[1] T Shathish Kumar

[2]B Booba

Rand-Index Target Projective Gradient Deep Belief Network for Software Fault Prediction

Abstract: - Predicting defective software modules before testing is a valuable operation that reduces time and cost of software testing. Source code fault prediction plays a vital role in improving software quality that effectively assists in optimization testing resource allocation. Several machine learning and ensemble learning techniques has been extensively evolved over the recent few years to predict defect at an early stage. These techniques made predictions based on historical defect data, the software metrics. Nevertheless, the time efficient and accurate fault predictions are the major challenging tasks that yet have to be addressed. In order to ensure accurate software fault prediction, a method called Rand-Index Target Projective Gradient Deep Belief Network (RTPGDBN) is designed. The proposed RTPGDBN method comprises of three processes namely acquiring JAVA packages, software metric selection and classification. First, the number of JAVA packages is used as input from the dataset. Second with the JAVA packages obtained as input, Rand similarity indexive target projection function is applied for selecting the most significant software metrics in order to minimize time complexity of fault prediction. Third, with the selected metrics, classification is performed using Tversky Gradient Deep Belief neural network. Also, gradient descent function is applied to minimize classification error, therefore ensuring accurate software fault classification results obtained at the output layer. Experimental setup of proposed RPGDDBN and existing methods are implemented in Java language and the dataset collected from smell prediction replication package. Performance analysis is carried out with different quantitative metrics such as accuracy, precision, recall, F-measure, and time complexity and space complexity. Through extensive experiments on repository data, experimental results indicate that our RPGDDBN method outperforms two state-of-the-art defect detection methods in terms of different performance metrics.

Keywords: Software Fault Prediction, Rand-Index, Target Projective, Tversky, Gradient Descent, Deep Belief Network

I. INTRODUCTION

Encountering a high-quality software design is constrained by experiencing software design that definitely positively controls the software produce quality features, like, affinity, sustainability, reliability, security and scalability. As each characteristics of quality has a greater influence on others, it is essential to act in accordance with the back and forth techniques, i.e., concentrating on the quality attributes that are interpreted by the circumstances and tolerance of others.

A Hybrid Approach to detect Large Class Bad Smell (HA-LCBS) was proposed in [1] to detect large class bad smell. Here, Genetic Algorithm (GA) was employed for automating module detection composition including cohesion and coupling metric. Following which the resulting paired value were passed to deep learning technique with the purpose of automating large class bad smell. As a result, accuracy with which bad smell detection were made was found to be significant. Nevertheless, owing to the immediate reliance between improved and consistent code, earlier machine learning based techniques cut out to model prolonged and deep dependencies, resulting in high false positive. To focus on this aspect, a Bug Prediction (BUGPRE) method was proposed in [2] to address the two issues relating to prolonged and deep dependencies. Here, propagation tree-based associated analysis was employed for performing effective defect prediction with the purpose of acquiring the changed modules in the current version. In addition, BugPre contrived capitalizing code context dependences and employed a graph convolutional neural network for learning representative characteristics of code. As a result, defect prediction potentiality was improved when updates were noticed during changes in version, therefore improving both accuracy and F1-score.

[1]* T Shathish Kumar, Department of Computer and Engineering, Vels Institute of Science, Technology & Advanced Studies (VISTAS),

Vels University, Chennai, Tamil Nadu, India

2 Department of Computer and Engineering, Vels Institute of Science, Technology & Advanced Studies (VISTAS),

Vels University, Chennai, Tamil Nadu, India

The quality and reliability of software specifically depend on eliminating defects in software. The conventional mechanism for identifying software defects is by means of testing and performing reviews, but necessitates time and effort. On contrary, automatic software defect prediction may assists developers in enhancing quality of code at a reduced cost upon comparison to manual model. Hence, Software Defect Prediction (SDP) has become a significant research area of topic in recent years. A holistic review of deep learning techniques for software defect prediction was investigated in [3]. Historical data is considered as the gold mine for predicting software defect with greater level of accuracy and confidence. This is owing to the reason that the software defect prediction data is available in its raw form, hence found to be not suitable as far as machine learning applications are concerned.

A robust tool was introduced in [4] that initially with the aid of raw data collection following with validation finally predicted the defect by means of machine learning. Multiple expert learning systems were integrated in [5] to decide on the faulty modules with elevated accuracy rate. The process of software defect-prediction comprises of metrics extraction and designing full proof defect prediction. Over the past few years, most conventional software defect-prediction methods employ machine learning for constructing defect prediction following which the extracted metrics were employed as model features. Nevertheless, the conventional metrics specifically concentrated on complexity involved in code designing. Also strong differentiation between semantics and semantic information in the source code were not made.

To explore information between sequences and also to learn code semantics in addition to syntactic structure, a method call sequence was presented in [6] that preserve the code context structure information. With this type of design, mean absolute error rate was found to be comparatively less. Yet another deep learning based prediction method was proposed in [7] to focus on the accuracy and time aspect. Since recently deep learning techniques have achieved significant results in several areas of applications, there is a requirement to apply for all type of problems, i.e., software defect prediction. In this work, the objective is to evaluate the performance of deep belief neural network and the effect of gradient function on defect prediction methods and also compare these results with the performance of deep learning algorithms. We report our findings on the comparison of the software defect prediction performance of two learning algorithms (i.e., HA-LCBS and BUGPRE). Experiments were performed on publicly available dataset.

A. Contributory remarks

To design Rand similarity indexive target projection-based feature selection where the time and accuracy is said to be improved by means of rand similarity index function and target projection. The problem is formulated as a defective and non-defective package modeled at different time instances owing to different class data samples and features employed for testing and accordingly results are evaluated. The contributing remarks of Rand-Index Target Projective Gradient Deep Belief Network (RTPGDBN) are provided below.

- To design efficient method for software fault prediction using Rand-Index Target Projective Gradient Deep Belief Network (RTPGDBN) method.
- To propose Rand similarity indexive target projection-based feature selection for obtaining computationally efficient software metrics.
- To present an algorithm called Tversky Gradient Deep Belief Neural Network-based classifier that with the aid of Tversky Gradient function and soft step activation function improves precision and recall with minimum complexity.
- Experiments are performed conducted on the smell prediction replication package and the results show that our method can efficiently enhance the prediction performance of defect prediction method with improved precision, recall and has the lowest complexity and time in terms of comparison with other methods.

B. Outline of the paper

The remainder of this paper is organized as follows: Section 2 introduces related work on traditional software metric defect prediction algorithms, deep learning based prediction and machine learning based prediction. Section 3 describes the proposed Rand-Index Target Projective Gradient Deep Belief Network (RTPGDBN) method. Section 4 presents the experimental setup and parameter settings, followed by discussion with the aid of table and graph in Section 5. Section 6 summarizes our work.

II. RELATED WORKS

As far as early phases of software evolution life cycle are concerned, one of the sophisticated means of defect identification is software defect prediction. This early warning mechanism can assist in discarding defects present in software and produce cost efficient and quality software products. However, the imbalanced nature of data remains the major concerns for software module defect prediction. A novel hybrid method called, Hellinger net model was designed in [9] that addressed the issues relating to imbalanced learning for enhancing software module defect prediction. Yet another method to focus on the accuracy aspect employing graph representation learning was presented in [10]. A case study of software defect prediction to explore feasibility of employing static software metrics were investigated in [11]. Yet another empirical study on the mechanism for software defect prediction using classification was discussed in [12].

