Bugging and De-Bugging the Dataset using Improved Hidden Morkov Model (IHMM) for Cybersecurity Applications

Abdullah Moaid Mohammed Al-Shehri[a] and Mourad Elloumi[b]
[a] Masters students, Faculty of Computing and Information Technology, University of Bisha, Bisha, Saudi Arabia
[b]Professor, Faculty of Computing and Information Technology, University of Bisha, Bisha, Saudi Arabia

Article History: Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 28 April 2021

Abstract: The priority has been gravely given to cyber security events' finding, calculation or treatment in the wake of research field for the sake of the centers that are available for response of security. In such a surrounding or situation, the automation is a vital requirement for many processes in quest of finding a solid solution to the damage caused by the threats and assailings to the companies as well to the citizens. The process of bug handling is almost as a manual process which is but as at large a costly administration or maintenance of software systems. While a vast sum of time as well as effort is poured upon dealing the bug reports, this way automating thus the parts of this process of bug handling can save the wasted vast sum of time as well as effort while dealing the bug reports. Bug cause comes handy for bug perception and bug localization, which to enkindle the developers must have scrutinized the source code during the process of bugging (bug fixing) and debugging.

This paper aims at exploiting the corresponding relationship between bug that fixes to automatically classify bugs from dataset using Improved Hidden Morkov Model (IHMM) which utilize Baum-welch genetic algorithm optimization process for better convergence. The aim of this work is to save the time as well as effort from being wasted in fixing the bugs while modifying (towards the improvement) the overall software's quality. The model proposed achieves 98.3% accuracy, 81.2% precision, 79.9% recall and 90.2% AUC under Mozilla setup and 97.1% accuracy, 88.4% precision, 91.4% recall and 83.7% comparatively with its performance as when compared with other models under android firefox setup.

Keywords: Cyber security, Deep learning, Bugging, Debugging, Feature extraction

1. Introduction

With the ever increasing incorporation between the society's normal life and internet, internet modifies and moulds the life of the society, yet it often makes the society yield the security threats. Along with the lots of learning, the society has to yield to the threats such too. Can we easily spot out the network threats and attacks? How it will be feasible to find out the variant attacks of the network, particularly the unseen so far? The main crisis is to detect the way out of all these attacks but for that we have to spot out such attacks is the further main criteria. Cyber security is such a technology that protects very meticulously the computers or intact networks, programs as well as data from attacks and access unauthorized **[Milenkoskiet al., (2015)].** Both the network security system as well as computer security systems are available for us in the world and each includes IDS (Intrusion Detection Systems, firewalls, and antivirus software, which detects, fixes and identifies unauthorized system behaviors such as use, copying, modification and destruction **[Modi et al., (2016)].**Categorizing a cyber security's inherent damage is a main procedure in the cyber security field to find out, calculate, and respond to an event in a manipulated way. These days we have variant modes or scoring standards that permit us to evaluate the harsh sense or severity of a cyber-event. The taxonomies assign **[Sun et al., (2019)].** Including the information addressed in the global reports for the calculation, the taxonomies assign indeed attacks' severity and these indicators are nowadays assigned manually by experts all by means of laborious or large qualitative descriptions. Such procedure as this is one of the main crises in management centers of cyber-incidents **[Sampathkumar et al., (2020)].** When it all comes to be a malfunction in a program that is computer oriented, then such erroneous malfunction or flaw is defined as a software bug, which is all utilized in cyber-attacks towards benefits and as well as towards the systems' behaving away from their virtual intention. Software bugs cause vulnerabilities of software, which when exploited result in cyber-attacks. A large number of cyber-attacks are caused thus only. There are algorithms that are feasible and automatic that categorize a cyber-event's severity using deep learning techniques for the sake of solving such a situation as this. This study comes handy in assisting in the wake of perceiving the merit of the algorithms of deep learning to categorize and as well to correlate malignant activities that are grasped or caught interrupting from many out sources like Domain Name System (DNS), email, Uniform Resource Locator (URLs)etc.,**[Rhode et al., (2018)]**. Unlike traditional machine learning approaches, these deep learning algorithms don't follow any engineering feature and don't have any illustration ways.

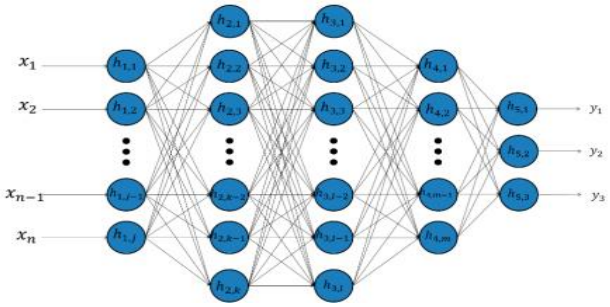

Figure 1: Deep Neural Network

Figure 1 shows such levels as that correspond to the concepts' distinct levels. Yet it is so that the options of the further domain level ought to outline the ways of deep learning in the field of information science and its several tasks. The cyber security events contemplated and concentrated during this study are enclosed by text **[Aminanto (2017)]**.

