Bidirectional Recurrent Neural Network Language Model: Cross Entropy Churn Metrics for Defect Prediction Modelling

Nivetha. R, Kavitha. S

ABSTRACT- Software Defect Prediction (SDP) plays an active area in many research domain of Software Quality of Assurance (SQA). Many existing research studies are based on software traditional metric sets and defect prediction models are built in machine language to detect the bug for limited source code line. Inspired by the above existing system. In this paper, defect prediction is focused on predicting defects in source code. The aim of this dissertation is to enhance the quality of the software for precise prediction of defects. So, that it helps the developer to find the bug and fix the issue, to make better use of a resource which reduces the test effort, minimize the cost and improve the quality of software. A new approach is introduced to improve the prediction performance of Bidirectional RNNLM in Deep Neural Network. To build the defect prediction model a defect learner framework is proposed and first it need to build a Neural Language Model. Using this Language Model it helps to learn to deep semantic features in source code and it train & test the model. Based on language model it combined with software traditional metric sets to measure the code and find the defect. The probability of language model and metric set Cross-Entropy with Abstract Syntax Tree (CE-AST) metric is used to evaluate the defect proneness and set as a metric label. For classification the metric label K-NN classifier is used. BPTT algorithm for learning RNN will provide additional improvement, it improves the predictions performance to find the dynamic error.

Keywords- Software Defect Prediction Modeling, Bidirectional RNN Language Model, Deep Learning, Software Metrics.

I. INDRODUCTION

When frequently existing of a software failure in system for many time it automatically indicates to a software defect/fault in a system. A software defect is an error it occurs with the poor software quality. When the program runs the compiler will show the error in the code that is syntax error and a static semantic error this kind of software defect/bugs can be found and fixed easily by the computer programmer. [1] But when the program compiles and runs the code the logical error and dynamic semantic error cannot be detected at the compile time and it produces wrong results when the program executed in the run time this software defect is called bug it will lack the software product that is the customer needs. So this software defects cant produces a good quality of software. [2] The Software Organization job is to develop and design the software projects, the main role of the computer programmers in IT is a developer has to build the software system and Tester has to find the bug and fix the issues and deliver a quality software product as per the customer requirement. Many Software Defect Prediction Model has been created it is the smart way of automating the process of training and learn from The enormous software information collection and application of this model to recognize new data bugs help feed the training module and test the software [3]. To attain this prediction model mainly influenced by the software metrics. And to build an effective defect prediction model Machine learning technology is used it formulates the predictive defect classification model from different code attributes using Machine learning algorithms [4]. Support vector Machines (SVM), Linear Regression (LR) Logistic Regression (LR), Naïve Bayes (NB), Ensemble learning (EL), Dictionary learning (DL), Transfer learning (TL) and Deep Neural Network (DNN) in this paper K-Nearest Neighbor classifier is used. So, that to develop model first thing is the model should be trained with the past datasets for this Machine learning is used to learn the syntactic and semantic information of the particular programming language [5].

Software metric which is used to measure the features (or) characteristics those are measurable (or) countable. It is important for measuring the software performance, measuring the productivity and for much purpose [3]. It is a quantitative metric to predict the amount of defects in the software metrics component which is used to assess the progress the software source code to measure the line of code, complexity, cohesion and churn (number of modification in LOC) and collects the previous defects data sets as a statistical traditional metric sets to predict the defect in code [3][4]. Software metric is the parameters to measure the features in a software system.

Many researches devote their effort to design new metrics (features) to measure and predict the entropic in code. To perform a defect prediction task many software metrics is commonly used to measure the natural language. There are some important software metrics used to predict task and they are considered as traditional metric sets [5]. Halstead Metrics based on operator & operand count. McCabe Essential complexity (MEC) Metric based on Dependencies. Chidamber and Kemerer (CK) Metric based on the method and inheritance counts. Code Change Metric (or) Code Churn Metric (or) Line of code (LOC) based on number of lines includes and code adding, removing etc. Object-oriented

Revised Manuscript Received on October 20, 2019.

***** Correspondence Author

Nivetha.R, Department of Computer Science, Auxilium College, Vellore, India. Email: nivetha13.r@gmail.com

Kavitha.S, Department of Computer Science, Auxilium College, Vellore, India. Email: kavithasenthil14@yahoo.com

Metric based on the class & object of the programme.

In this paper, five common metric set is taken and improved and new software metric Cross-Entropy is introduced to measure the accurate defects in code. It is used as a code feature and it denotes the naturalness of Software component and the unnatural code (improper code) is more likely to be bug/defect code more entropic (i.e. measures the disorder feature in code). Cross-Entropy Churn Metric is a statistical explanation that is a prevalent distinction between two distribution of probability that gathers data codes that are comparatively own high probability of low entropy (i.e. less code disorder). In this situation, these codes will be assigned a (trained) linguistic model with low estimated entropy. Oppositely, uncommon (or) unnatural codes will be measured high estimated entropy [5]. So, the estimated high and low entropy is called cross-entropy churn metric to predict the defects in additional to predict the deep rich syntactic information and semantic features that extract from the code used an Abstract Syntax Tree (AST) which will detect all the nodes in the program. So, the CE-AST it will be more complementary to the existing metric set [5][6]. CE evaluated value of AST nodes results perform well in Defect Prediction. A Deep Learner in Deep Neural Network [7] is developed for C# language, this Defect Prediction framework is built in Visual Studio version 2012. This Defect Prediction Modeling is used to predict the defects for file level source code. In learning phase of defect learner, the source code is extracted from the software repository. Using Dynamic Link Library (DLL) AST is used to find the node in software component, and it map those nodes with the language model in BRNNLM is built to train and test the software components and find the defects, and combined with software metric sets and evaluated by CE-AST and measure as metric label [8]. After training the model software testing is used to develop by mining concept to analysis the data and classify the code using K-NN algorithm, is classified from the metric label. if any new features is detected it includes in CBOW and it has a buggy code or a clear code and predict the bug and gives suggestion using BPTT. This training and testing iteration process does again and again till it finds the accurate bug in the code. This is how the SDP model is built.