The core objective of a software project remains in carrying out the anticipated performance while assigning the indispensable measure of quality on time and within a defined budget. From the angle of software projects evolved over the past few years, the intricacy in developing software has risen owing to the expanded number of customer requirements. This complexity has made it more laborious and cumbersome to achieve their main objectives. Ensemble machine learning techniques and RNN-based deep learning was applied in [13] for software fault prediction. Nevertheless, few studies considered the influence of time factors on prediction results. An improved Elman neural network was presented in [14] for improving the defect prediction adaptability to the time-varying features.

Most of the prevailing defect prediction methods are designed on the basis of lines of code, stack reference depth for defect prediction. However, it did not take into consideration the semantic features into account that in turn would result in unsatisfactory prediction results. In [15], convolutional neural network was designed with the purpose of mining semantic features towards accurate software defect prediction. Yet another enhanced random forest technique was applied in [16] for defective system prediction. Also with defect density prediction before module testing is said to be laborious and time consuming, decision makers require to construct a prediction method that can assist in defective module detection. With this type of design would result in minimizing both the testing cost and enhance resource utilization testing significantly. However, data sparsity involved in defect density prediction was not made. To address on this issue, deep learning was applied in [17] to handle data sparsity in defect density. Yet another graph-based ML was presented in [18] by taking into consideration CNN for ensuring accuracy involved in prediction.

The prevailing software defect prediction practiced on file level cannot predict failures in an accurate fashion. To solve this issue, a novel technique combining statement level granularity and attention based LSTM was designed in [19] for predicting defects in statement-level. To reduce the complexity, a graph based ML technique for defect prediction was presented in [20]. Though certain works provided in the literature address on accuracy aspects, certain others focuses on precision and recall. In this work, a method called, Rand-Index Target Projective Gradient Deep Belief Network (RTPGDBN) is proposed. The elaborate description of RTPGDBN is provided in the following sections.

III. METHODOLOGY

This section formulates a new method to software defect prediction by applying Tversky Gradient Deep Belief neural network for automating module detection composition. Our proposed RPGDDBN method is illustrated in figure 1 includes three steps: 1) obtaining metrics data and repository data for extracting the JAVA packages, 2) selecting significant software metrics which reveal the behavior of programs, and 3) applying a Tversky Gradient Deep Belief neural network on selected significant software metrics for ensuring accurate software fault classification. Figure 1 shows the structure of RPGDDBN method.

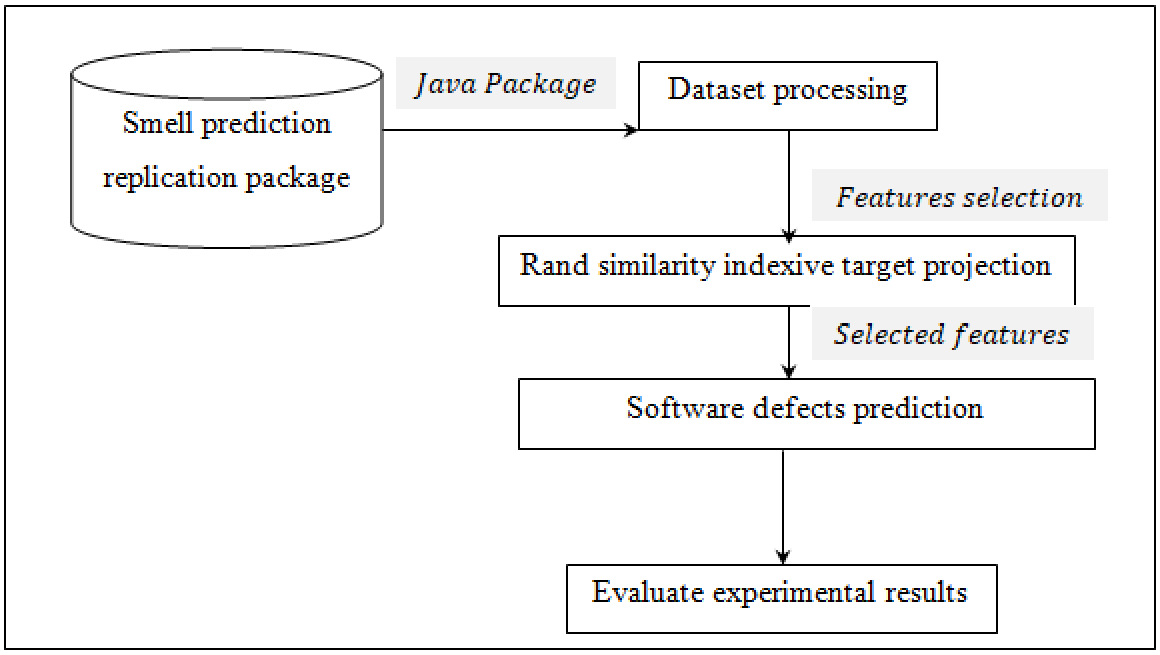
Smell prediction replication package
Java Package
Dataset processing
Features selection
Rand similarity indexive target projection
Selected features
Software defects prediction
Evaluate experimental results

Figure 1 Structure of RPGDDBN method

As shown in the above figure, in the first, step, design smell and metrics are extracted from the corresponding JAVA classes. Next, in the second step, Rand similarity indexive target projection is applied to obtain the most significant software metrics of a program. Finally, we leverage a powerful Tversky Gradient Deep Belief neural network to automatically build predictive models based on gradient descent function for accurate and precise classification. The elaborate description of the RPGDDBN method is provided in the following sections.

A. Dataset details

The smell prediction replication package dataset employing metrics-data and repository-data validated in our work is one of the most common benchmark datasets extensively utilized for software defect prediction analysis. In this work, for each system, both metrics-data and repository data containing design smell and metrics in addition to commit sequences are manipulated to train and test the proposed method. The smell prediction replication package dataset is obtained from https://drive.google.com/drive/folders/101QbQ-TtQpyZa-APCFo4hCGAc_c-g6-Y. The design smells are reported in the columns ranging between 5 and 23, whereas the product metrics are reported in columns ranging between 24 and 53. Finally, the last two columns are discarded as they refer to the type of artifact.

