Chapter 3

BPDET: Classification Based Software Defect Prediction Model

3.1 Introduction

The main objective of the software bug prediction technique is to classify faulty and fault-free modules, and then the developer will assign reasonable testing sources, allocate testing preferences of various software modules that boost the quality of software systems. This chapter presents a novel SDP approach called Bug Prediction using Deep representation and Ensemble learning Technique (BPDET). According to the latest research and literature survey over SBP, we have framed three research queries (RQ).
RQ-1: Effectiveness of BPDET in terms of performance metrics compared with other fault prediction model.

RQ-2: How useful BPDET model compared with traditional methods regarding the class imbalance and over-fitting problem?

RQ-3: How much training time taken by the BPDET model?

Software practitioners revealed that deep representation is better than traditional feature extraction techniques [92, 128]. DBN and SDA have successfully implemented for SBP, whereas SDA outperforms for noisy datasets [254]. Few other

researchers [80, 194] claims over the quality of datasets and their noisy instances, which should be considered during the SBP model. The researcher revealed that ensemble learning could lead to avoid the risk of over-fitting problem [253] and the biased result (class imbalance issue) [240] but surpass the performance over weak classifiers. It is necessary to extract more relevant features which cause a noisy removal extraction process for better results in an SBP model.

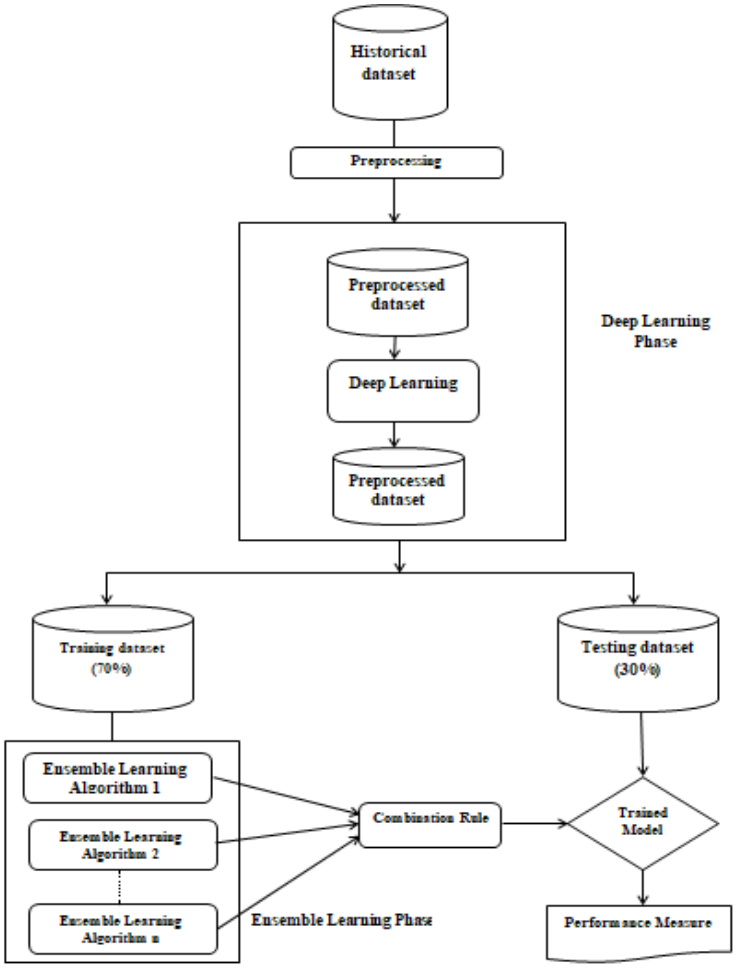


FIGURE 3.1: Working architecture of BPDET.

3.2 Architectural design

BPDET is a classification-based SBP model; it mainly classifies the module into faulty and fault-free classes. The complete architectural design of BPDET is shown

in Fig. 3.1. Its design contains two different phases, the deep learning phase, and the ensemble learning phase. Both Deep learning and Ensemble learning phases will be able to deal with both class imbalance and over-fitting problems.

