on the metrics selected by FSS algorithm Consistency gave a considerably good result.

In all the cases, we can observe that the machine learning algorithms and the LSTM perform better after the FSS algorithms have been applied on the dataset. This implies that the FSS algorithms can help to filter out the relevant metrics that are required to increase the accuracy of the prediction. Among the machine learning algorithms, the quantile regression performs best on the datasets giving an average accuracy of 89%.

V. LIMITATIONS OF THE RELATED WORK IN COMPARISON TO OUR PROPOSED WORK

Following are some of the relevant limitations of the related works in comparison to our proposed work:

1. There is a lack of proper measure of precision. We thought of applying the Neural System because in common classes in several open source projects, simple neural systems have been used as models of central variations generation. Such insatiable behavior has been investigated in the SDLC (Software Development Life Cycles), and presented in this paper. As in the isolated sequence case, it is desirable to use the dynamical behavior of each of those classes in these more complex sequence problems to measure the precision that we have achieved in this paper in future. There have been many recent works related to software maintainability metrics prediction, but none of these works have provided any measure to determine the accuracy of the precision [11]. Here we have proposed a forget layer which helps in enhancing the precision measure as opposed to the previous published works in this area that have not done proper precision measurements.
2. In order to increase the precision, and decrease the complexity, this paper deals with multi layers of LSTM networks. In the previous works [11], concurrency and recurrence were taken into consideration when dealing with huge datasets, and the different software cycles i.e. the SDLC of different platforms were not taken into consideration when dealing with huge datasets. So, in order to get a higher precision level, we intend to use 299 projects, and the results have thus far indicated that we have been successful in producing results with a higher level of precision accuracy compared to the previous works [57].
3. As shown in Table 11, the proposed LSTM model gives a very high level of accuracy compared to all of the other methods. This further verifies our initial claims that the deep learning model gives better prediction results compared to other conventional machine learning algorithms such as Decision Tree, Support Vector Machine.

VI. COMPARATIVE STUDY

In this section, we present a comparative analysis of our proposed work and existing works in the literature. This is summarized in Table 12.

In this study, we have proposed a model for software maintainability metrics prediction. Moreover, we have introduced a new measure of precision. It has been found experimentally that the proposed Deep Learning model outperforms all of the other Machine Learning Techniques that were considered. With more datasets we may achieve better results. We may also consider more SDLC for the same.

VII. CONCLUSION

This paper addresses the research gaps in the study of software maintainability metrics prediction. From the previously published works and the experiments, it is clear that our proposed LSTM algorithm outperforms the other machine learning algorithms. It can likewise be seen that in most of the other works, machine learning calculations neglect to indicate which measurements influence the product viability, whereas our LSTM algorithms measures this accurately. In this paper, the measurements of twenty-nine projects have been considered and their effect on programming practicality of open source programming has been studied. On applying the proposed algorithms on the chosen datasets, we found that the LSTM algorithm should be used for the optimal prediction of software maintenance. Furthermore, we found that the FSS algorithm gives the best results in determining the most relevant metrics to be considered for prediction.

Future work in this area include using a fluffy profound neural system, and studying if the precision of the results obtained via this method is comparable to results obtained via other methods. We could also apply a blend of CNN (Concurrent Neural Network) profound net, followed by layers of LSTM profound net. Furthermore, we can also study the impact of taking the recurrence of a model into consideration on the imperfections and the timing of these imperfections, as well as on the frequency and seriousness of the deformity.

REFERENCES

[1] R. Malhotra and A. Chug, "Software maintainability: Systematic literature review and current trends," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 8, pp. 1221–1253, 2016.
[2] Q. Huang, E. Shihab, X. Xia, D. Lo, and S. Li, "Identifying self-admitted technical debt in open source projects using text mining," *Empirical Softw. Eng.*, vol. 23, no. 1, pp. 418–451, 2018.
[3] J. Guo *et al.*, "Data-efficient performance learning for configurable systems," *Empirical Softw. Eng.*, vol. 23, no. 3, pp. 1826–1867, 2018.
[4] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer, "A survey on search-based model-driven engineering," *Automated Softw. Eng.*, vol. 24, no. 2, pp. 233–294, 2017.
[5] I. Gondra, "Applying machine learning to software fault-proneness prediction," *J. Syst. Softw.*, vol. 81, pp. 186–195, Feb. 2008.