The scope of this paper is, 1) Defect prediction is the smart way of automating the process of testing the source code using Deep Learner in Deep Neural Network which is the advance of Machine learning which helps to find the deep features in codes. 2) Using cross-entropy churn metric as code metric to predict the defect in file-level. CE-AST it combines the Tradition metric set and the deep learner model trained & tested and evaluate the effectiveness of accurate defect prediction in code. 3) Bi-RNN are used to deep learner that trains & test each single token in code from both the end and predicts the defect in syntax and semantic information, it improves by tuning hyper parameter and by adding advanced regularization techniques such as embedding dropout (or) weight tying technique used CBOW and for further investigation into BPPTT back propagation through time algorithm for learning RNN will provide additional improvement, it seems that complementary information even to dynamic model trained with BPTT which prediction the logical error and gives the suggestion to correct the error. 4) This Defect Prediction Model advantage is reduce the developer testing work and helps to find the potential bug in code to enhance the efficiency and reduce cost and improve the software quality. 5) This Defect Prediction Model is built in Visual Studio software which minimizes the cost of getting software testing tool.

The main contribution of this paper:

- Motivated by the earlier work, this paper introduce a Cross-Entropy Churn Metric with Abstract Syntax Tree (CE-AST) as a new software code metric to typically predict the defect for file-level (contribution 1).
- An approach of Bidirectional Recurrent Neural Network Language Model (BRNNLM) based on defect prediction built a framework called Defect learner in learning phase. And five Software Traditional Metric set is measured in measuring phase. CE-AST is automatically generate the software component of NLP and Metric set and identify the bug in code (contribution 2).
- K-Nearest Neighbour classifier is used to classify the bug/defect code and a clear code, CBOW technique is used to train the NLM if new feature is detected and BPTT algorithm is used to predict the bug and suggest the potential bug in defect prediction tasks (contribution 3).

The following section of the journal is structured as follows: Section II. Related Literature work, Section III. Background of Defect Prediction Model, Section IV. Approach of Overall framework to Defect Prediction, Section V. Implementation, Section VI. Result and Analysis and Section VII. Conclusion and Future Work.

II. LITERATURE STUDY

Prediction of Software Defect have been improved an active topic for developing a good software quality by finding the defect-prone in code. So, that it reduce the test effort of the developer. Since the defects in code the data (features) are commonly include many entropic and redundant features and missing many features to address these issues. Learning model is built to find the essential code feature [1]. To find this features in code many language model is built in n-gram model rule based approach is called tokenization. It selects high level tokens in sequence and collects as vocabulary in the n-gram model and calculates the detected bug [2]. Buggram breaks a token sequence from methods into sequence of fixed length software defects prediction to devote the effort to design new metrics (features). But till now there is no one metrics is used for predicting the defects in code. Different software metric perform different function tradition metric set which help to predict the defect [3]. From the measured metric label the software component from the software repository and the features are trained & tested and classify those metric label by using machine learning algorithm [4]. To improve the prediction model Deep Brief Network (DBN) it automatically learns the semantic features from the code tokenize each vector token

using AST and train the defect prediction model with this features [5]. LSTM allows the model in the word-based description of an issue by long term context. RHN provides a deep representation of the model. In input layer is a sequence of word are taken as individual words and embedded into LSTM which helps to collects the sequence of words for long term in memory [6]. The metric is similarly matching with such distribution (or) correlation between source & target after this matching process finally matched the metric set between the source & target project and build an HDP prediction Model and predictable of targeted instances. This HDP is different project uses different metric set, to learned about defect prediction [7]. DNN is used to achieve accuracy and to train & build the SDP Model. A smali2vec approach used to produce features that capture smali.apks ' characteristics as smali2vec that extracts both token and semantic features of error documents in apks [8]. Methods they have used to measure and solve class imbalance problem and the effective features learning methods which divide each class into several sub-classes called Sub-class Discriminative Analysis (SDA) which is used to learn many features to predict the defects using metric sets. ISDA defect prediction framework provides the same work for both (WPDP) and (CPDP) class-imbalance issues [9]. The impact of automated optimization is to measure the defect prediction modules, these modules are trained and set a default setting by using CARET- optimized setting to certify the training and testing have related characters [10]. CARET optimized classifier is least stable and that are trained to use for default settings.

III. BACKGROUND

A) Language Model

Natural language (eg: speech reorganization, textual information) are human communication that helps the computer to understand the user requirement for this Natural Language Processing (NLP). The machine language and artificial intelligent helps the computer to learn the NLP. In NLP, statistical LM is important part to learn the system. The aim of the LM is to compute the joint probability of function, token (or) features in code (eg: sentence (or) a sequence of words) in a particular language. Each word is a token when a sentence is tokenized into words, same in rule based language model each sentence can also be a token if tokenized the sentence out of a paragraph. For each single text segmentation called token, to generate this LM n-gram Model is used to understand the Code to machine point of view. Probability of language model, the probability denote P, probability of sequence is S and word W. $P(S) = P(w1, w2, w3 \ldots wn)$ The probability of sequence of word will predict the high possibility of next word in a sequence (eg. Type three words in your mobile keyboard it will predict the upcoming words the 'The' 'next' 'word' is the sequence of words with the previous words it shows the next word as 'from' 'on' 'it').

Token Sequence

$$P(S)=P(w1,\ldots,wn)=P(w1)P(w2|w1)P(w3)P(w1,w2)\ldots P(wn|w1,\ldots\ldots\ldots wn-1)=\prod iP(wi|w1,\ldots wi-1)$$

Simply calculating the n-gram model with the token / features likelihood condition depends only on n-1 latest token.

$$P(S) = P(w1, w2, w3, w4) \equiv P(w4|w1, w2, w3)$$

To solve the NLP take such as machine translation and speech reorganization the Neural Language Model is a fundamental part of NLP.