B. Rand similarity indexive target projection-based feature selection

The magnitude of the data employed in training the neural network can have certain influences on the neural network performance. With input data possessing fewer variables, the classification system is said to be faster with comparatively better convergence speed. Therefore, the magnitude of defective metrics should be reduced. Hence, software metric (i.e., feature) selection is extensively employed in picking out the most significant software metric to train the neural network.In this work, Rand similarity indexive target projection-based feature selection model is used to minimize the number of data by selecting the most important features. The Rand similarity indexive target projection is used to reduce the number of input variables and to transform the original defect dataset to select the smallest subset of data. With the hypothesis that there are ‘M*N’ software packagesor files in a software system where each software package is represented with ‘(P_i,P_j),where i,j∈ M*N’, a correlation matrix ‘CM’ of size ‘M*N’ is generated that exhibits the source code similarities among the packages. This correlation matrix is initially formulated as given below.

$$CM = \begin{bmatrix} CM(P_1,P_1) & CM(P_1,P_2) & \dots & CM(P_1,P_N) \\ CM(P_2,P_1) & CM(P_2,P_2) & \dots & CM(P_2,P_N) \\ \dots & \dots & \dots & \dots \\ CM(P_M,P_1) & CM(P_M,P_2) & \dots & CM(P_M,P_N) \end{bmatrix} \quad (1)$$

From the above equation (1), the correlation matrix 'CM' is formulated by taking into consideration '$P_M * P_N$' packages for simulation. Then, given 'n' element set '$S = \{S_1, S_2, \dots, S_n\}$' and two portions of '$S$' to equate or compare '$A = \{A_1, A_2, \dots, A_i\}$' a portion of '$S$' into '$i$' subsets and '$B = \{B_1, B_2, \dots, B_j\}$' a portion of '$S$' into '$j$' subsets define the following hypothesis.

$$H0\colon software\ metrics\ pairs\ in\ S\ that\ are\ in\ same\ subset\ A\ and\ same\ subset\ B(ss)$$

$$H1\colon software\ metrics\ pairs\ in\ S\ that\ are\ in\ \ diff\ subset\ A\ and\ diff\ subset\ B \rightarrow (dd)$$

$$H2\colon software\ metrics\ pairs\ in\ S\ that\ are\ in\ \ same\ subset\ A\ and\ diff\ subset\ B \rightarrow (sd)$$

$$H3\colon software\ metrics\ pairs\ in\ S\ that\ are\ in\ \ diff\ subset\ A\ and\ same\ subset\ B \rightarrow (ds)$$

Based on the above hypotheses, '$H0$', '$H1$', '$H2$' and '$H3$', the Random Similarity Index is mathematically represented as given below.

$$RI = \frac{ss+dd}{ss+dd+sd+ds} = \frac{ss+dd}{\binom{n}{2}} \quad (2)$$

From the above formulate (2), '$(ss+dd)$' symbolizes the magnitude of agreements between 'A' and 'B', and '$(sd+ds)$' symbolizes the magnitude of disagreements between 'A' and 'B' respectively. Also as the denominator represents the total numbers of pairs, the Random Similarity Index represents the repetition of incident of agreements over the total sample instances. Then for each package 'P_i', correlation to defective package 'CDP_i' and correlation to non-defective package '$CNDP_i$' is measured as given below.

$$CDP_i = \sum_{j=1}^{N} SM(P_i, P_j) \quad (3)$$

From the above equation (3), '$SM(P_i, P_j)$' is taken into consideration if 'P_j' is defective and in a similar manner, '$SM(P_i, P_j)$' as given below (4) is taken into consideration if 'P_j' is non-defective.

$$CNDP_i = \sum_{j=1}^{N} SM(P_i, P_j) \quad (4)$$

On the other hand, we also take into consideration how faultiness is simulated or affected when the class data samples is s considered together with the correlation. That is why two more metrics are introduced (i.e., correlation to defective packages with class data samples and correlation to non-defective packages with class data samples). Assuming the size of a class data samples in terms of instances, we calculate '$CDP[C_i]$' (i.e., correlation to defective packages with class data samples) and '$CNDP[C_i]$' (i.e., correlation to non-defective packages with class data samples) metrics for each package as given below.

$$CDP[C_i] = \sum_{j=1}^{N} SM(P_i, P_j) * C_j \quad (5)$$

$$CNDP[C_i] = \sum_{j=1}^{N} SM(P_i, P_j) * C_j \quad (6)$$

From the above equation (5), correlation to defective packages with class data samples is formulated if 'P_j' is defective whereas from equation (6), correlation to defective packages with class data samples is formulated if 'P_j' is non-defective. Finally, the overlap between 'A' and 'B' is outlined in a contingency table '$[k_{ij}]$' where each entry 'k_{ij}' represents the number of software metrics in repeated between 'A_i' and 'B_j'. This is mathematically represented as given below.

$$k_{ij} = [A_i \cap B_j]) \quad (7)$$

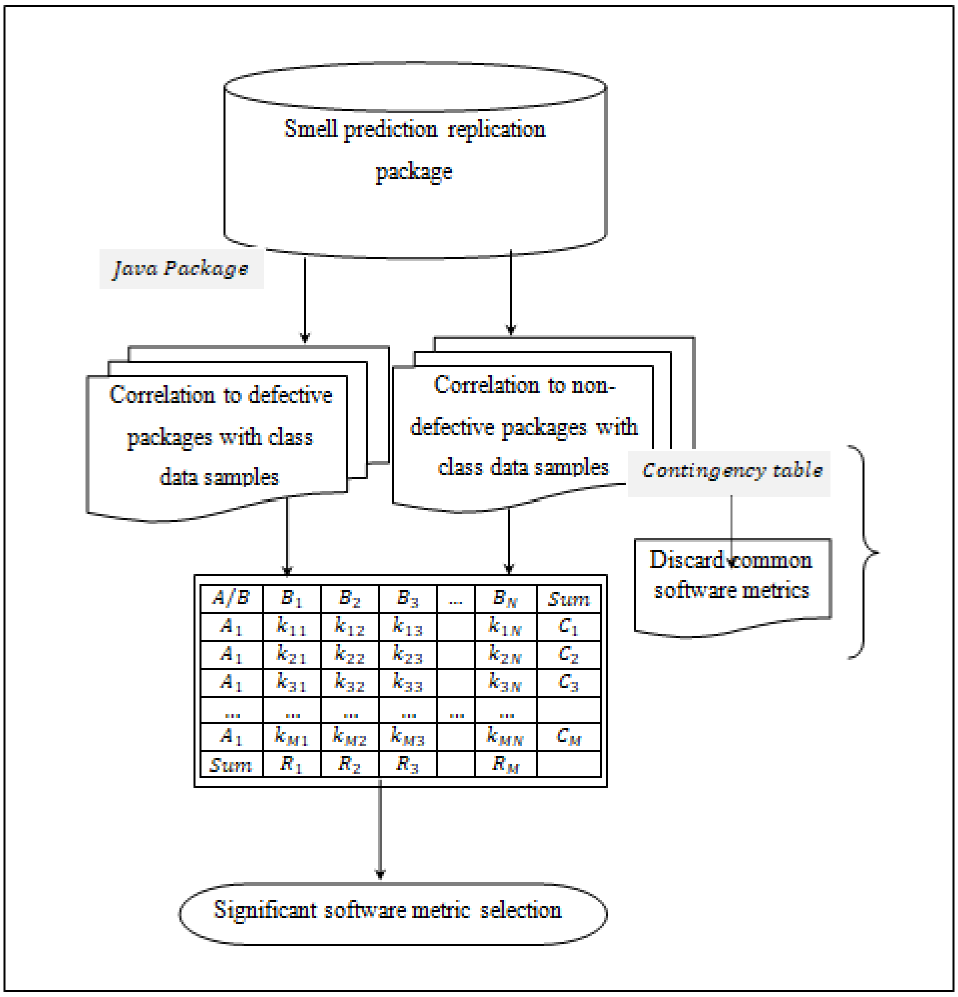
Smell prediction replication package
Java Package
Correlation to defective packages with class data samples
Correlation to non-defective packages with class data samples
Contingency table
Discard common software metrics
A/B, B1, B2, B3, ..., BN, Sum
A1, k11, k12, k13, k1N, C1
Sum, R1, R2, R3, RM
Significant software metric selection

Figure 2 contingency table construction using target projection

From the above equation (7), finally, the most significant software metrics selected are obtained according to common software metrics inclass data samples. Figure 2 shows the contingency table construction using target projection (i.e., correlation to both defective/non-defective packages with class samples).