[6] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, vol. 1, 2016, pp. 2073–2083.

[7] S. Mishra and A. Sharma, "Maintainability prediction of object oriented software by using adaptive network based fuzzy system technique," *Int. J. Comput. Appl.*, vol. 119, no. 9, pp. 24–27, 2015.

[8] M. A. Ahmed and H. A. Al-Jamimi, "Machine learning approaches for predicting software maintainability: A fuzzy-based transparent model," *IET Softw.*, vol. 7, no. 6, pp. 317–326, 2013.

[9] M. R. H. A. Mohd, A. Sarkheyli, A. M. Zain, and H. Haron, "Fuzzy logic for modeling machining process: A review," *Artif. Intell. Rev.*, vol. 43, no. 3, pp. 345–379, 2015.

[10] Amarjeet and J. K. Chhabra, "Improving package structure of object-oriented software using multi-objective optimization and weighted class connections," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 29, no. 3, pp. 349–364, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157815001093#!

[11] M. O. Elish, H. Aljamaan, and I. Ahmad, "Three empirical studies on predicting software maintainability using ensemble methods," *Soft Comput.*, vol. 19, no. 9, pp. 2511–2524, 2015.

[12] J. Finlay, R. Pears, and A. M. Connor, "Data stream mining for predicting software build outcomes using source code metrics," *Inf. Softw. Technol.*, vol. 56, no. 2, pp. 183–198, 2014.

[13] R. Francese, M. Risi, G. Scanniello, and G. Tortora, "Proposing and assessing a software visualization approach based on polymetric views," *J. Vis. Lang. Comput.*, vol. 34, pp. 11–24, Jun. 2016.

[14] X. Huo, M. Li, and Z. H. Zhou, "Learning unified features from natural and programming languages for locating buggy source code," in *Proc. 25th Int. Conf. Artif. Intell. (IJCAI)*, 2016, pp. 1606–1612.

[15] T. Haije, B. O. K. Intelligentie, E. Gavves, and H. Heuer, "Automatic comment generation using a neural translation model," *Inf. Softw. Technol.*, vol. 55, no. 3, pp. 258–268, 2016.

[16] V. -S. Ionescu, H. Demian, and I. G. Czibula, "Natural language processing and machine learning methods for software development effort estimation," *Stud. Inform. Control*, vol. 26, no. 2, pp. 219–228, 2017.

[17] P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2017, pp. 497–508.

[18] A. Kaur and K. Kaur, "Statistical comparison of modelling methods for software maintainability prediction," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 23, no. 6, pp. 743–774, 2013.

[19] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder, "CCLearner: A deep learning-based clone detection approach," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 249–260.

[20] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Inf. Softw. Technol.*, vol. 58, pp. 388–402, Feb. 2015.

[21] M.-J. Lin, C.-Z. Yang, C.-Y. Lee, and C.-C. Chen, "Enhancements for duplication detection in bug reports with manifold correlation features," *J. Syst. Softw.*, vol. 121, pp. 223–233, Nov. 2016.

[22] Q. Luo, A. Nair, M. Grechanik, and D. Poshyvanyk, "FOREPOST: Finding performance problems automatically with feedback-directed learning software testing," *Empiricals Softw. Eng.*, vol. 22, no. 1, pp. 6–56, 2017.

[23] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[24] R. Malhotra and M. Khanna, "An exploratory study for software change prediction in object-oriented systems using hybridized techniques," *Automated Softw. Eng.*, vol. 24, no. 3, pp. 673–717, 2017.

[25] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput.*, vol. 27, pp. 504–518, Feb. 2015.