UNIDIRECTIONAL RNN

Unidirectional language model is simple and it checks the size of input token values (i.e. from left to right backward pass) from the vocabulary of RNN and predict the next token output value."

Token sequence

$$P(w1,\ldots,wn)\approx\prod iP(wi)$$

BIDIRECTIONAL RNN

Bidirectional language model is difficult to discover in code the deep syntactic and semantic characteristics. It checks the layer code of the inputs from this forward (left to right) and passes backwards (right to left) in RNN. The values of forward and backward should match to find the defects in code. It checks from the vocabulary predict the output token. By using BRNN, it helps to predict accurate deep defects and it make more efficient in source code.

Token sequence

$$P(w1,\ldots,wn)\approx\prod iP(wi|wi-1)P(w1,\ldots,wn)\approx\prod iP(wi|wi-1)$$

B) Recurrent Neural Netwoek Model

The Neural Language Model is the important and basic part of many systems to solve NLP tasks (Machine translation and speech reorganization) Language Model is designing and overcoming the semantic resemblance between words to solve the language sequence modeling issue. The distributed representation of words is also called (word-embedding (or) word-vector) and the method of neural networking is used to assess the joint chance of code sequence.

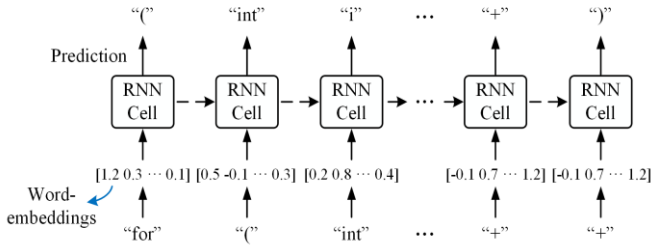

Fig.1 Unidirectional-RNN LM with Word-Embedding as Inputs

RNN in language model is like a vocabulary which collects the tokens of words from the source code. Using word-embedding technique is used to predict the possibility dissemination over the 'next' token with the collection of previous token. So, that the next token is predicted. From the word embedding layer, each input of token can be mapped each token of words into vector in set of fixed length of continuous vector (eg. Token "for" is converted into [1.2 0.3….0.1] word is converted into vector and map this token vector will be trained as a parameter to train the learning phases model. With this NLM it utilize the word embedding method and map the token

vectors for continuous line of code, Related words are likely to be a associated vector to enhance the learning of semantic language interactions. Leveraging the highly non-linear neural network to fit the probability of sequence. It leverages the high non-linear neural network to fit the occurrence probability of sequence model and it can be describing a language more flexible with the rise of deep learning (RNN) it has the capability to learn the long-term of dependence and collect it as vocabulary to analysis the token and predict the next token.

C) Cross-Entropy

In statistical principles of Cross- Entropy churn metric with AST nodes is used to calculate the difference between two possibility dissemination of **p** and **q** and identify the average amount of code used for the optimization set for the estimated real distribution p and the probability distribution of q from the collection of averaging schemes. In paper [on "Naturalness" of bug code in software] have used this code feature which denotes naturalness (i.e. highly repetitive & predictable in code) in a software component (or) instance through statistical model gives good effect and suggest the code that appears improbable (i.e. unnatural) tends to be more entropic (measuring of disorder feature in code) and this code are more likely to be buggy. In this situation there cross-entropy need to measure but the distribution of **p** is unknown (unnatural code). In collection of data codes are relatively common and own high probability of appearance (low entropy) (i.e. less disorder in code). Thus language model is developed on the basis of the training & testing set assigned as T in the language training model assign this code to the low entropy, in oppositely it assign uncommon (or) unnatural code will be measure high estimated entropy in testing set. Where p is the (unknown unnatural) test data set in any software corpus from the real distribution of words. In training data set **q** is the (known naturalness) from software corps that distributed the words for prediction. Here the high and low entropy of probability distribution from the corpus used cross-entropy in a statistical language model to evaluate the code naturalness for predicting the bug.

$$H(T,q) = -\sum_{i=1}^{N} \frac{1}{N} \log_2 q(x_i)$$

Where **N** is the size of the test set, and **q(x)** is the probability of event x estimated from the training set. The calculation of the amount is over N. It's a Monte Carlo estimate of the true cross entropy, where the test set is treated as samples from **p(x).**

D) Problem Definition

i) To learn the predictabilities in source code from software source Unidirectional Recurrent Neural Network Language Model (Uni-RNNLM) is used. It predicts the word in static (one way) that is from left to right where it predicted defects is not accurate. ii) Need even more accurate Deep Learner to find the syntactic and semantic feature information. iii) In previous work it predicts the results in static error there is no suggestion to clear the defect bugs. iv) RNN can capture truly simple and limited context information as cache model to predict the defect. v) It consumes more cost and time to build an efficient defect prediction model.

IV. APPROACH

In this section, presents a detail work of proposed Defect Learner framework.

A) Overall Framework of Defect Learner

The overview of proposed Defect Learner is to design a defect learner framework which automatically generates the cross-entropy metric which scores the software component from the software repository for accurate software defect prediction in source code. There are three phases of framework contains: (phases 1) Learning Phases of Neural Language Model, (phase 2) Measuring phases of software metrics, (phases 3) Prediction Phases. To perform this phases first step is to extract the software component (packages, files (or) methods) from software repository.

In learning phase (1), the aim is to build a Neural Language Model (NLM) for training & testing the LM, to learn the common patterns (or) usage regularities of programming language by mining software repository. The source code are extracted from the software repository and it transfer out data pre-processing on the collection of projects code Lines such as removing, commenting and tokenizing using the AST sequence node to find data about syntax and semantic information. Next step is mapping word-embedding methods the tokens sequence can be represented as a set of real valued vectors.