For the sake of recognition, the sensitive data is contained by the input data and thus as to the Deep Learning machines. The user feels safer when the user installs the model of the Deep Learning on its platform as well as handles the model for the use .And it is not so possible for the user to use the Deep Learning model always, for it does consist of massive data and it processes such massive data **[Sampathkumar et al., (2018)].** Every enterprise wishes a secret administration of its data, for its competitor enterprise may still fish out the data of its own and use such for their business purposes **[Rinaldi et al., (2001)]**. We come to a finding of three cardinal requirements:

- The training model's stored data shouldn't be exposed to the server
- The user request too shouldn't be exposed to the server
- The configurations of the server as well ought not to be disclosed to the user

No intruder can disclose information and no attacker too can disclose the information. Thus is the course of computation and while sharing such computation the privacy or the secrecy of the information is still as protected and so only it is all as advisable for the organizations to make use of the deep learning so that the frame works of cyber security can be strongly established. Thus we come to a conclusion that the identified bug handling is indeed as a costly and laborious task and the more automation of the bug handling process occurs, the more efficient process of it can be done and also less wasteful process of it can be made.

Organization of this paper is as: Section 1 explained the introduction about cyber security, bugging in software and role of deep learning in cyber security applications. In section 2 various existing deep learning and classification techniques in cyber security. Section 3 illustrates the proposed model with feature extraction, dataset minimization and feature selection. The performance analysis is illustrated by means of Section 4 along with results as well as graphs and this work ends by presenting the future work in section 5.

2. Literature Survey

Sepahvand et al., (2020) suggests a new and strange model of deep learning called rather as Deep Long Short Term Memory Prediction (DeepLSTMPred), which transforms constituent terms into a vector of real numbers by means of considering the semantic meaning of the numbers, and finds out the long-term dependencies feasible between terms by means of deep Long Short Term Memory (LSTM) and finally categorizes the sequences into short or long fixing time. DeepLSTMPred is assessed on bug reports that are as emendated from the Mozilla project. Experimentally it is come to be known that DeepLSTMPred accomplishes 15-20% further elevation so far as accuracy, f-score, as well as recall are concerned. But implementation rules are by far inferred inductively from training data is a limitation [9].

Ferenc et al., (2020) made the metrics of static source code as to become revitalized and thus made these metrics amidst predictors that are meaningful as well as easily calculable and combined such with deep learning amidst the techniques that are very assuring and common to flag code segments. The authors took up some deep neural networks to apply to a large dataset of bug as to find out the bug prediction. The authors thus finally applied the taken deep neural networks to a data set of a large kind that contained about 8780 bugged Java classes as well as 38,838 not bugged Java classes and then, finally compared the same neural networks practically to several algorithms that are traditional. Ensemble model that consists of a deep learning component model has an F-measure of 53.59%, that increases to 55.27% for the best. The limitation is there is no information about relative difference during bugging and debugging process [10].

Sushant Kumar Pandey et al., (2019), suggested or proposed Bug Prediction [which is as a rudimentary classification based framework, and which combinedly applies together the Ensemble Learning (EL) and Deep Representation (DR)] by using Deep representation and Ensemble learning (BPDET) Technique. Mostly the traditional metrics of software are used for Software Bug Prediction (SBP). A strong feature learning method is this encoder called SDA (Staked Denoising Auto-encoder), which comes to be of much use in representing software metrics. The authors use SDA in representing software metrics. The authors grasp dual stages such as

deep learning stage as well as Two layers of Ensemble Learning stage (TEL) as proposed by dividing SDA into two stages but the authors didn't concentrate much to address the problem of class imbalance [11].

In search of bug reports' deep calculation or prediction, **Waheed YousufRamay et al., (2019**) suggests an approach that is both automatic as well as neural (net work based).In this approach, after applying processing techniques of inherent (natural) language for the sake of bug reports' text preprocessing, an emotion score is calculated and assigned for every one of the bug reports. Then it so occurs that after creating a vector rather for each and every bug report that is as preprocessed, the every bug report's so constructed vector as well as emotion score are passed by us to a classifier that is as deeply as neural network based and thus for the wake of severely predicting but due to this way a proceeding there can be a drawback, that is, overall improvement in f-measure is very less [12].