The software defect dataset is divided into two parts; training data and testing

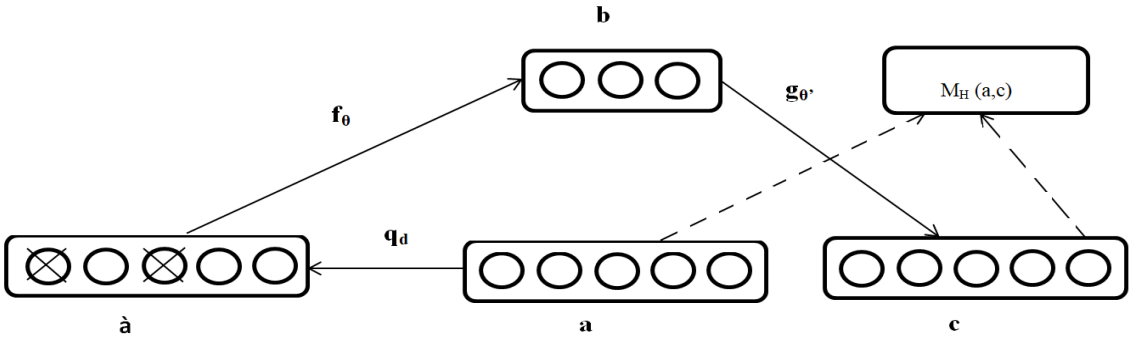


FIGURE 3.2: Design of Denoising auto encoder, the initial input is a and it is corrupted to à, à is mapped with b, then finally c tries to regenerate to a. The regeneration error is represented by M_H(a,c). Taken from [254].

data. The objective of the training dataset is to train the model, and the testing dataset is then applied to measure the performance of the trained model. But before performing training and testing, the dataset will be preprocessed, which includes duplicate instance deletion, missing values replacement, and normalization. Afterward, to extract deep representation from stacked denoising autoencoders, both training and testing data will be input to it. Then after training data is identified by deep representations, it is used to train the EL phase. Then testing data is determined by the deep representation phase to analyze the performance of the trained EL phase. Every detail of the proposed work is presented in the next section.

3.2.1 Proposed model

In this section, we fully explain the Bug Prediction using the Deep representation and Ensemble learning Technique (BPDET) model. Firstly we will present implications for combining SDA and EL, then preprocessing of datasets, then will explain different phases of BPDET and its algorithm. The deep learning phase and ensemble learning phase are two different phases of BPDET.

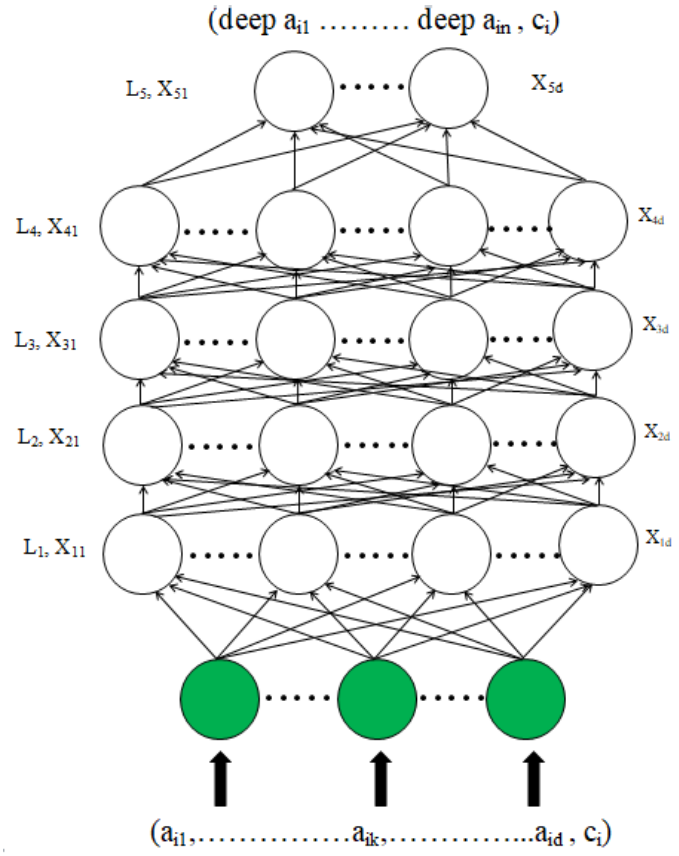


FIGURE 3.3: Staked denoising auto-encoder (SDA), the input layer nodes represented by the solid circle(green) at the bottom, the middle circles are hidden layer nodes. The class label c_i of deep representation is the same as the input.

3.2.1.1 Implication of combining SDA and EL techniques

The SDA has a powerful feature extraction mechanism; it was applied in many applications such as electric load forecasting [248], an intelligent diagnosis system [269], big data analysis [173] etc. SDA is also useful in classification problems like image classification such as [143, 102].