[26] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Inf. Softw. Technol.*, vol. 55, no. 8, pp. 1397–1418, 2013.

[27] S. Rahimi and M. Zargham, "Vulnerability scrying method for software vulnerability discovery prediction without a vulnerability database," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 395–407, Jun. 2013.

[28] H. R. Tavakoli, A. Borji, J. Laaksonen, and E. Laaksonen, "Exploiting inter-image similarity and ensemble of extreme learners for fixation prediction using deep features," *Neurocomputing*, vol. 244, pp. 10–18, Jun. 2017.

[29] X. Yang, D. Lo, X. Xia, and J. Sun, "TLEL: A two-layer ensemble learning approach for just-in-time defect prediction," *Inf. Softw. Technol.*, vol. 87, pp. 206–220, Jul. 2017.

[30] O. Araque, I. Corcuera-Platas, J. F. Sánchez-Rada, and C. A. Iglesias, "Enhancing deep learning sentiment analysis with ensemble techniques in social applications," *Expert Syst. Appl.*, vol. 77, pp. 236–246, Jul. 2017.

[31] C.-M. Wang and Y.-F. Huang, "Self-adaptive harmony search algorithm for optimization," *Expert Syst. Appl.*, vol. 37, no. 3, pp. 2826–2837, 2010.

[32] A. Chug and R. Malhotra, "Benchmarking framework for maintainability prediction of open source software using object oriented metrics," *Int. J. Innov. Comput., Inf. Control*, vol. 12, no. 2, pp. 615–634 , 2016.

[33] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Syst. Appl.*, vol. 83, pp. 187–205, Oct. 2017.

[34] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," *Appl. Soft Comput.*, vol. 21, pp. 286–297, Aug. 2014.

[35] F. De Smedt, L. Struyf, S. Beckers, J. Vennekens, G. De Samblanx, and T. Goedemé, "Faster and more intelligent object detection by combining OpenCL and KR," *J. Ambient Intell. Humanized Comput.*, vol. 5, no. 5, pp. 635–643, 2014.

[36] M. Staron, W. Meding, and B. Söderqvist, "A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation," *Inf. Softw. Technol.*, vol. 52, no. 10, pp. 1069–1079, 2010.

[37] M. Li and C. S. Smidts, "A ranking of software engineering measures based on expert opinion," *IEEE Trans. Softw. Eng.*, vol. 29, no. 9, pp. 811–824, Sep. 2003.

[38] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Inf. Softw. Technol.*, vol. 55, no. 7, pp. 1165–1199, 2013.

[39] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[40] P. N. Druzhkov and V. D. Kustikova, "A survey of deep learning methods and software tools for image classification and object detection," *Pattern Recognit. Image Anal.*, vol. 26, no. 1, pp. 9–15, 2016.

[41] V. Gulshan *et al.*, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *J. Amer. Med. Assoc.*, vol. 316, no. 22, pp. 2402–2410, 2016.

[42] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.

[43] S. Min, B. Lee, and S. Yoon, "Deep learning in bioinformatics," *Briefings Bioinf.*, vol. 18, no. 5, pp. 851–869, 2017.

[44] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: Review, opportunities and challenges," *Briefings Bioinf.*, vol. 19, no. 6, pp. 1236–1246, 2017.

[45] W. Sun, S. Shaoa, R. Zhaob, R. Yana, X. Zhangc, and X. Chen, "A sparse auto-encoder-based deep neural network approach for induction motor faults classification," *Measurement*, vol. 89, pp. 171–178, Jul. 2016.

[46] H. T. Cheng *et al.*, "Tensorflow estimators: Managing simplicity vs. flexibility in high-level machine learning frameworks," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 1763–1771.

[47] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 874–896, Sep. 2018. doi: 10.1109/TSE.2017.2720603.

[48] P. D. Singh and A. Chug, "Software defect prediction analysis using machine learning algorithms," in *Proc. 7th IEEE Int. Conf. Cloud Comput., Data Sci. Eng.-Confluence*, Jan. 2017, pp. 775–781.