Duc Tran et al., (2018), in order to fight the botnet type, gave a Long Short-Term Memory network (LSTM), which functions on raw domains and is easy to apply for immediate applications though it is however prone to yield to multiclass imbalance problem, which becomes even more significant in malware detection. The original LSTM is taken for to make a strange Long Short-Term Memory Machine Learning (LSTM-ML) algorithm to give a combination of both binary and multiclass classification models, and further to make it to be cost-sensitive, the original LSTM is taken. The experiments are conducted on a real-world collected dataset to prove that LSTM.MI gives such an improvement as of at least 7% in terms of macro-averaging recall and precision comparatively, and thus when compared to the original LSTM as well as other state-of-the-art cost-sensitive methods. But this method achieves not too far an accuracy but a less accuracy only and that is the limitation of this method, and hence the author gives further future work to improve the accuracy [13].

Dharmendralalgupta et al., (2017) have developed an object-oriented model and that is, Software Bug Prediction System (SBPS), which can find out by means of metrics the bugs' existence in a class and can predict rather the bugs' occurrences too. This model has accomplished an aggregate accuracy of 76.27% when tested on dataset and Combined Dataset validated. The limitation is fault-prone classes is not included which results in high computation cost [14].

JifengXuan et al., (2014) face data reduction problem for the end result of bug triage. We mix both the instance selection as well as feature selection together to obtain the attributes from historical bug data sets and then to sequently build a predictive model for a new bug data set and the results demonstrate that our data reduction can efficiently do the data scale's reduction and thereby, as a result, the bug triage's accuracy can be improved. The limitation is, because of a lot of bug repositories there occurs a lagging of apt labels to observe and due to such cause there occurs a redundancy [15].

ShivkumarShivajiet al., (2012) researched techniques which are as multiple feature selection, which are also as commonly as can be applied to bug prediction methods that are classification-based. Till we get maximized classification performance, the techniques throw off less important features. The sum total of the features employed for training is brought down to less than 10 percentage of the original and while thus the reduction is substantial and as often, the SVM's improvement in buggy F-measure is as 9 percentage. When it comes to be the matter of performance analyzing, a strong performance is achieved at even 1% of the original number of features [16].

Sureka et al.,(2010), in quest of sharpening the accuracy of the detection of automatic duplicate bug report and hence, proposed an approach called as a character-level n-gram approach. The character-level n-grams are independent of language saving languages specific pre-processing time. As envisioned as per their experiments, their approach performed modestly to accomplish almost 21 percentage and 34 percentage recall rates for the top 10 as well as top 50 recommendations, correspondingly [17].

2. Research Methodology

In this section, Improved Hidden Morkov Model (IHMM) is proposed by hybridizing Baum-Welch (BW) and Genetic algorithm. Fig. 2 shows the proposed model's architecture, whereas variant dataset types like Iris1.6, Glass 6.0, Ant 1.7, jEdit4.3, Apache Tomcat 2.0, arc, Q-teaching, berekal, forest 0.8, Zuzel and Intercafe are deemed for bugging and debugging processes. These datasets undergone bugging and debugging process and hence the bug report was updated. Some of the features are extracted for the purpose of dataset minimization. This extracted features undergone Learning process which consists of three stage process. After learning process, Expectation maximization is done by Baum-welch genetic algorithm method for the purpose of likelihood function.

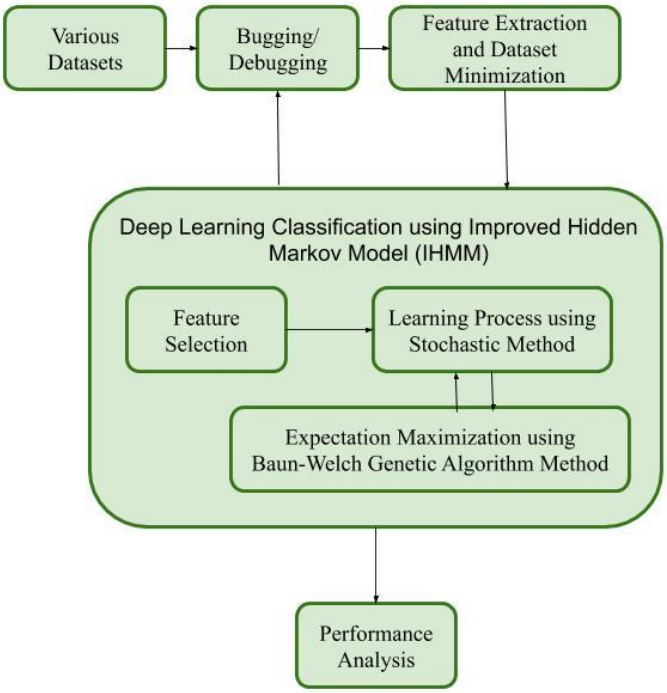

Figure 2: Architecture of the proposed model

Dataset consideration

Open source projects of the same mould kinds (Iris1.6, Glass 6.0, Ant 1.7, jEdit4.3, Apache Tomcat 2.0, arc, Q-teaching, berekal, forrest 0.8, Zuzel and Intercafe) have been preferred, and such are indeed as up to the mark or modern as well as got from Promise Software Engineering Repository.