Ensemble learning is a powerful technique for classification/regression, as it has a property to perform as a hybrid classifier and combine the results of each base learner, which made it more efficient compared with other methods. Combining both SDA and EL can lead to more efficient EL methods.

3.2.1.2 Data preprocessing

The most common dataset which is used for an SBP domain is the NASA repository dataset. Duplicate instances need to be deleted [81]. Missing data points handled by mean corresponding values [257] as there will be no big mean difference after applying it. Then we applied min-max normalization technique. We have applied SMOTE [33] sampling technique to avoid the class imbalance problem. A detailed discussion about data preprocessing is presented in an earlier chapter 2.3.1.

3.2.1.3 Deep learning phase

The deep learning phase consists of three sub-phases: stacked denoising auto-encoder (SDA), parameter selection for SDA, and generation of deep learning representation; how these phases are working are explained below.

(a) **Staked denoising auto-encoders (SDA)**
After combining various denoising autoencoders (DAE) staked [254] on the top of each other a powerful model build called a stacked denoising autoencoder (SDA) [255]. DAE is a reshape variety of fundamental auto-encoder by appending the corruption process, and auto-encoder is the feed-forward neural network, with input layer, hidden layer, and an output layer. The objective of DAE is to establishing new input from the corrupted input. The basic framework of SDA is shown in Fig. 3.2.
Input vector d-D where a $\in R^d$, the corrupted version of a is à which also belongs to R^d. It can be acquired by using first arbitrary choosing $v*d$ elements from a, then altering their value to zero and without changing reminder, where v is the corrupted rate. In the first stage of the basic auto-encoder encoded the à to hidden layer c, then in the second stage, it decodes the output of the hidden layer, then produces a reconstruction of c. These two stages can be formalized as below.

Encoding: $\mathrm{r}(\grave{a}) = f_{\theta_1}(M_1\grave{a} + \beta_1)$

Decoding: $\mathrm{s}(\grave{a}) = f_{\theta_2}(M_2\grave{a} + \beta_2)$

In the above formula, $M_1 \in R^{p_1 \times d}$ and $M_2 \in R^{d \times p_1}$ are weight metrics and $\beta_1 \in R^{p_1}$ and $\beta_2 \in R^d$ are different bias term correspondingly. f_{θ_1} and f_{θ_2} are two non-linear activation functions, p_1 is the number of nodes in hidden layer. In this work Sigmoid function is used as a default activation function.
To train the model, the given input series is $[a_i]_{i=1}^n$, the objective is to minimize the reconstruction error, this error is mainly generated in between, input a_i, and it's correspondingly its reconstruction s(à). The framework of the model is represented as ϕ.

$$\phi = \arg\min \frac{1}{n} \sum_{i=1}^{n} M_H(a_i, s(à))$$

Reconstruction error measure by M_H; its values can be squared for more simplicity, $M_H(a,c) = ||\ a - c||^2$. In the end, the back-propagation algorithm and stochastic gradient descent rule for updating the framework until stopping condition reached, such as reconstruction error less than its minimum limit.
After training the first DAE, the training data is then feed-forwarded from the first DAE, and the output of the hidden layer of first DAE is used as training data for the second DAE. Similarly, third DAE and so on. The DAE from a staked and lie on top of each other and trained using unsupervised greedy layer techniques [17].

(b) **SDA parameters selection**
As from the previous discussion, three structural parameters that need to be set are given below:

(i) Number of hidden layer

(ii) Number of nodes in a hidden layer

(iii) Corruption rate value

(iv) Regularization methods

We have used three hidden layers for the simplicity of the model in SDA, the basic configuration of SDA was given by [255]. In the first two hidden layers, we set several nodes by five different discrete values; we tested by 100, 200, 500, and 1000 nodes in the first two hidden layers during experiments, whereas for the last hidden layer, we tested with six discrete values, i.e., 5, 10, 20, 40, 100 and 200. The corruption rate is tested at seven different values i.e., 0.1, 0.2,

0.3, 0.6, 0.7, 0.8 and 0.9. First, we fix the number of node in the first hidden layer and change the nodes in the other two layers and performed experiments, then we fix the nodes of the last hidden layer and vary the nodes of the other two layers. We have also tried four hidden layers of SDA and different nodes in each hidden layer, but it was not effective as compared to three hidden layer architecture. The cross-validation method was used again to achieve the most favorable corruption rate for every dataset. Similar SDA parameters are also taken by [249]. We have applied dropout and early stopping as regularization techniques. Dropout [236] with "keepProbability" equal to 0.8 at each layer in the training set. Early stop regularization not only reduces cost but also avoids over-fitting chances, whereas early-stopping [271] also avoids chances of over-fitting.