[49] G. Litjens *et al.*, "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, pp. 60–88, Dec. 2017.

[50] N. Fenton *et al.*, "Predicting software defects in varying development lifecycles using Bayesian nets," *Inf. Softw. Technol.*, vol. 49, pp. 32–43, Jan. 2007.

[51] Y. Liu, Y. Li, J. Guo, Y. Zhou, and B. Xu, "Connecting software metrics across versions to predict defects," in *Proc. IEEE 25th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2018, pp. 232–243.

[52] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: Multi-objective effort-aware just-in-time software defect prediction," *Inf. Softw. Technol.*, vol. 93, pp. 1–13, Jan. 2018.

[53] D.-L. Miholca, G. Czibula, and I. G. Czibula, "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks," *Inf. Sci.*, vol. 441, pp. 152–170, May 2018.

[54] R. Krishna and T. Menzies, "Bellwethers: A baseline method for transfer learning," *IEEE Trans. Softw. Eng.*, to be published. doi: 10.1109/TSE.2018.2821670.
[55] M. Asghar, I. A. Khan, W. Anwar, and B. Ahmad, "Systemized approach for software corrective maintenance effort reduction," *J. Basic Appl. Sci. Res.*, vol. 1, no. 10, pp. 1356–1362, 2011.
[56] T. M. Pigoski, *Software Maintenance*. Pensacola, FL, USA: Technical Software Services, 2011.
[57] J. Hernández-Orallo, "Evaluation in artificial intelligence: From task-oriented to ability-oriented measurement," *Artif. Intell. Rev.*, vol. 48, no. 3, pp. 397–447, 2017.
[58] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia, "Detecting code smells using machine learning techniques: Are we there yet?" in *Proc. 25th IEEE Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers, Mar. 2018, pp. 612–621.

SUDAN JHA was born in Kathmandu, Nepal. He received proficiency in certificate level from the Saint Xavier's College, Kathmandu and the B.E. degree in electronics engineering from the Motilal Nehru Regional College, Allahabad, Uttar Pradesh, India, in 2001. He joined as a Lecturer with the Nepal Engineering College (nec), one of the premium and largest engineering college and the first one in the private domain in Nepal, where he got full sponsorship from the employer (nec) to pursue master's degree in computer science. He was an Assistant Professor with the Department of Computer Science and Engineering after completion of his master's degree and no sooner than later, he became the Head of the Computer Science and Engineering Department, Nepal Engineering College. In due course of time, he chaired and organized five international conferences, and some of the proceedings of those conferences had been published by Springer Verlag, World Science Series, and Imperial Press London.

RAGHVENDRA KUMAR received the B.Tech. degree in computer science and engineering from SRM University Chennai, Tamil Nadu, India, the M.Tech. degree in computer science and engineering from KIIT University, Bhubaneswar, Odisha, India, and the Ph.D. degree in computer science and engineering from Jodhpur National University, Jodhpur, Rajasthan, India. He is currently an Assistant Professor with the Computer Science and Engineering Department, L.N.C.T Group, College Jabalpur, India. He has published 86 research papers in international/national journals and conferences, including the IEEE, Springer, and ACM. He serves as the Session Chair, the Co-Chair, a Technical program Committee Member in many international and national conferences and as a Guest Editor in many special issues from reputed journals (indexed by: SCOPUS and ESCI). He also received the Best Paper Award in the IEEE Conference 2013 and the Young Achiever Award 2016 from the IEAE Association for his research work in the field of distributed database. His research interests include computer networks, data mining, cloud computing and Secure multiparty computations, theory of computer science, and design of algorithms. He has authored 12 computer science books in the field of data mining, robotics, graph theory, and turing machine published by IGI Global Publication, USA, IOS Press Netherland, Lambert Publication, Scholar Press, Kataria Publication, Narosa, Edupedia Publication, S. Chand Publication, and Laxmi Publication.