Finally, with the contingency table construction using target projection the software metrics in common between two packages are eliminated from further processing. In this manner, more significant software metrics are obtained with minimum complexity. The pseudo code representation of Rand similarity indexive target projection-based feature selection is given below.

```
Input: Dataset '$DS$', Java Package '$P = \{P_1, P_2, \dots, P_n\}$'
Output: Computationally-efficient and significant software metrics '$k_{ij}$'
1: Initialize'$n$' independent packages
2: Begin
3: For each Dataset '$DS$' with Java Package '$P$'
4: Obtain correlation matrix as given in (1)
5: For each hypothesis
6: Evaluate Random Similarity Index as given in (2)
7: Measure correlation to defective package and non-defective package as given in (3) and (4)
8: For each class data samples
9: Evaluate correlation to defective and non-defective packages with class data samples as given
in (5) and (6)
10: End for
```

```
11: End for
12: Return significant software metrics ‘$k_{ij}$’
13: End for
14: End
```

Algorithm 1 Rand similarity indexive target projection-based feature selection

As given in the above algorithm, for each dataset and java packages acquired as input, correlation matrix is initially generated. Following which four distinct hypotheses with same and different subsets are evolved so that the number of agreements can be increased and also the number of disagreements can be decreased. By achieving this objective using rand similarity index function, the accuracy involved in software fault prediction can be improved significantly. Following which, with the aid of target projection evolved using contingency table via defective and non-defective packages with class data samples significant software metrics are obtained in a timely manner, therefore improving the software fault detection time.

C. Tversky Gradient Deep Belief Neural Network-based classifier

Over the recent few years, the software industry has contributed sizeable amount of endeavor to enhance software quality in organizations. Registering dynamic software defect prediction will assists both the developers and testers to identify the defects at an early stage, therefore minimizing both the time and endeavor. Conventional software defect prediction methods focus on models concentrate on standard source code characteristics like, complexity involved in code, lines of code and so on. However, these characteristics go wrong in predicting software defect with minimum complexity. In this work, with the selected metrics, classification is performed using Tversky Gradient Deep Belief Neural Network-based classifier. The proposed classifier comprises numerous layers such as one input layer, two hidden layers, and one output layer. Figure 3 shows the structure of Tversky Gradient Deep Belief Neural Network-based classifier model.

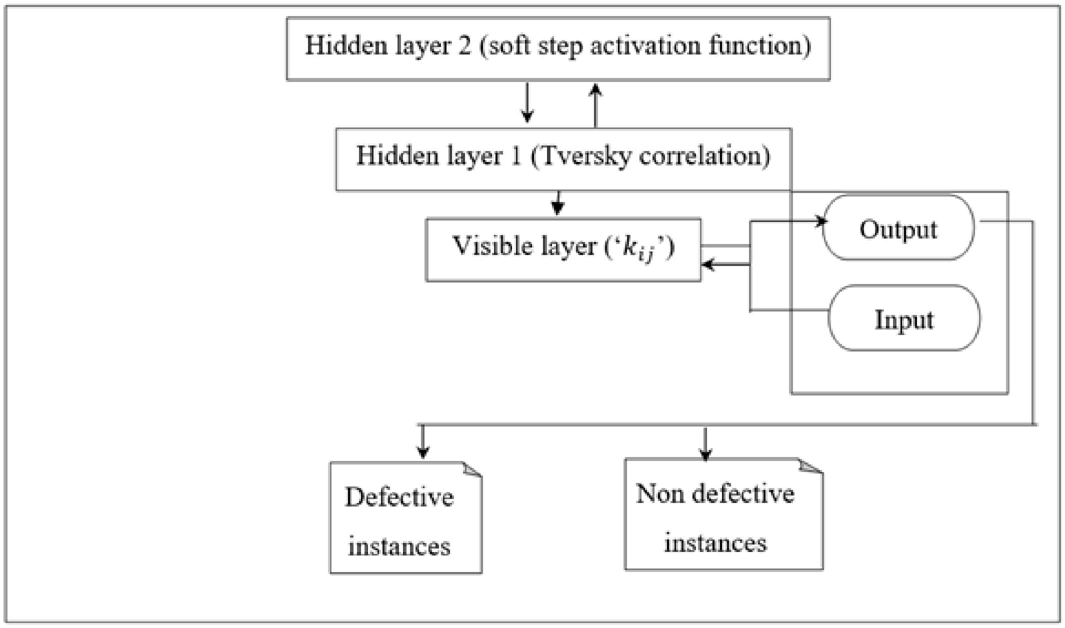
Hidden layer 2 (soft step activation function)
Hidden layer 1 (Tversky correlation)
Visible layer (‘k_{ij}’)
Output
Input
Defective instances
Non defective instances

Figure 3 Structure of Tversky Gradient Deep Belief Neural Network-based classifier model

As illustrated in the above figure, distinct numbers of selected software metrics are given as input in visible

input layer. Next, the visible input layer value is transferred into the hidden layer where Tversky correlation is applied with the purpose of identifying the software faults in Java package classes. Following which the correlation values are given to the next hidden layer where soft step activation function is applied for analyzing the given input in visible input layer and provides the final classified results as defect or non-defects in the output visible layer. Also, gradient descent function is applied to minimize classification error, therefore ensuring accurate software fault classification results obtained at the output layer with minimal complexity.