(c) **Deep representation generation**
We feed the preprocessed training and testing data to acquire corresponding deep representation when the SDA model is trained. Let us take an example to understand, as shown in Fig. 3.3, that a_{ik} represent the k^{th} metrics of i^{th} instance, whereas c_i represents the class label of i^{th} instance. The class label c_i is the same as the output of deep representation. The deep a_{i1},.., deep a_{in} is the output deep representation of input a_1,.. a_{ik},.. a_{id} which is coming from L_5 (last hidden layer). $L_1, .., L_5$, are layers of SDA whereas $X_{1d}, X_{2d}, .., X_{5d}$ are the end nodes of SDA layers respectively.

3.2.1.4 Ensemble learning phase

EL phase utilized the deep representation-based training data and gave input to ten different classifiers. This phase extracts the deep representation features from training data. EL phase is the collection of ten different weak and robust classifiers, which have been used in experiments. Ensemble learning [52] is the hybrid technique in which numerous weak or robust classifiers combine to create a new classifier and result of different classifiers combined and form a new result.

$$Y_{final} = AverageProbability[y_1, y_2, .., y_{10}] \tag{3.1}$$

Here Y_{final} is the final output of the ensemble learning technique after applying the average probability of each base learner y_1 to y_{10}. We try to utilize the deep

representation-based training set to build a powerful classification model and then apply a deep representation-based test set to estimate the model's performance. EL has the capability to build a strong classifier using base learners and combine the results of each base learner.

The input to the algorithm 1 is $D_{tr} = [\mathrm{a}_i, c_i]_{i=1}^{n}$ and $D_{ts} = [c_j]_{j=1}^{m}$ are training and testing data respectively. σ_d is the defective rate for heterogeneous ensemble learning i.e., Voting algorithm [126] method. Where a_i, $\mathrm{a}_j \in R^p$ are two p-D vectors which are made of p software metrics. If $c_i = 1$, then i^{th} software module is non-defective (majority class), elsewhere defective (minority). In the preprocessing step, we delete every repeated instance in both D_{tr} and D_{ts} then replace the missing value with corresponding value and normalized as by min-max method as described in section 3.2.1.2.

Stacked denoining auto-encoder (SDA) gets an input a_{ik} of k^{th} metrics and i^{th} instance and give output a deep representation as $Deepa_{ik}$ with same class label (c_i) both in input and output.

The dataset is divided into training (D_{tr}) and testing data (D_{ts}). The training data is 70% of the total dataset; the rest 30% is the testing dataset. We experimented using 80%, 75%, 65%, and 60% as a training dataset and 20%, 25%, 35%, and 40% as a testing dataset but got the better result in 70% and 30% training and testing dataset ratio. The expected output is class label prediction of every instance in testing data.

There are ten base learner in an EL phase, we symbolize each trained base learner as b_i (l=1, 2,...,10). We use performance of a model as a average performance of each base learner b_i. After implementing 10-fold cross validation we calculate weight (w_i) of every base learner for both MCC and F-measure (see section 3.2.2.4). For $\mathrm{a}_j \in D_{ts}$, the probability of prediction of class label $c_j = 1$ and $c_j = 0$ by the base learner b_k.

$$MCC_i = \frac{AvG_{MCC_i}}{\sum_{i=1}^{10} AvG_{MCC_i}} \tag{3.2}$$

The D_{tr} is for ensemble learning technique phase, as in EL phase, the dataset is divided into ten different subsets with replacement, it means the data points can be common in some or all section. Calculate the various performance evaluation parameters, MCC, AUC, PRC, F-measure of the BPDET as explained in section 3.2.2.4.

$$F_i = \frac{AvG_{F_i}}{\sum_{i=1}^{10} AvG_{F_i}} \tag{3.3}$$

For the base classifier b_i, the probability of prediction for the class label $c_j = 1$, $c_j =$