LE HOANG SON received the Ph.D. degree in mathematics and informatics from the VNU University of Science, Vietnam National University (VNU), in 2013. He has been an Associate Professor in information technology, since 2017. He was a Senior Researcher and the Vice Director of the Center for High Performance Computing, VNU University of Science, Vietnam National University, from 2007 to 2018. Since 2018, he has been the Head of the Department of Multimedia and Virtual Reality, VNU Information Technology Institute, VNU. His major fields include artificial intelligence, data mining, soft computing, fuzzy computing, fuzzy recommender systems, and geographic information systems.

Dr. Son is a member of the International Association of Computer Science and Information Technology (IACSIT), Vietnam Society for Applications of Mathematics, Vietnam, and the Key Laboratory of Geotechnical Engineering and Artificial Intelligence, University of Transport Technology, Vietnam. He serves on the Editorial Board of *Applied Soft Computing* (ASOC) in SCIE, *International Journal of Ambient Computing and Intelligence* (IJACI) in SCOPUS, and Vietnam *Journal of Computer Science and Cybernetics* (JCC). He is an Associate Editor of the *Journal of Intelligent & Fuzzy Systems* (JIFS) in SCIE, IEEE Access in SCIE, *Neutrosophic Sets and Systems* (NSS), Vietnam *Research and Development on Information and Communication Technology* (RD-ICT), *VNU Journal of Science: Computer Science and Communication Engineering* (JCSCE), and *Frontiers in Artificial Intelligence*.

MOHAMED ABDEL-BASSET received the B.Sc., M.Sc., and Ph.D. degrees in information systems and technology from the Faculty of Computers and Informatics, Zagazig University, Egypt. He has published more than 150 articles in international journals and conference proceedings. His current research interests include optimization, operations research, data mining, computational intelligence, applied statistics, decision support systems, robust optimization, engineering optimization, multi-objective optimization, swarm intelligence, evolutionary algorithms, and artificial neural networks. He is working on the application of multi-objective and robust meta-heuristic optimization techniques. He is also an Editor or a Reviewer in different international journals and conferences. He holds the program chair in many conferences in the fields of decision making analysis, big data, optimization, complexity and the Internet of Things, as well as, editorial collaboration in some journals of high impact.

ISHAANI PRIYADARSHINI received the B.Tech. degree in computer science and engineering from KIIT University, and the double master's degree in information security from KIIT University and in cybersecurity from the University of Delaware, USA, where she is currently pursuing the Ph.D. degree in electrical and computer engineering (cybersecurity). Her current research interests include cryptography, network security, and machine learning.

ROHIT SHARMA received the B.Tech. degree in electronics and communication engineering from Uttar Pradesh Technical University, Lucknow, India, the M.Tech. degree in communication engineering from Shobhit University, India, and the Ph.D. degree in electronics and communication engineering from Teerthanker Mahaveer University, Moradabad, India. His Ph.D. thesis was entitled Evolution in RFID Security to Building It in the Global Environment. He is currently an Assistant Professor of electronics and communication engineering with SRM University, Delhi NCR Campus, Ghaziabad, India. He joined SRM University, in 2011. He has a teaching experience of over seven years at SRM University and Dewan V.S. Group of Institutions, India. He has published about 38 research papers in international/national journals and about nine research papers in international/national conferences. He is an Active Member of the ISTE, ICS, IAENG, and IACSIT. He is an Editorial Board Member and a Reviewer of more than eight international journal and conferences. He has been an Editor of the 2nd International Conference on Microelectronics and Telecommunication.

HOANG VIET LONG received the Ph.D. Diploma degree in computer science from the Hanoi University of Science and Technology, in 2011, where he defended his thesis on fuzzy and soft computing field. He is the Head of the Faculty of Information Technology, People's Police University of Technology and Logistics, Bac Ninh, Vietnam. He is currently a Researcher of the Institute for Computational Science, Ton Duc Thang University, Ho Chi Minh City, Vietnam. He has been an Associate Professor in information technology, since 2017. Recently, he has been concerning in cybersecurity, machine learning, bitcoin, and block chain and published more than 20 papers in ISI-covered journal.