Let us consider the significant software metrics 'k_{ij}' to be initialized in the training vector.

$$k_{ij} = \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1N} \\ k_{21} & k_{22} & \dots & k_{2N} \\ \dots & \dots & \dots & \dots \\ k_{M1} & k_{M2} & \dots & k_{MN} \end{bmatrix} \quad (8)$$

From the above equation (8), '$M * N$' significant software metrics are initialized in the visible units. Following which, Tversky Gradient function is applied that tunes the parameters of the measure so as the optimally adapt it to identifying the software faults. To this end, we assume suitable training data to be given, that model about the similarity or dissimilarity between class data samples in the first hidden layer. To be more specific, let us assume the training data as given below:

$$P = \{f(A_n), f(B_n), Res_n\}_{n=1}^{N} \in P[k_{ij}] \quad (9)$$

From the above formulate (9), each training data or training example is a triplet, '$(A_n), f(B_n), Res_n$', where '$Res_n \in \{0,1\}$' infers whether the two class data samples 'A_n' and 'B_n' are considered similar or not. Next, in the second hidden layer,soft step activation function is applied to the first hidden layer results as given below.

$$\varphi(Res) = \frac{1}{1+e^{-Res}} \quad (10)$$

Next, the hidden units are updated in parallel given the visible unit as given below.

$$Prob(h_j = 1|V) = \sigma\left(X_j + \sum_i v_i w_{ij}\right) \quad (11)$$

From the above equation (11), 'σ' represent the soft step activation function with 'X_j' denoting the bias of 'h_j'. In a similar manner, the visible units are updated in parallel given in the hidden layer as given below.

$$Prob(v_i = 1|H) = \sigma\left(Y_i + \sum_j h_j w_{ij}\right) \quad (12)$$

From the above equation (12), 'σ' represent the soft step activation function with 'Y_i' representing the bias of 'v_i'. The pseudo code representation of Tversky Gradient Deep Belief Neural Network-based classifier is given below.

```
Input: Dataset '$DS$', Java Package '$P = \{P_1, P_2, \dots, P_n\}$'
Output: Precise and complexity-minimized software fault prediction
1: Initialize'$n$' independent packages, significant software metrics '$k_{ij}$'
2: Initialize visible units (i.e., significant software metrics '$k_{ij}$') to training vector
3: Begin
4: For each Dataset '$DS$' with Java Package '$P$'
5: Initialize significant software metrics '$k_{ij}$' in the training vector as given in (8)
6: Formulate Tversky Gradient function as given in (9)
7: If '$Res_n = 0$'
8: Both class data samples in training and test set are similar
9: Go to step 14
10: End if
11: If '$Res_n = 1$'
12: Both class data samples in training and test set are dissimilar
13: Formulate soft step activation function as given in (10)
14: If '$\varphi(Res) > 0$ and $\varphi(Res) \leq 0.5$'
15: Then packages identified with non defective instances
16: Go to step 23
17: Else if '$\varphi(Res) \geq 0.5$ and $\varphi(Res) \leq 1$'
18: Then packages identified with defective instances
```

```
19: Update hidden and visible units as given in (11) and (12)
20: **End if**
21: **End if**
22: **End for**
23: **End**
```

Algorithm 2 Tversky Gradient Deep Belief Neural Network-based classifier

As given in the above algorithm, the overall software fault prediction process is split into two sections, i.e., visible layer and hidden layer. Here visible layer serves as both the input and output layer. In the hidden layer the actual intermediate process is performed where the actual classification process is done to provide the final results (i.e., defective or non defective instances). First, the significant software metrics are obtained as input in the visible layer with which the actual process of software fault prediction has to be made. Second, Tversky Gradient function is applied in the first hidden layer, where with the aid of different weights reduces the false positive and false negative rate, therefore improving the overall precision and recall rate significantly. Following which soft step activation function is applied in to the second hidden layer, where the function being differentiable, i.e., the slope of the sigmoid curve can be identified at any two points with either defective or non defective instances. In this manner, the complexity involved in software fault prediction is improved significantly.

IV. Experimental setup

In the previous section, the entire process of modeling and implementation of the Rand-Index Target Projective Gradient Deep Belief Network (RTPGDBN) method is introduced. The method is also validated to be hypothetically practicable. In this section, the theoretical model is applied to real data for verification and validation. To obtain more accurate and complete data and make the experiment credible, we use the dataset collected from https://drive.google.com/drive/folders/101QbQ-TtQpyZa-APCFo4hCGAc_c-g6-Y and are implemented in JAVA language. This work conducts experiments using the prediction method and basic data proposed above and analyzes the results and validates accordingly. Considering that the entire software defect detection process includes several specific processing procedures, it is important to ensure that we use appropriate and effective methods at each step.Performance analysis is carried out with different quantitative metrics such as software fault prediction time, software fault prediction accuracy, precision recall and space complexity.

V. Evaluation measures

To measure and validate the performance of defect prediction, the following performance metrics are used, precision, recall, software fault prediction accuracy, software fault prediction time and space complexity. All the five metrics are introduced below.

A. *Case scenario 1: Software fault prediction time*

This section represents the software fault prediction time complexity of the methods based on the testing. Furthermore, the testing technique is viewed as a critical statistic as it reveals the efficiency and general performance and significant performance indicators. This is mathematically stated as given below.

$$SFPT = \sum_{i=1}^{n} C_i * Time\ [SFP] \qquad (13)$$

From the above equation (13), the software fault prediction time '$SFPT$' is measured by taking into consideration the class data samples 'C_i' for simulation and the time involved in the software fault prediction process '$Time\ [SFP]$'. It is measured in terms of milliseconds (ms). Table 1 compares the RTPGDBN method with other state-of-the-art software fault prediction methods with the same settings (as in table 1). The RTPGDBN method achieved the best results with minimum amount of time taken or consumed for predicting the fault prone classes.

Table 1 Software fault prediction time using RTPGDBN, HA-LCBS [1] and BUGPRE [2]

Number of class data	Software fault prediction time (ms)		
	RTPGDBN	HA-LCBS	BUGPRE
1000	11	14	17
2000	11.55	15.35	19.25
3000	13.15	20.15	25.35
4000	18	25.35	30.15
5000	21.35	30	38.35
6000	25.85	33.45	45.25
7000	28.25	35.35	50.35
8000	30	38.45	55.35
9000	31.45	41.35	60.25
10000	35.85	48.15	70.15

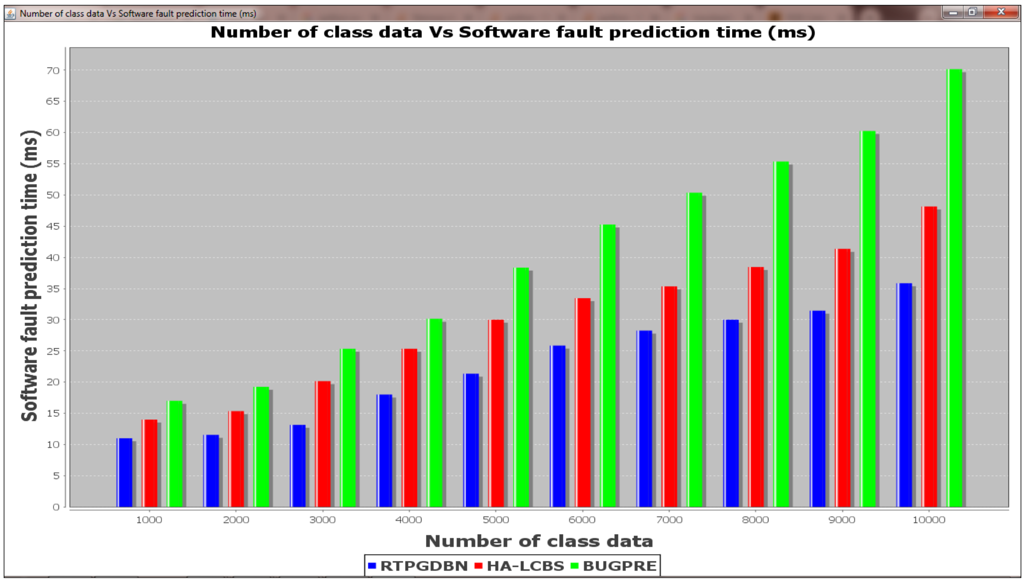

Figure 4 Comparison of software fault prediction time using RTPGDBN, HA-LCBS [1] and BUGPRE [2]