Then for building a Bi-directional RNNLM is used to train the task of predicting next word in sequence of words, but that their performance can be improved by hyper parameter (adjustable) and by adding advanced regularization techniques such as embedding dropout (or) weight tying the sequence of words are predicted by using Continues Bag of Words (CBOW) technique. Based on BRNNLM used Long and Short Term Memory (LSTM) algorithm This enables the input token vectors to learn the deep sematic characteristics.

At the same time the source code is extracted from software repository and the components are measured using traditional metric sets. By using CE-AST it evaluate the probability distribution between the NLM values and the traditional metric sets values are set as a metric label as buggy (or) clear code, if any new feature is detected it set as new label and it send to learning phases.

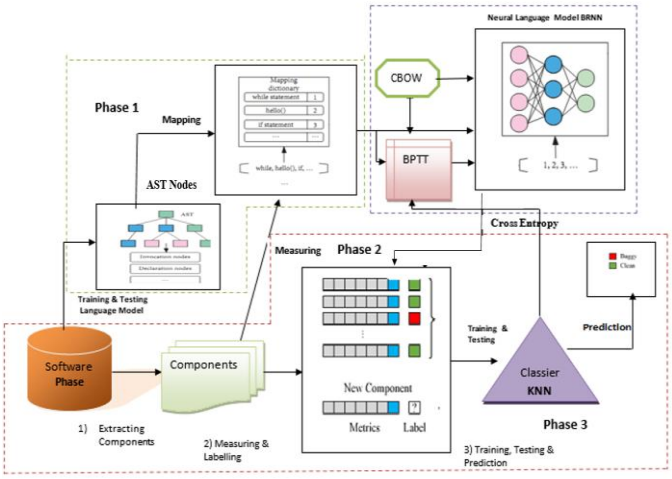

Fig. 2 Overall Framework of Defect Learner

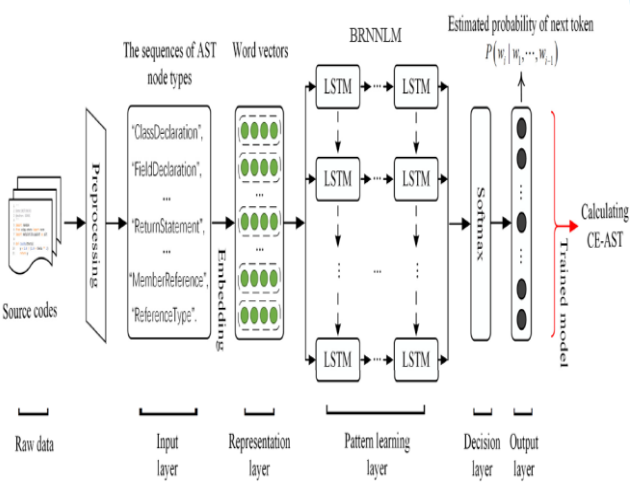

Fig. 3 Bidirectional RNNLM for Learning Phase

B) Neural Language Model

In this defect learner training & testing language model is built in DNN, based on Bi-directional RNNLM's, LSTM is a technique in RNN used in the architecture of deep learner and for calculation CE-AST is used. This BRNNLM's is designed, as a defect learner for C# programming language processing. In this language model, first things is to extract the software components (i.e. source code file) from the software repository create a corpus (i.e. collection of data), to analysis this raw data from the corpus of source code, data pre-processing is the first step to generate token sequence for language learning and removes blank links and comments. After data processing next step is Input layer, the sequence of AST node types is used to find the rich syntax information and semantic information.

For example: class declaration, return statement, control statement. Next step is to map those nodes into integer that is embedding the AST node sequence input layer to representation layer; here each AST nodes (token) are mapped into integer called word vector in representation layer. Here the goal of training NLM is to learn the common patterns (or) regularities of code. Then build a bi-directional LSTM based RNN is used to learn the syntactic and semantic information from the input vector.

Basic RNN perform well when being trained on the task of predicting the next word in sequence of words, but it sets a fitted parameter for training model. The performance can be improved by tuning hyper parameter (i.e. adjustable parameter that must be tuned in order to obtain a model with optimal parameter by adding regularization techniques such as embedding dropout (or) weight tying (i.e. CBOW) this layer called pattern learning.

In decision layer (Softmax) classifier is connected from pattern learning layer to predict the next token for each single token and process as predicting token sequence and obtained probability distribution can be used to define like hood of occurrence of sequence in the output layer. After training language model is used to measure the CE-AST features with software traditional metric sets.

1. Data Pre-Processing

Data pre-processing is the first step to execute source code (raw data) done before the training the NLM. This is the specific steps to follow

STEP 1 To produce an actual code to remove the comments and blank line, recollecting actual line of code.

STEP 2 Parse (i.e.) analysing a string in the program code it breaks the data into smaller chunks for following a set of rules, so it will be easy to transmit the code. According to C# language specification, the source code is parsed into AST node type sequence for example: class declaration, file declaration, continues statement. In this paper used an open-source package called (C#) language to parse the C# file-level source code and used all AST node sequence type is used in input layer.

STEP 3 Create a vocabulary from the input layer the merge the AST nodes type sequence file and word embed with the representation layer, here this AST node (token) are mapped into integer called word vector. These unique token counts and their frequencies are built as vocabulary and it size denoted

as V.

STEP 4 Replace tokens to control the complexity of the mode, need to remove the low. Frequency token. In this paper the total number of vocabulary is set as V, so, the top (vocabulary of token) V.

STEP 5 Built a training set & testing set for mode validation, in these step long token fragments are cut into a series of sequences with equal length for feeding the language model.

2. Rnn Building

The language model is the part of Deep Neural Network which helps to learn the deep code feature for given source code. And this network structure consist of input layer (the sequence of AST nodes), Representation Layer (word2vector), Pattern learning layer (BRNNLM), Decision layer (Softmax classification) and in output layer (finds the probability of next token).