XML file of Bug Reports is taken as input with the summary and on the textual summary the tokenization, Stop Word Removal and stemming are done and these are standard preprocessing steps.

a) Tokenization: When we removing periods, commas, punctuation, hyphens and brackets, we call such task as tokenization.

b) Stop Word Removal: When we reduce the textual data as removing unwanted words, we call such task as Stop Word Removal and as a result of this task the system's performance is improved.

c) Stemming: When we reduce the textual data as cutting the word from their root node, we call such task as Stemming.

Feature extraction and dataset minimization

Feature selection is a main processing step applied usually to the dataset before running the machine learning algorithm for evicting the irrelevant or weakly relevant features and choosing the optimal subset of all features. Feature selection identifies exactly the proper subset of all the data features. It only makes the data as serviceable.

Deep learning classification using Improved Hidden Morkov Model (IHMM)

Process of Markov indeed deems with all its assumptions that (while in HMM the states are as hid) the states are obviously seen to the system produced observation data. But every hidden state's output (i.e., the state transition's probability) is as according to the observation data. An HMM model's training needs must have the defining of the parameters following:

Hidden states' Number: In order to accomplish one HMM model's training, we ought to arrange the hidden states' count or number (N) in the process of Markov. Let us assume that the very clear(distinct) states in a Markov process are as M(n) , n= { 0 , 1 , …, i− 1 } and let us assume that the notation does represent the sequence of hidden states M(n) at time t.

Number of observation symbols: We ought to arrange the number of observation symbols (obs(n)) in order to train an HMM model. Let us assume the distinct observation symbols as $dist_{obs(n)}$, n= { 0 , 1 , …, obs(n)− 1 } and let us further assume the notation as obs(n))= $dist_{obs(n)}$, as that does represent the observed symbol $dist_{obs(n)}$, at time *t* for the given observations ($obs_{\Omega 0}$, $obs_{\Omega 1}$,$obs_{\Omega 2}$…, $obs_{\Omega T-1}$) sequence , where T is as that sequence's length.

Learning process using stochastic method

State transition probability distribution: The probability distribution matrix of first-row B= { b_{ij}}. B is an N ×N square matrix and each element's probability { b_{ij}} is computed by means of the following equation:

$$bij = P(state\,M(n)_j att + 1 \mid state\,M(n)_i att),\ i, j = \{ 0 , 1 , \ldots, N - 1 \} \quad (1)$$

Observation symbol probability distribution: The probability distribution matrix of the second-row B= { $M(n)_j(dist_{obs(n)})$ } . is an $N \times M$ matrix which is calculated based on the sequences of observation (i.e., the stack

traces' temporal order). The probability of each element $M(n)_j(dist_{obs(n)})$ is provided by means of the following equation:

$$M(n)_j(dist_{obs(n)}) = P(observationsymbol\, dist_{obs(n)}\, att \text{ and } state M(n)_j att) \quad (2)$$

Initial state probability distribution: The probability distribution matrix of the third-row $\beta = \{ \beta i \}$. $\beta$is a $1 \times N$ row matrix and the probability of each element $\{ \beta j \}$ is provided by means of the following equation:

$$\beta i = (state M(n)\ att = 0) \quad (3)$$

Baum-Welch Genetic algorithm:

HMM model's training targets the maximizing of HMM model's training targets the maximizing of the likelihood function ($\Omega \mid \lambda$) and the Baum-Welch (BW) algorithm is all very normally used employed algorithm called as expectation-maximization (EM) algorithm towards calculating the parameters of HMM . It is a genetic algorithm and it can be as at each iteration to efficiently assess the likelihood function ($\Omega \mid \lambda$). The model parameters are updated by it until either the likelihood function reaches no further improvement or until an optimum number of iterations is accomplished or either until the slow convergence problem is suppressed by sheer means of applying an iteration of Baum-Welch (BW) algorithm and the initial generation Pop (a) that isn't at all rather generated at random as in normal Genetic Algorithm (GA).