Firstly, we compare our proposed RTPGDBN with HA-LCBS [1] and BUGPRE [2] with regard to the software fault prediction time efficiency. As shown in figure 4, the software fault prediction time efficiency of our method is the best in these three methods, followed by HA-LCBS [1] and the worst by BUGPRE [2]. In the HA-LCBS [1] method GA was employed for automating module detection composition including cohesion and coupling metric involvinglarge class bad smell. The time consumed in the overall testing process was not focused, so the software fault prediction time of HA-LCBS [1] is less. On the other hand, the BUGPRE [2] does not consider in obtaining changed modules in the current version. Soit views defective and non defective instances as equal, and performs prediction directly on whole sample package. So the software fault prediction time of BUGPRE [2] isslightly higher than that of the proposed RTPGDBN method. In our method, contingency table was used in distinguishing between defective and non defective cases, with which basic probability calculations were performed easily via multivariate frequency distribution. As a result, the software fault prediction time using RTPGDBN method was found to be comparatively better by 25% compared to [1] and 44% compared to [2] respectively.

B. *Case scenario 2: Software fault prediction accuracy*

The software fault prediction accuracy is an experimental measure to express the software fault prediction diagnostic tests' evaluation performance. The software fault prediction accuracy is mathematically stated as given below.

$$\mathrm{SFPA} = \sum_{i=1}^{n} \frac{C_{PC}}{C_i} * 100 \qquad (14)$$

From the above equation (14), the software fault prediction accuracy 'SFPA' is measured based on the class data samples 'C_i' and the class data samples predicted correctly 'C_{PC}'. It is measured in terms of percentage (%).The comparison results between the proposed RTPGDBN method and existing methods, HA-LCBS [1] and BUGPRE [2] are listed in table 2, showing that the RTPGDBN method improve the prediction of software defect significantly. We noted that the RTPGDBN method had good prediction performance in distinguishing defective and non defective. In addition, we compared the RTPGDBN method with other state-of-the-art software defect prediction methods in identifying defect/non defect with the same settings (as in table 2). The RTPGDBN achieved the best results, suggesting that the presented method had prospective to be utilized in computer-aided diagnostic system for locating code in software areas where the occurrence of fault is high.

Table 2 Software fault prediction accuracy using different software fault prediction methods

Number of class data	**Software fault prediction accuracy (%)**		
	RTPGDBN	HA-LCBS	BUGPRE
1000	99.5	90.5	85.5
2000	96.35	86.55	80
3000	95	85.35	79.15
4000	94.15	85	76
5000	93	84.85	74.15
6000	92.85	83	71
7000	91.35	82.35	69.25
8000	91	82	66
9000	90	81.15	64.15
10000	88.15	80	61

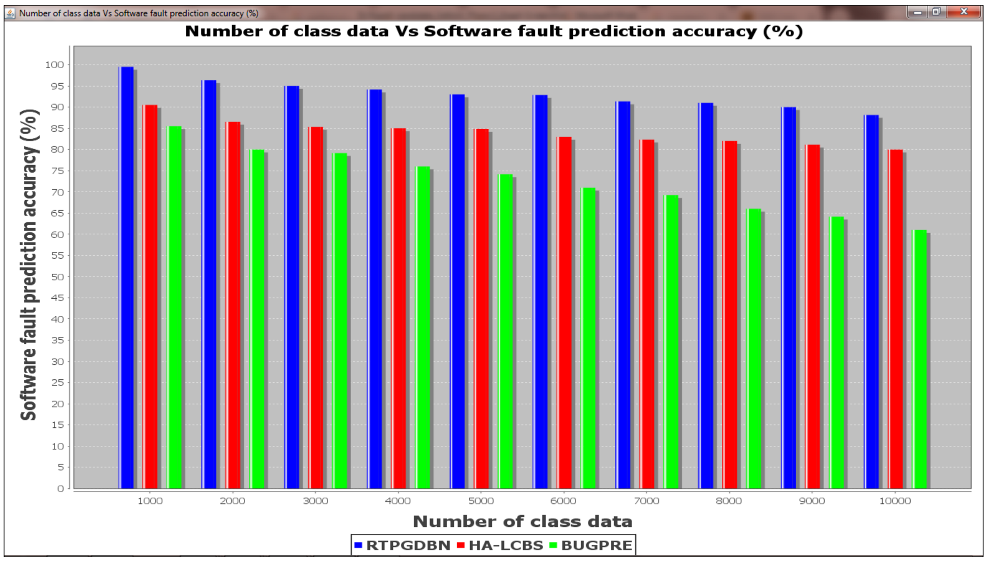

Figure 5 Comparison of software fault prediction accuracy using RTPGDBN, HA-LCBS [1] and BUGPRE [2]

Figure 5shows the comparison results of software fault prediction accuracy efficiency. The more class data the software quality manager stores for simulation purpose, the lower the software fault prediction accuracy efficiency is. It can be seen clearly from figure 5 that the software fault prediction accuracy efficiency is influenced by number of class data samples involved in simulation. As shown in figure 5, the software fault prediction accuracy efficiency of our method is the best in these three schemes. In addition, as the number of class data samples increases, the prediction efficiency gap among these three methods becomes larger and larger. In these three methods, the software fault prediction accuracy efficiency of our method is the highest. The state-of-the-art methods, HA-LCBS [1] and BUGPRE [2] and the proposed RTPGDBN method automated large class bad smell to ensure accurate prediction. This bad smell detection method though can enhance the accuracy, but it also needs additional storage space to store the bad smell. Nevertheless, this additional storage space is unavoidable. On the one hand, the automation of large class bad is derived from the consistent code. In other words, the accuracy is said to be compromised. On the other hand, by applying rand similarity indexive target projection-based feature selection algorithm, our distinct hypotheses with same and different subsets were evolved with the objective of increasing the agreements and decreasing the disagreements. This in turn improved the software fault prediction accuracy using RTPGDBN method by 11% compared to [1] and 29% compared to [2].

C. Case scenario 3: Precision and Recall

Precision is one of the performance metrics used to represent the correctness classification. The precision performance metric is calculated by dividing the number of class data samples that correctly predicted the software defect divided by the total number of correctly predicted class data samples. This is mathematically stated as given below.

$$Precision = \frac{TP}{(TP+FP)} \tag{15}$$

From the above equation (15), precision '$Precision$' is measured based on the true positive 'TP' that represent the true prediction of positive values and the false positive 'FP' that represent the false prediction of the positive values respectively. Recall on the other hand denotes the rate of defecting models. The recall rate is calculated by dividing the number of class data samples that correctly predicted defect divided by the total number of modules or class data samples that are actually defective. This is mathematically represented as given below.

$$Recall = \frac{TP}{(TP+FN)} \tag{16}$$

From the above equation (16), recall rate '$Recall$' is measured by taking into consideration the true positive 'TP' that represent the true prediction of positive values and the false negative 'FN' that denotes the false prediction of the negative values respectively. Table 3 given below lists the precision and recall rate arrived at using 1000 distinct numbers of class data samples obtained at different time instances.