Algorithm 1: BPDET algorithm

Input $\leftarrow$ D_{tr} and D_{ts} are training and testing data respectively. σ_d is defective rate. Heterogeneous ensemble learning(Voting) method call.
Output $\rightarrow$ Class label c_j prediction of every instance in testing data.
for every instance of D_{tr} and D_{ts}
Delete every duplicate instances, both in D_{tr} and D_{ts}. .
Fill the missing instance value both in D_{tr} and D_{ts} with mean of the corresponding instance value2.3.
Normalize both D_{tr} and D_{ts} using min-max method.
Input $\leftarrow$ SDA for generating deep representation of a_{ik}, where k^{th} metrics of i^{th} instance of class label c_i.
Output $\rightarrow$ $Deepa_{ik}$ is the deep representation of a_{ik} As described in section 4.1.2 (c)
end for
for every base learner b_l EL phase l=1, 2,...10.
do train D_{tr}, and hypothesis give back from different base learner b_l.
According to b_l conduct 10 fold cross fold validation in D_{tr}
Calculate average MCC(AvG_{MCC_i}) and average F-measure(AvG_{F_i}) (using equations 3.2 & 3.3)
end for
Apply b_l, l = 1, 2, 3,...,10 to get the probability of instance a_j as defective and non-defective as PD_{lj}, PND_{lj}, where $PD_{lj} + PND_{lj} = 1$.
Set combined average probability of non-defective and defective as

$$[cPND_j, cPD_j] = max \begin{cases} \sum_{i=1}^{10} MCC_i * [PND_{ij}, PD_{ij}] \\ \sum_{i=1}^{10} F_i * [PND_{ij}, PD_{ij}] \end{cases}$$

Find the maximum probability for being defective instance w.r.t l^{th} base learner as $maxPD_{lj}$.
Training dataset D_{tr}(70%) given input to Ensemble learning phase as suggested in section 3.2.1.4.
Set threshold T = $\frac{e^{\lambda*(\sigma_d-\delta)} - e^{-(\lambda*(\sigma_d-\delta))}}{e^{\lambda*(\sigma_d-\delta)} + e^{-(\lambda*(\sigma_d-\delta))}}$
    If ($\sigma_d < 0.25$)
        If (T <$maxPD_{lj}$ <0.5)
            then $[cPND_j, cPD_j] = [PD_{lj}, PND_{lj}]$
        If($maxPD_{lj} > 0.5$)
            then $[cPND_j, cPD_j] = [PND_{lj}, PD_{lj}]$
    else $[PND_{lj}, PD_{lj}] = [cPND_j, cPD_j]$
    If $cPND_j > cPD_j$
    then $c_j = 1$, testing instance a_j is non-defective.
    else $c_j = 0$, a_i is defective.
Output $\rightarrow$ Class label c_i prediction of every instance a_i is testing data.

0 for non-defective and defective software module respectively and these represents

as PND_{ij} and PD_{ij}. The complete tuple is represented as $[PND_{ij}, PD_{ij}]$.
We are taking the maximum probability among the weighted probability of MCC and F-measure (from equation 3.2 and 3.3). The computation of combined probabilities of a software module for defective and non-defective is evaluated using equation 3.4.

TABLE 3.1: Comparison of MCC over baseline and EL methods with BPDET. Note** "-" symbol implies model cannot give output.

Dataset	**BPDET(mean)**	Tong et al. [249]	**AdaBoost**	**Bagging**	**Random forest**	**Logistic Boost**
CM1	**0.42**	**0.1333**	-	0.021	0.042	0.080
JM1	**0.2319**	0.2217	0.083	0.208	**0.223**	0.179
KC1	**0.333**	0.2405	0.211	**0.330**	0.301	0.255
KC2	**0.415**	0.3587	0.387	0.381	0.394	**0.406**
KC3	0.137	**0.306**	**0.344**	0.110	0.189	0.274
MC1	0.142	**0.329**	-	-	**0.268**	-0.005
MC2	**0.332**	**0.3289**	0.102	0.198	0.317	0.205
PC1	**0.382**	0.3179	-	0.286	**0.364**	0.250
MW1	**0.331**	**0.3207**	0.243	0.136	0.217	0.193
PC2	**0.171**	**0.1242**	-	-	-	0.103
PC3	0.166	**0.3095**	-	0.088	**0.238**	0.172
PC4	**0.469**	**0.5758**	0.347	0.491	0.484	0.103

$$[cPND_j, cPD_j] = max \begin{cases} \sum_{i=1}^{10} MCC_i * [PND_{ij}, PD_{ij}] \\ \sum_{i=1}^{10} F_i * [PND_{ij}, PD_{ij}] \end{cases} \tag{3.4}$$

The maximum probability for defective a software module is defined as $[maxPD_{lj}]_{l=1}^{10}$, l is the index of the base learner which gives a maximum probability of defective for class label c_j. For example $maxPD_{6j}$ show that, 6^{th} base learner gives maximum probability.
Now, we have to adjust the defective rate (σ_d) with combined probability. We set the threshold to determine σ_d; we have used the Hyperbolic tangent activation function (tanh(x)) as an activation function. [249] have used sigmoid function as activation

function for defective rate. Tanh(x) function is better in such cases where so many negative instances are suitable for addressing the class imbalance problem.