Fitness function calculation: The fitness function is a mechanism of evaluation and it evaluates the mechanisms of the chromosome; a higher fitness value reflects the chromosome's chances to be chosen in the next generation. The log likelihood has been used, which represents the likelihood that the training observation utterances have been generated by means of the current model parameters and it is a function of the following form

$$A(n) = \sum_{k=1}^{m} \log (p(_{ak/\lambda}n)))/m \quad (4)$$

Selection and Crossover: Few chromosomes are randomly selected by mutation and some genes are alerted to produce new chromosomes. Selection imitates the fittest mechanism's survival seen in Nature. Chromosomes of higher fitness values can rather survive amidst the forthcoming generations and further, to apply crossover, some elements will be chosen from the pool of population.

Improved Hidden Morkov Model (IHMM) algorithm:

1. Number of hidden states (N)
2. Distinct states M(n)
   M(n)<=n = { 0 , 1 , …, i − 1 }
   $X(t)$<= M(n)
3. Distinct observation symbols be $dist_{obs(n)}$
   $dist_{obs(n)} <= n$ = { 0 , 1 , …, obs(n)− 1 }
4. Apply stochastic method
   B= { b_{ij}}
   If
   $(state M(n)_j att + 1 \mid state M(n)_i att)$,
   Else
   $\mathcal{B}$= { $M(n)_j(dist_{obs(n)})$ }
   $(observationsymbol\, dist_{obs(n)}\, att \text{ and } state M(n)_j att)$
   Else
   $\beta = \{ \beta i \}$
   $\beta i = (state M(n)\ att = 0)$
   end if
5. BW algorithm can be applied to produce the initial population pop(a) where pop(a) = {D1, D2, ... , DN}, and Dj is one chromosome
6. Measure every chromosome Dj's fitness function Fit(Dj) as within the original population pop(t).
7. For intermediate population pop'(t), select only a few chromosomes.
8. A certain crossover can be applied to some chromosomes pop'(t).
9. A certain mutation can be applied to few chromosomes pop'(t).
10. Apply a certain three iterations of B-W algorithm to the population pop'(t) for each ten generations.
11. t = t+l; if not convergence, and then go to step 6

4. Performance Analysis

In order to evaluate the value of the proposed Improved Hidden Morkov Model (IHMM) with two existing classification algorithms such as Deep Long Short Term Memory Prediction (DeepLSTMPred) and Predicting Bug adopting Deep representation and Ensemble learning (BPDET) Technique experiments are conducted. Variant data sets like Iris1.6, Glass 6.0, Ant 1.7, jEdit4.3, Apache Tomcat 2.0, arc, Q-teaching, berekal, forrest 0.8, Zuzel and Intercafe are considered with two experimental areas such as, Mozilla and Android FireFox. As for the purpose of evaluating the proposed model, the performance measures such as accuracy, precision, recall as well as F-measure were used.

The table 1 shows the simulation parameters for Improved Hidden Morkov Model (IHMM)

Table 1: Parameters of Simulation

Parameter	Value
Learning rate	0.05-0.10
Number of epoch	200
Number of hidden nodes	50-500
Batch size	1500
Optimization	Baum-Welch genetic algorithm
Number of iterations	200

The table 2 shows the construction of confusion matrix for bugging and debugging process to analyze parameters such as, precision, accuracy, recall and Area Under Curve(AUC)

Table 2: Confusion of Matrix

Confusion of Matrix	Class Predicted	
	Bugging	**Debugging**
Class Actual	True Positive	False Negative
	False Positive	True Negative

TP is True Positive
TN is True Negative
FP is False Positive which is Type-I error
(a) (really bugging instances are categorized as debugging)
FN means False Negative which is Type-II error
(b) (really debugging instances are categorized as bugging)
Sum total of instances = aptly categorized instances +inappropriately categorized instances
Aptly categorized instance = TP + TN
Inappropriately categorized instance = FP + FN

- **Accuracy(acc):**It is as the ratio of the number of correctly classified bugging and debugging datasets to the sum total of bugging and debugging reports

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

- **Precision(Pre):** It is the ratio of precisely classified bugging and debugging datasets to the total number of bugging and debugging reports

$$\textbf{Precision} = \frac{TP}{TP+FP} \quad (2)$$

- R**ecall(Rec):** It is aptly as even as the ratio of aptly classified bugging and debugging datasets to that total available datasets

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

- **Area Under Curve (AUC):**It is used for single numeric measurement to predict a classifier's potential indeed. Mostly a higher AUC is well and good. The table 3 represents the analysis of accuracy between existing DeepLSTMPred and BPDET with proposed IHMM method.