Table 3 Precision and Recall rate using different software fault prediction methods

Number of class data	**Precision**			**Recall**		
	RTPGDBN	HA-LCBS	BUGPRE	RTPGDBN	HA-LCBS	BUGPRE
1000	99.79	98.41	97.43	99.48	97.92	96.33
2000	98.35	95.25	92.35	98.35	91.35	84.35
3000	98	95	94.15	98	90.15	84
4000	97.55	94.35	91.85	97.16	89.35	83.15
5000	97.35	94	90.35	97	89	82
6000	97	93.55	89.15	96.35	89.25	80.15
7000	96.55	93.35	89	96	89	80
8000	96.15	92.35	87.15	95.55	87.15	79.15
9000	95.35	92	86	95	86	78
10000	95	91.15	85.25	94.15	89.15	77.35

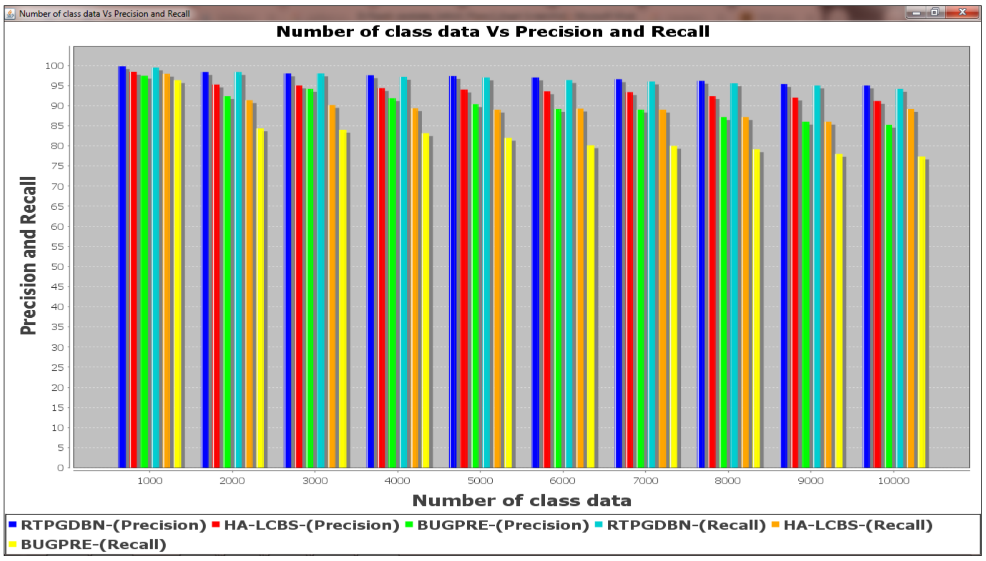

Figure 6 Comparison of precision and recall using RTPGDBN, HA-LCBS [1] and BUGPRE [2]

In the figure 6 we observe that the precision and recall rate of our proposed RTPGDBN method is moderately higher than that of HA-LCBS [1] and BUGPRE [2]. For the sake of inspecting the governing aspects, we compare the true positive, false positive and false negative rate for class data samples to detect defective or non defective cases in these three methods in figure 6. Note that we define the true positive for class data samples to generate the probability values as the precision and recall. As shown in the figure, the precision and recall of RTPGDBN method is the highest, followed by HA-LCBS [1] and BUGPRE [2] respectively. Just as we analyzed in section 3.3, measure of quality and quantity for refactoring process for both defective and non defective instances were made significantly. So the false positive rate and false negative rate of HA-LCBS [1] and BUGPRE [2] is much higher than that of RTPGDBN method. Also only two layers were employed where the visible layer holds the significant software metrics and the hidden layer performed the actual process of predicting software faults. Following which by applying the Tversky Gradient function via distinct weights reduced the false positive and false negative rate, therefore improving the overall precision and recall rate using RTPGDBN method by 3% 8% (i.e., in terms of precision) and 8% 18% (i.e., in terms of recall) respectively.

D. Space complexity

Finally, in this section, space complexity involved in software fault prediction is analyzed to validate the efficiency of the method. The space complexity is mathematically formulated as given below.

$$SC = \sum_{i=1}^{n} C_i * Mem\ [SFP] \quad (17)$$

From the above equation (17), the space complexity 'SC' is measured by taking into consideration the class data samples 'C_i' involved in the simulation process and the actual memory consumed '$Mem\ [SFP]$' is software fault prediction. It is measured in terms of kilobytes (KB). Finally, table 4 given below lists the space complexity results using the three methods, RTPGDBN, HA-LCBS [1] and BUGPRE [2].

Table 4 Space complexity using different software fault prediction methods

Number of class data	**Space complexity (KB)**		
	RTPGDBN	HA-LCBS	BUGPRE
1000	35	50	75
2000	38	55	80
3000	45	70	85
4000	50	73	93
5000	55	78	100
6000	58	85	110

7000	70	89	118
8000	75	93	125
9000	83	95	133
10000	90	105	140

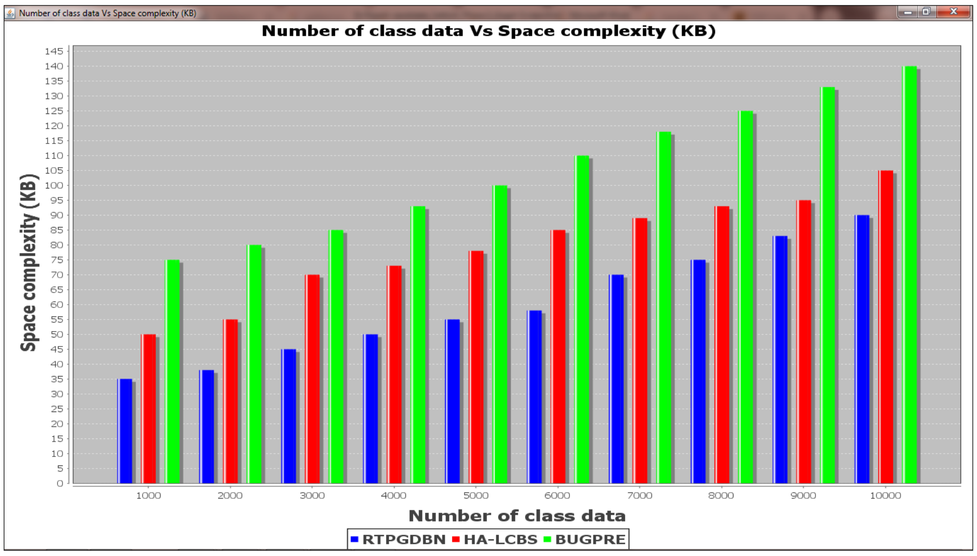

Figure 7 Comparison of space complexity using RTPGDBN, HA-LCBS [1] and BUGPRE [2]