The pre-processed information characteristics are gathered in the input layer and the code is parsed in AST node (eg: control statement, method

declaration) these nodes are collected from the Dynamic Link Library. Each line of code are assigned ad sequences of AST nodes every single tokens (or) features (eg. 'for', 'if', 'static') are word and this single words are embedded with the word vector (i.e. word2vector) in representation layer. It map each code token into a high dimensional real-valued vector, in which tokens with similar contents will be assigned as similar vector. With word mapping is collected as a vocabulary in Neural Language Model each single word are fixed as high frequency of token vector and low frequency of token vector are set as vocabulary and feed into BRNNLM to train the model from the given source code.

In pattern layer, the BRNNLM learns the deep syntactic & sematic features in code. Using Bidirectional it checks each line of code from forward and backward for advance learning regularization hyper-parameter techniques is used to learn the code from top to bottom. LSTM cell is one of the most popular algorithm based on RNN to understand the sequence of dependencies, has been widely adopted to build in RNN.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$
$$h_t = o_t \odot \tanh(c_t)$$

This method use as a memory gates to control its inputs gate i, output gate g, forgot gate f, x_1 input vector, c_1 cell state, h_1 hidden states. Where weight matrix. W and b bias vector. $\sigma(x) = 1/(1+e^{-x})$ and $\tanh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$ are activation function; $\odot$ denotes wise multiplication of elements. Using LSTM, the effects of disappearing and exploding gradient can be effectively reduced and the long-term learning capacity to train the NLM improved.

Finally, it combines information in the decision lahyer using the softmax tool to calculate and predict the next token of the input code line with an estimated distribution of probability that predicts the next word in output. As complete forecast error falls, the languuae model may be considered to have learned to use patterns from the specified C #languages and to train the language model.

3. Training And Testing The Language Model

While training and testing the network architecture it often encounter many troubles when learning and measuring the code to predict, troubles such as over-fitting, gradient explosion (or) gradient vanishing. Therefore, numerous strategies for optimization are adopted in this work.

DROPOUT

Neural Network training with main strategy concept to drop the units randomly. Dropout unit is an effective regularization technique to prevent over-fitting hyper parameter tying helps to adjust the parameter.

GRADIENT CLIPPING

It can be used to minimize the error terms by changing each weight in proportion to derivative error with respected non-linear activation function.

ADAPTIVE LEARNING RATE

The learning rate l_{rate} is adjusted dynamically with the number of training epochs (iteration and it use to optimize the network.

C) Estimation of Software Component

Once the learning phase (1) is completed, the same process is done in phase (2) Measuring phases. The software component is extracted from the software repository. Each software component the source code is measured with the five different software traditional metric sets, and this metrics are used to find common features for measuring the source code. It measures each lines of code and fix the length of each lines and this measured inputs are kept separately. As a results, the average of cross-entropy with AST (CE-AST) as a new code churn metric. It combines the NLM values with the measured software traditional metric sets values. So, the cross-entropy churn metric which helps to evaluate the probability distribution between NLM and metric sets.

By training the source code with this phases it determine the accurate defective-proneness and finds the buggy code and clear code features is measured (or) detected it set as a new component (i.e. this new features may not present in the training & testing process).

D) Performing Defect Prediction

In this paper, four common classifier is taken for prediction support vector machine (SVM), Random Forest (RF), Naïve Bayes (NB), Logistic Regression (LR) are used in previous defect prediction and compared with K-NN to improvise the prediction task.

To calculate the performance of prediction models with different software metric sets classifier K-NN and accessed via confusion matrix to overcome the typical binary classification problem four widely measures are adopted Precision, Recall, F1, Area Under Curve (AUC).

Table I Confusion Matrix

Actual	Prediction	
	Predicted Buggy	*Predicted Clean*
True Buggy	TP	FN
True Clean	FP	TN

i) Precision

Precision is to measure the ratio of positive value (bug) from the detected metric label to predict the actual value.

Precision = TP / (TP+FP)

ii) Recall

Recall is to measure the ratio of true positive values from the metric label (bug/clear) code to predict the actual true positive value.

Recall = TP/ (TP+FN)

iii) F1

It combines both precision & recall using harmonic mean formula.

$$\mathbf{F1} = \frac{\mathbf{2\ X\ Precision\ X\ Recall}}{\mathbf{Precision\ X\ Recall}}$$

iv) AUC

It measures the Receiver Operating Characteristic (ROC).
X axis represent

FPR = FP / (FP+TN)

Y axis represents

TPR = TP / (TP+FN)

V. IMPLEMENTATION

A Deep Learner in DNN is developed for C# language; this defect prediction framework is built in Visual Studio version 2012. This defect prediction modeling is used to predict the defects for file level source code. In learning phase of defect learner, the source code is extracted from the software repository. Using Dynamic Link Library (DLL) AST is used to find the node in software component, and it map those nodes with the language model in BRNNLM is built to train and test the software components and find the defects, and combined with software metric sets and evaluated by CE-AST and measure as metric label. After training the model software testing is used to develop by mining concept to analysis the data and classify the code using K-NN algorithm, has a buggy code or a clear code and predict the bug and gives suggestion using BPTT. This is how the software defect prediction model is built.