Table 3: Comparison of Accuracy(%)

Dataset	DeepLSTMPred [9]		BPDET[11]		IHMM(proposed)	
	Mozilla	Fireox Android	Mozilla	FireFox Android	Mozilla	FireFox Android
Iris1.6	77.2	85.2	60.2	80.2	97.2	96.2
Glass 6.0	77.4	85.4	60.4	80.4	97.4	96.4
Ant 1.7	77.6	85.6	60.6	80.6	97.6	96.4
jEdit4.3	77.8	85.8	60.8	80.8	97.6	96.8
Apache Tomcat 2.0	78	86	61	81	97.8	96.8
arc	78.2	86.2	61.2	81.2	98	97
Q-teaching	78.4	86.4	61.4	81.2	98.2	97.2
berekal	78.6	86.6	61.6	81.4	98.4	97.4
forrest 0.8	78.8	86.8	61.8	81.6	98.6	97.6
Zuzel	79	87	62	81.8	98.8	97.8
Intercafe	79	87	63	82	99	98

The table 4 represents the analysis of precision between existing DeepLSTMPred and BPDET with proposed IHMM method.

Table 4: Comparison of precision(%)

Dataset	DeepLSTMPred [9]		BPDET[11]		IHMM(proposed)	
	Mozilla	FireFox Android	Mozilla	FireFox Android	Mozilla	FireFox Android
Iris1.6	60	73	55	76	80.2	87
Glass 6.0	60.2	73.2	55.4	76.3	80.4	87.3
Ant 1.7	60.4	73.4	55.6	76.5	80.6	87.5
jEdit4.3	60.6	73.6	55.8	76.5	80.8	87.8
Apache Tomcat 2.0	60.8	73.8	56	76.8	81	88
Arc	61	74	56	77	81.2	88
Q-teaching	61.2	74	56.2	77.2	81.2	88.2
Berekal	61.4	74.3	56.4	77.4	81.4	88.4
forrest 0.8	61.6	74.6	56.6	77.6	81.6	88.6
Zuzel	61.8	74.9	56.8	77.8	81.8	88.8
Intercafe	62	75	57	78	82	89

The table 5 represents the analysis of recall between existing DeepLSTMPred and BPDET with proposed IHMM method.

Table 5: Comparison of recall(%)

Dataset	DeepLSTMPred [9]		BPDET[11]		IHMM(proposed)	
	Mozilla	FireFox Android	Mozilla	FireFox Android	Mozilla	FireFox Android
Iris1.6	57	83.2	70	86.2	78.4	90.2
Glass 6.0	57.2	83.4	70.2	86.4	78.4	90.4
Ant 1.7	57.5	83.4	70.4	86.6	78.6	90.6
jEdit4.3	57.8	83.6	70.6	86.6	78.8	90.8
Apache Tomcat 2.0	58	83.8	70.6	86.8	78.8	90.8
Arc	58.2	84	70.8	87	79	91
Q-teaching	58.2	84.2	71	87.2	79.2	91.2
Berekal	58.4	84.4	71.2	87.4	79.4	91.4
forrest 0.8	58.6	84.6	71.2	87.6	79.6	91.6
Zuzel	58.8	84.8	71.6	87.9	79.8	91.8
Intercafe	59	85	72	88	80	92

The table 6 represents the analysis of Area Under Curve(AUC) between existing DeepLSTMPred and BPDET with proposed IHMM method.

Table 6: Comparison of AUC(%)

Dataset	DeepLSTMPred [9]		BPDET[11]		IHMM(proposed)	
	Mozilla	FireFox Android	Mozilla	FireFox Android	Mozilla	FireFox Android
Iris1.6	76	83.2	80.2	75	89.3	82.2
Glass 6.0	76.2	83.4	80.4	75.2	89.5	82.4
Ant 1.7	76.4	83.4	80.6	75.4	89.8	82.6
jEdit4.3	76	83.6	80.8	75.6	90	82.8
Apache Tomcat 2.0	76.2	83.8	81	75.6	90	83
Arc	76.4	84	81.2	75.8	90.3	83.4
Q-teaching	76.6	84.2	81.2	75.8	90.5	83.5

Berekal	76.8	84.4	81.4	76.2	90.8	83.6
forrest 0.8	76.8	84.6	81.6	76.4	90.8	83.8
Zuzel	77	84.8	81.8	76.8	91	83.8
Intercafe	77	85	82	77	91.2	84

The table 7 represents the comparative analysis of various parameters between existing DeepLSTMPred and BPDET with proposed IHMM method when applied in Mozilla setup