We provide a comparison of space complexity in figure 7. We arbitrarily sorted out 1000, 2000,…, 10000 class data from the datasetto do experiments. When the total number of class data samples is proportionately negligible, the space complexity of the three methods, RTPGDBN, HA-LCBS [1] and BUGPRE [2] are less. When the number of class data is 1000, the space complexity of RTPGDBN method is the shortest, followed by HA-LCBS [1] and finally BUGPRE [2]. The main reason is that HA-LCBS [1] and the proposed RTPGDBN method both cohesion and coupling metric type passes paired value results (i.e., significant software metrics) to a deep learning model for automating the detection of large class bad smell. In contrast BUGPRE [2] employs propagation tree-based associated analysis that though improved accuracy and F1-score, however consumes more stack for storing intermediate results. Also as shown in the above figure, when the total number of class data increases, the space complexity of BUGPRE [2] is the highest,followed by HA-LCBS [1] and our method. This is due to the application of Deep Belief Neural Network where with the aid of visible and hidden layer for processing minimizes the layer involved in the defect prediction. This in turn reduces the space complexity involved in the overall process using RTPGDBN by 26% compared to [1] and 45% compared to [2].In conclusion, compared with methods [1], [2], our proposed RTPGDBN method has shown good efficiency in terms of software fault prediction accuracy, software fault prediction time, precision, recall and space complexity.

VI. Conclusion

Coming up with a high-quality software product is a prerequisite task in the course of two distinct phases, namely, software testing and maintenance. One of the significant considerations as far as software product quality is concerned is the density involved in defect. In this work we proposed an enhanced deep belief neural network called Rand-Index Target Projective Gradient Deep Belief Network (RTPGDBN) to predict software fault. The constructed RTPGDBN method has been estimated against other popular deep learning methods using smell prediction replication package dataset. First, Rand similarity indexive target projection-based feature selection algorithm was employed in selecting computationally efficient significant software metric via Random Similarity Index. Second Tversky Gradient Deep Belief Neural Network-based classifier model was applied to selected software metric for making significant classification between defective and non defective class instances. The obtained results demonstrate that the RTPGDBN significantly outshines the other software fault prediction methods with very accuracy levels. Furthermore, our method is competitive to other deep learning methods such as HA-LCBS and BUGPRE in terms of precision and recall with minimum space complexity.

References

[1] Ayad Tareq Imam, Basma R. Al-Srour, AyshAlhroob, “The automation of the detection of large class bad smell by using genetic algorithm and deep learning”, Journal of King Saud University –Computer and Information Sciences, Elsevier, Volume 34, Issue 6, 2022, Pages 2621-2636. https://doi.org/10.1016/j.jksuci.2022.03.028[Hybrid Approach to detect Large Class Bad Smell (HA-LCBS)]

[2] Zixu Wang, Weiyuan Tong, Peng Li, Guixin Ye, Hao Chen, Xiaoqing Gong &Zhanyong Tang, “BUGPRE: an intelligent software version-to-version bug prediction system using graph convolutional neural networks”, Complex & Intelligent Systems, Springer, 2022, Pages 1-21. https://doi.org/10.1007/s40747-022-00848-w[Bug Prediction (BUGPRE)]

[3] GörkemGiray, Kwabena EboBennin, ÖmerKöksal, Önder Babur, BedirTekinerdogan, “On the use of deep learning in software defect prediction”, The Journal of Systems & Software, Elsevier, Feb 2023

[4] JalajPachouly, Swati Ahirrao, Ketan Kotecha, “SDPTool : A tool for creating datasets and software defect predictions”, Software, Elsevier, Feb 2022

[5] Pradeep Singh and Shrish Verma, “Multi-Classifier Model for Software Fault Prediction”, The International Arab Journal of Information Technology, Vol. 15, No. 5, September 2018

[6] Fengyu Yang, Yaxuan Huang, Haoming Xu, Peng Xiao, and Wei Zheng, “Fine-Grained Software Defect Prediction Based on the Method-Call Sequence”, Computational Intelligence and Neuroscience, Hindawi, Aug 2022

[7] Raymon van Dinter, CagatayCatal, GörkemGiray, BedirTekinerdogan, “Just-in-time defect prediction for mobile applications: using shallow or deep learning?”, Software Quality Journal, Springer, Apr 2023

[8] Tanujit Chakraborty and Ashis Kumar Chakraborty, “Hellinger Net: A Hybrid Imbalance Learning Model to Improve Software Defect Prediction”, IEEE Transactions on Reliability, Mar 2020

[9] Jiaxi Xu, Fei Wang, and Jun Ai, “Defect Prediction with Semantics and Context Features of Codes Based on Graph Representation Learning”, IEEE Transactions on Reliability, Vol. 70, No. 2, June 2021

[10] ChakkritTantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto, “The Impact of Automated Parameter Optimization on Defect Prediction Models”, IEEE Transactions on Software Engineering, Dec 2017

[11] Xue Han, Gongjun Yan, “Fault Prediction with Static Software Metrics in Evolving Software: A Case Study in Apache Ant”, Journal of Computer and Communications, Oct 2022

[12] PetarAfric, Davor Vukadin, Marin Silic, Goran Delac, “Empirical Study: How Issue Classification Influences Software Defect Prediction”, IEEE Access, Feb 2023

[13] EminBorandag, “Software Fault Prediction Using an RNN-Based Deep Learning Approach and Ensemble Machine Learning Techniques”, Applied Sciences, Apr 2023

[14] Kun Song, ShengKaiLv, Die Hu, and Peng He, “Software Defect Prediction Based on Elman Neural Network and Cuckoo Search Algorithm”, Mathematical Problems in Engineering, Hindawi, Nov 2021

[15] Can Liu, SumayaSanober, Abu Sarwar Zamani, L. Rama Parvathy, Rahul Neware, and Abdul Wahab Rahmani, “Defect Prediction Technology in Software Engineering Based on Convolutional Neural Network”, Security and Communication Networks, Wiley, Apr 2022

[16] Fahad H. Alshammari, “Software Defect Prediction and Analysis Using Enhanced Random Forest (extRF) Technique: A Business Process Management and Improvement Concept in IOT-Based Application Processing Environment”, Mobile Information Systems, Hindawi, Sep 2022

[17] Firas Alghanim, Mohammad Azzeh, Ammar El-Hassan, Hazem Qattous, “Software Defect Density Prediction Using Deep Learning”, IEEE Access, Nov 2022

[18] Can Liu, SumayaSanober, Abu Sarwar Zamani, L. Rama Parvathy, Rahul Neware, and Abdul Wahab Rahmani, “Defect Prediction Technology in Software Engineering Based on Convolutional Neural Network”, Security and Communication Networks, Hindawi, Apr 2022

[19] Hafiz Shahbaz Munir, Shengbing Ren, Mubashar Mustafa, Chaudry Naeem Siddique, Shazib Qayyum, “Attention based GRU-LSTM for softwaredefect prediction”, PLOS ONE, Mar 2021

[20] Jonathan Bryan, Pablo Moriano, “Graph-based machine learning improves justin- time defect prediction”, PLOS ONE, Apr 2023pr 2023