A) Phase 1 Learning Phases For Building Language Mode

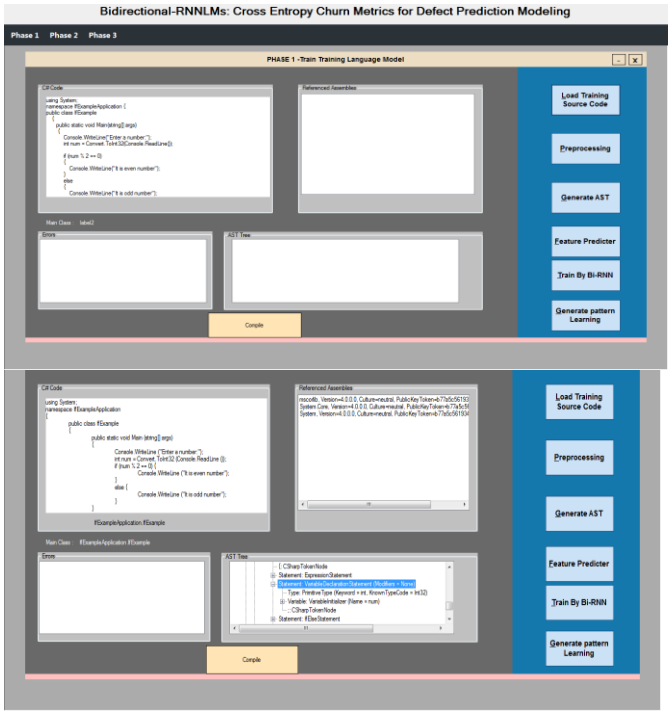

Fig. 5 Generate AST Node

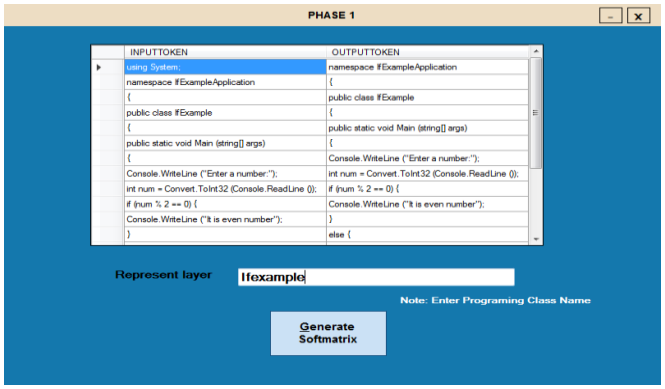

Fig. 6 Training the BRNNLM

B) Phase 2 Measuring Phases Using Cross-Entropy Metric

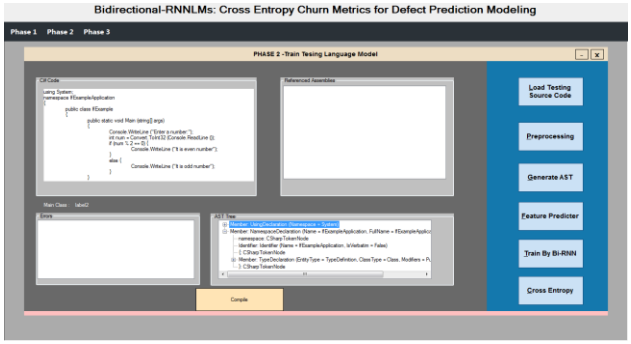

Fig. 7 Measuring Phase using Cross-Entropy Metric

C) Phase 3 Prediction Phases Using Knn, Cbow And Bptt

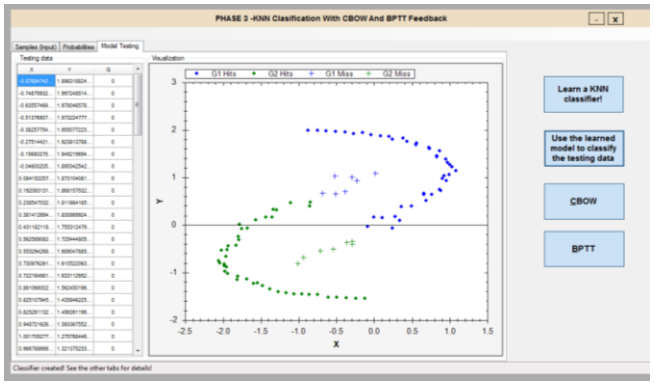

Fig. 8 Predicts the Bug

VI. RESULT AND ANALYSIS

A) Result for Defect Learner

Each iteration (epoch) process of training the NLM to learn the entire features of data sets and this number of iteration of training the LM denoted as x-axis. For testing set loss of curve function score it is the inverse probability of language model. Each new number of testing features denotes y-axis and it is the best language model to predict the unseen test set to find the deep semantic features.

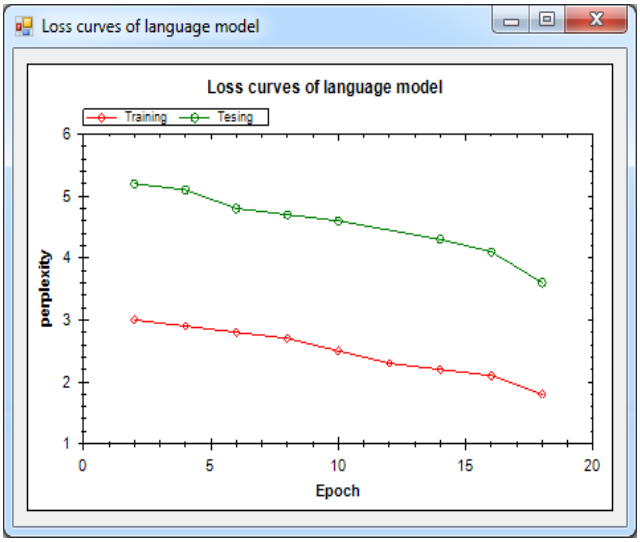

Fig. 9 Loss Curves of Language Model

In the above graph the loss curves of language model for both train curves and testing curves are differed from training set to testing set, which means it meet gradually as time goes on through 18 epochs of training, the model is finally get an average of 2.923 perplexity on training set and 5.218 perplexity on testing set. The initial perplexity is more than 2.295. It implies that the model, gets full trained and has learned the learned the common patterns regularities in C# code.