Table 7: Comparative analysis of proposed and existing methods for experiment-1 (Mozilla setup)

Method	**Accuracy(%)**	**Precision (%)**	**recall(%)**	**AUC (%)**
DeepLSTMPred**[9]**	78.2	61.4	58	77
BPDET**[11]**	61.3	56	71	81
IHMM(proposed)	98.3	81.2	79.9	90.2

The table 8 represents the comparative analysis of various parameters between existing DeepLSTMPred and BPDET with proposed IHMM method when applied in android firefox setup

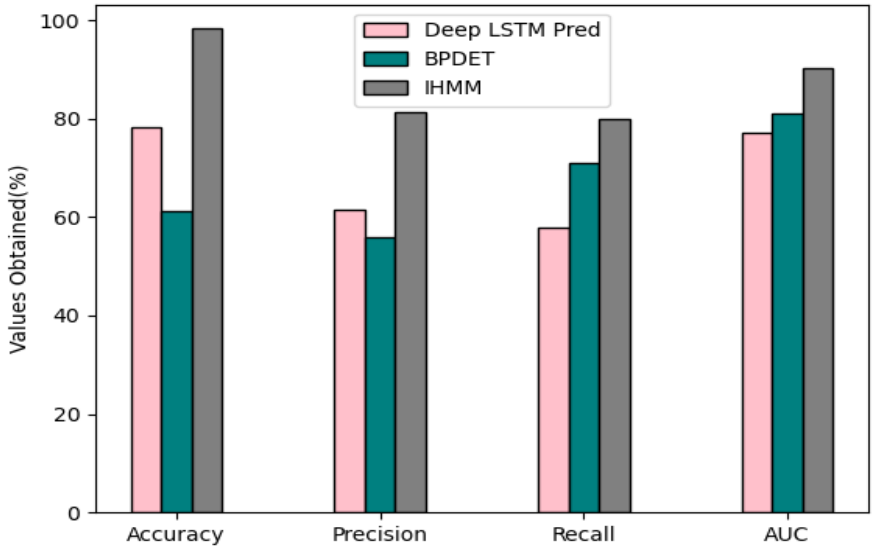

Figure 3: Comparison of various parameters for Mozilla setup

Figure 3 depicts the delay comparison of existing DeepLSTMPred and BPDET with proposed IHMM method when applied in Mozilla setup.X axis and Y axis shows that various parameters and the values obtained in percentage respectively. When compared, existing method achieves better accuracy, precision, recall and AUC values.

Table 8: Comparative analysis of proposed and existing methods for experiment-2 (Android firefox setup)

Method	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**AUC (%)**
DeepLSTMPred**[9]**	86	74	84.2	84
BPDET**[11]**	81	77.56	87.4	76.76
IHMM(proposed)	97.1	88.4	91.4	83.7

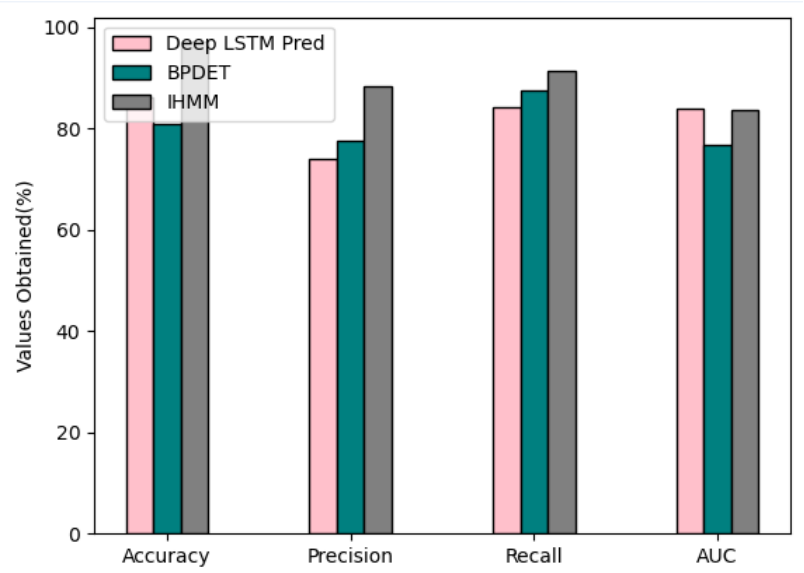

Figure 4: Comparison of various parameters for android firefox

Figure 4 depicts the delay comparison of existing DeepLSTMPred and BPDET with proposed IHMM method when applied in Android firefox setup. X axis and Y axis shows that various parameters and the values obtained in percentage respectively. When compared, existing method achieves better accuracy, precision, recall and AUC values