B) Different Software Traditional Metric Sets With Cross-Entropy Churn Metric Comparison

The contribution of five different software traditional metric set is compared with cross-entropy and to evaluate a metric set with cross-entropy with file level source code to find the performance improvement of cross entropy metric. The software traditional metric sets are Halstead metric, Mccabe Essential Complexity (MEC), Chidamber and Kemerer (CK) metric, Line of Code (LOC), Object oriented metric. Which characterize different features of software component to give a comprehensive measurement, cross-entropy is combined with five different metric sets together to feed the prediction model.

Cross entropy metric brings improvement in each prediction task, no matter using which measures (or) with which metric sets. For example: Precision measure the cross-entropy contributes an average of 4.81% absolute improvement and 11.57% relative improvement with existing metric sets. The minimum increases are the CK metric and maximum increase is the LOC metric. So, it considers the reason of this two aspects of results (absolute increase and relative increase). Since traditional metric sets are used different measurement mechanisms, cross entropy has different degrees of complementary to them. If the size of defect dataset is large enough, the classifier is more difficult to train with more features input. Results with different measurement are complementary to traditional metric.

Table II the Cross-Entropy Churn Metric Performance With Different Software Metric Sets

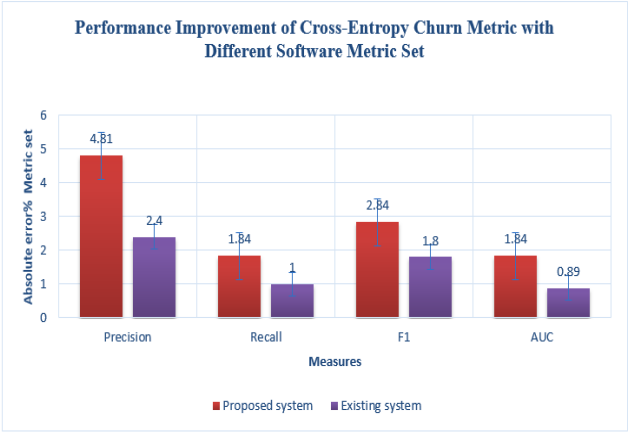

Fig. 10 Absolute Increase Performance of Prediction with Cross-Entropy

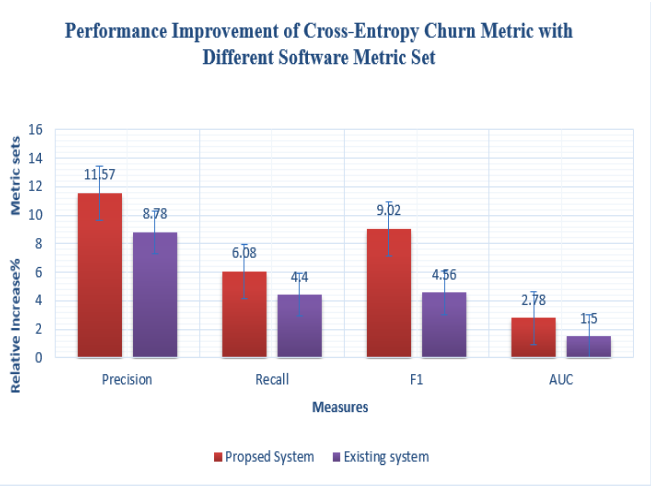

Fig. 11 Relative Increase Performance of Prediction with Cross-Entropy

C) Comparison between the Performances Changes in Existing Classifier with K-Nn Classifier in Cross-Entropy Metric

The performance changes of different classifier with cross-entropy. K-NN classifier is compared with Support Vector Machine (SVM), Logistic Regression (LR), Navi Bayes (NB), and Random Forests (RF). The results are calculated with Harmonic means of minimum and maximum classify the results as two parts. The performance is changed when comparing with four different classifier, cross-entropy metric is more contributed with classifier K-NN performance is increased in absolute and relative respectively in Precision 4.81%, Recall 1.84%, F1 2.84%, AUC 1.81%.

Table III The Performance Improvement Of K-Nn With Different Classifier

Measure	Metric Set Name	Metric Set	Metric Set Improved	Absolute Increase	Relative Increase
Precision	SVM	0.4032	0.4303	2.72%	6.98%
	LR	0.5522	0.5718	1.96%	4.45%
	NB	0.4116	0.5273	11.57%	9.30%
	RF	0.5057	0.5355	2.98%	5.63%
	K-NN	0.4681	0.5162	**4.81%**	**11.57%**
Recall	SVM	0.2744	0.2832	0.68%	2.09%
	LR	0.3223	0.3359	0.35%	2.77%
	NB	0.2800	0.3437	1.36%	4.43%
	RF	0.4020	0.3395	2.07%	7.15%
	K-NN	0.3197	0.3381	**1.84%**	**6.08%**
F1	SVM	0.2882	0.3005	1.33%	0.91%
	LR	0.3603	0.3738	1.63%	0.91%
	NB	0.2947	0.3875	9.30%	4.11%
	RF	0.4401	0.3468	8.07%	4.76%
	K-NN	0.3458	0.3742	**2.84%**	**9.02%**
AUC	SVM	0.6759	0.6927	1.27%	0.40%
	LR	0.7149	0.7112	0.40%	0.56%
	NB	0.5862	0.6303	4.95%	7.45%
	RF	0.6979	0.7101	2.41%	1.78%
	K-NN	0.6680	0.6861	**1.81%**	**2.78%**

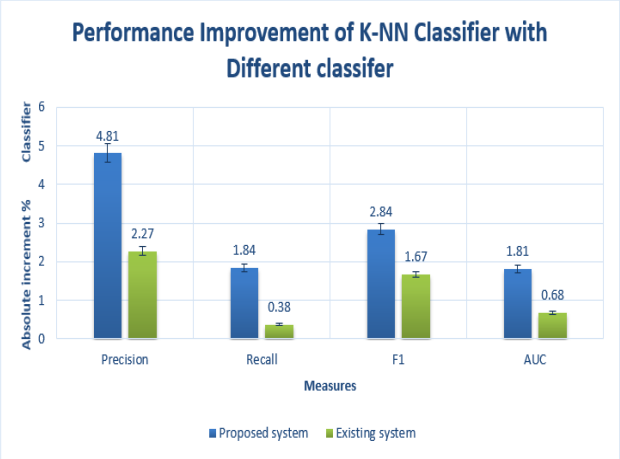

Fig. 12. Absolute Increase Performance Improvement of K-NN Classifier with Different Classifier

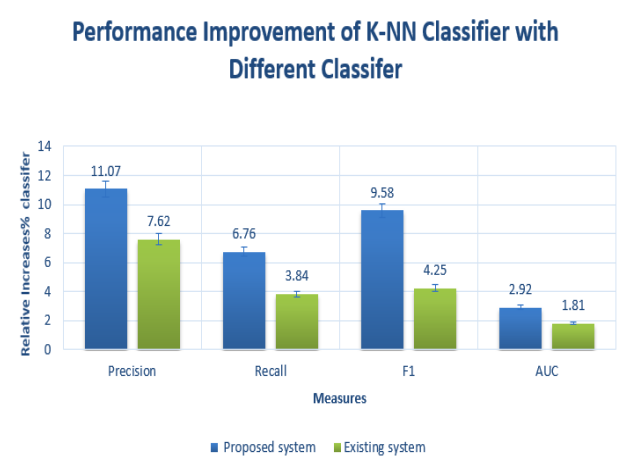

Fig. 13. Relative Increase Performance of Improvement of K-NN Classifier with Different Classifier

VII. CONCLUSION AND FUTURE WORK

A) Conclusion

In recent study's, to improve a software quality in code many defect prediction modelling is introduced to predict the bug in code and use different software metrics to measure the code which tends to be more entropic. Based on these findings, this paper introduces a defect prediction model for C# language. This model is built in Deep Neural Network of Bidirectional RNNLM is to learn the deep semantic feature from the source code and train and test the language model. Same source code is measured with software traditional metric sets. Both the language model and metric set are automatically evaluated the defect-proneness with effectiveness of CE-AST. This cross-entropy churn metric with Abstract Syntax Tree produces better accurate defects-proneness and complement then the existing metric set. For, additional improvement CBOW, BPTT and K-NN algorithms are used to find the accurate defects prediction in code. The advantage of using Visual studio it develops the defect prediction model in software. So, that it reduce the software tool cost. As a result it shows the cross entropy churn metric with AST (CE-AST) has more discrimination for defect classification with five common metric set and improves the defect prediction model with file level of source code. It brings the improvement of Precision 4.8%, Recall 2.4% and F1 8.5% by average of defect prediction model.

In conclusion, these BRNNLM and CE-AST as a defect learner in defect prediction model are suggested to find accurate prediction in the source code and improves the quality of software and it complementary with the existing defect prediction model.

B) Future Work

In future, Defect Prediction Model would like to develop for more projects by using many metric sets to measure the LOC and to reduce the bug in code. So, this thesis finds the defect in file level source code within the project, it is more challenging to predict the defect for cross-project.

This thesis in Defect Prediction Model is built for C# language still many metric sets and many Deep Neural Network Language Model are needed to add in Defect Prediction Model to build a Defect Perdition tool for C# language.

REFERENCE

1. Li, J., He, P., Zhu, J., & Lyu, M. R. (2017, July). Software defect prediction via convolutional neural network. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (pp. 318-328). IEEE.
2. Wang, S., Chollak, D., Movshovitz-Attias, D., & Tan, L. (2016, August). Bugram: bug detection with n-gram language models. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (pp. 708-719). ACM.
3. Peng, F., & Schuurmans, D. (2018, April). Combining naive Bayes and n-gram language models for text classification. In *European Conference on Information Retrieval* (pp. 335-350). Springer, Berlin, Heidelberg.

4. Punitha, K., & Chitra, S. (2013, February). Software defect prediction using software metrics-A survey. In *2013 International Conference on Information Communication and Embedded Systems (ICICES)* (pp. 555-558). IEEE.
5. Wang, S., Liu, T., & Tan, L. (2016, May). Automatically learning semantic features for defect prediction. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (pp. 297-308). IEEE.
6. Choetkiertikul, M., Dam, H. K., Tran, T., Pham, T. T. M., Ghose, A., & Menzies, T. (2018). A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*.
7. Nam, J., Fu, W., Kim, S., Menzies, T., & Tan, L. (2017). Heterogeneous defect prediction. *IEEE Transactions on Software Engineering*, *44*(9), 874-896.
8. Dong, F., Wang, J., Li, Q., Xu, G., & Zhang, S. (2018). Defect prediction in android binary executables using deep neural network. *Wireless Personal Communications*, *102*(3), 2261-2285.
9. Jing, X. Y., Wu, F., Dong, X., & Xu, B. (2016). An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Transactions on Software Engineering*, *43*(4), 321-339.
10. Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2016, May). Automated parameter optimization of classification techniques for defect prediction models. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (pp. 321-332). IEEE.

AUTHORS PROFILE

R.NIVETHA is currently studying as a Research Scholar M.Phil (Computer Science) research domain in Software Quality of Assurance in Department of Computer Science in Auxilium College (Autonomous), Gandhi Nagar Vellore-632006. She has obtained her MCA Degree from Vellore Institute of Technology (Vellore campus), Vellore-632014. She has presented many paper in International Conferences and National Conference, and also published her papers in leading Journals. She has participated in many seminars and workshops in Computer Science. She have done projects in Software Engineering, Image Processing and Data Mining.

KAVITHA.S is currently working as Associate Professor in Department of Computer Science, Auxilium College, (Autonomous), and Gandhi Nagar Vellore-632006. .She has obtained her MCA Degree in Nehru memorial college, Trichy. M.Phil (Computer Science) in Mother Teresa University. At present I am pursuing Ph.D in Periyar University. I published 10 in international journal and also presented paper in national and international. . She has guided Around 30 M.Phil students.