5. Conclusion

Both the bugging as well as debugging of software do progress forth as up against the cyber networks even enough as to outwit the capacity of the cyber defenders rather. Total lot of the bugs don't relate to critical issues; some are of indeed light weighted and need must have refinement, however, manual severity classification is often tremendously tedious as well as time-consuming. The deep learning methods offer a rich opportunity to apply on cyber security applications for better classification. In this particular paper, a novel approach has been sincerely presented by us and it has also been aimed so by us as at automatically classifying the bug from dataset by bugging and debugging using Improved Hidden Morkov Model (IHMM). The IHMM model has been validated upon eleven real datasets under tow experimental scenarios such as Mozilla and android firefox setup. As a result the proposed IHMM model achieves 98.3% accuracy, 81.2% precision, 79.9% recall and 90.2%AUC under Mozilla setup and 97.1% accuracy, 88.4% precision, 91.4% recall and 83.7% under android firefox setup.

References

1. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, "Evaluating Computer Intrusion Detection Systems:A Survey of Common Practices", *AcmComput. Surv.*, vol.48, no.1, pp.1-41, 2015.
2. N. Modi and K. Acha, "Virtualization layer security challenges and intrusion detection/prevention systems in cloud computing: a comprehensive review", *J. Supercomput.*, vol.73, no.3, pp.1-43, 2016.
3. Sun N, Zhang J, Rimba P, Gao S, Zhang LY, Xiang Y,"Data-driven cybersecurity incident prediction: a survey",*IEEE CommunSurv Tutor*; vol.21, no.2, pp.1744-1772, 2019
4. Sampathkumar A, JaisonMulerikkal and M. Sivaram,"Glowworm swarm optimization for effectual load balancing and routing strategies in wireless sensor networks", *Wireless Networks*,pp.1-12, 2020.
5. M. Rhode, P. Burnap and K. Jones, "Early-stage malware prediction using recurrent neural networks", *Computers & Security*, vol.77, pp.578-594, 2018.
6. M. E. Aminanto, R. Choi, H. C. Tanuwidjaja, P. D. Yoo and K. Kim, "Deep abstraction and weighted feature selection for Wi-Fi impersonation detection", *IEEE Transactions on Information Forensics and Security*, vol.13, no.3, pp.621-636, 2017
7. Sampathkumar A and P. Vivekanandan,"Gene selection using multiple queen colonies in large scale machine learning", *International Journal of Electrical Engineering*, vol.9, no.6, pp.97-111, 2018.
8. Rinaldi SM, Peerenboom JP, Kelly TK,"Identifying, understanding,and analyzing critical infrastructure interdependencies",*IEEE Control Systems Magazine*,vol.21, no.6, pp.11-25, 2001.
9. Sepahvand, Reza, Reza Akbari and SattarHashemi,"Predicting the bug fixing time using word embedding and deep long short term memories", *IET Software*, 2020.
10. Ferenc, Rudolf, DénesBán, TamásGrósz and TiborGyimóthy,"Deep learning in static, metric-based bug prediction", *Array*,2020.
11. Pandey, Sushant Kumar, Ravi Bhushan Mishra and Anil Kumar Tripathi,"BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques", *Expert Systems with Applications* , vol.144, 2019.
12. Ramay W.Y, Umer Q, Yin X.C, Zhu C and Illahi I, "Deep neural network-based severity prediction of bug reports", *IEEE Access*, vol.7, pp.46846-46857, 2019.

13. Tran, Duc, Hieu Mac, Van Tong, HaiAnh Tran and LinhGiang Nguyen,"A LSTM based framework for handling multiclass imbalance in DGA botnet detection", *Neurocomputing* , vol.275,pp.2401-2413, 2018.
14. Gupta, DharmendraLal and KavitaSaxena,"Software bug prediction using object-oriented metrics", *Sādhanā* , vol.42, no.5,pp.655-669, 2017.
15. Jifeng, Xuan, He Jiang, Yan Hu, ZhileiRen, WeiqinZou, ZhongxuanLuo and Xindong Wu,"Towards effective bug triage with software data reduction techniques", *IEEE transactions on knowledge and data engineering*, vol.27, no.1, pp.264-280, 2014.
16. Shivaji, Shivkumar, E. James Whitehead, Ram Akella and Sunghun Kim,"Reducing features to improve code change-based bug prediction", *IEEE Transactions on Software Engineering*, vol.39, no.4,pp.552-569, 2012.
17. Sureka and P. Jalote, "Detecting duplicate bug report using character N-gram-based features", *Journal of Software Engineering*, pp.366-374, 2010.