TABLE 3.2: Comparison of MCC with various classification methods with BPDET. Note** "-" symbol implies model cannot give output.

Dataset	**BPDET(mean)**	**SVM**	**Naive Bayes**	**Bayesian network**	**Multilayer perceptron**	**PART**
CM1	**0.420**	0.149	**0.224**	0.194	-0.019	0.006
JM1	**0.2319**	0.140	0.208	**0.210**	0.136	0.199
KC1	**0.333**	0.119	0.296	**0.311**	0.309	0.250
KC2	**0.415**	-0.022	0.402	**0.410**	0.409	0.399
KC3	0.137	-	**0.278**	**0.243**	0.217	0.183
MC1	**0.142**	0.116	0.113	**0.135**	0.096	0.128
MC2	**0.332**	-	0.317	**0.325**	0.308	0.235
PC1	**0.382**	0.277	0.218	0.163	0.278	**0.337**
MW1	**0.331**	-0.029	0.275	0.276	0.264	**0.311**
PC2	**0.171**	-	**0.163**	0.117	-0.001	-0.001
PC3	0.166	**0.249**	0.152	**0.275**	0.201	0.100
PC4	**0.469**	0.076	0.348	0.346	**0.452**	0.506

$$T(\sigma_d) = \frac{e^{\lambda*(\sigma_d-\delta)} - e^{-(\lambda*(\sigma_d-\delta))}}{e^{\lambda*(\sigma_d-\delta)} + e^{-(\lambda*(\sigma_d-\delta))}} \tag{3.5}$$

The range of Tσ_d (equation 3.5) is always from -1 to +1. Still, we are considered only in the range from 0 to 1, because it will be irrelevant to consider the negative defective rate. The λ, and δ are positive parameters that are adjusted according to the threshold.

We performed our experiments by varying σ_d from 0 to 1, so that T(σ_d) >0.5. We had adjust positive parameters according to it. We set λ as 100, 200, 300, 350, 400, 450, 475, 480, ad 500 then the adjusted δ are 0.0946, 0.1973, 0.2982, 0.3984, 0.498, 0.5988, 0.698, 0.7987, and 0.898.

After considered the relation between defective rate and $maxPD_{lj}$, we have divided the algorithm in to three different cases.

TABLE 3.3: Comparison of ROC over baseline and EL methods with BPDET.

Dataset	**BPDET(mean)**	Tong et al. [249]	**AdaBoost**	**Bagging**	**Random forest**	**Logistic Boost**
CM1	**0.756**	0.6560	0.7000	0.697	**0.748**	0.724
JM1	**0.752**	0.6587	0.710	0.741	**0.745**	0.713
KC1	**0.811**	0.6717	0.783	**0.805**	0.803	0.784
KC2	**0.835**	0.7645	0.784	0.821	**0.825**	0.824
KC3	0.718	0.7262	0.573	**0.732**	**0.742**	0.627
MC1	0.851	0.861	0.842	**0.867**	**0.883**	0.843
MC2	**0.741**	**0.735**	0.579	0.636	0.682	0.609
PC1	**0.882**	0.8251	0.805	0.853	**0.870**	0.829
MW1	**0.776**	0.736	0.748	0.749	**0.767**	0.741
PC2	**0.810**	0.758	**0.818**	0.808	0.803	0.806
PC3	**0.842**	0.825	0.782	0.818	**0.839**	0.819
PC4	**0.944**	**0.935**	0.913	0.920	0.935	0.818

Case 1: If $\sigma_d < 0.25$ and T $< maxPD_{lj} < 0.5$ (training class is exceptionally imbalance.), we reverse the expected defective probability for l^{th} base learner with combined defective probability, i.e., $[cPND_j, cPD_j] = [PD_{lj},\ PND_{lj}]$.
Case 2: If $\sigma_d < 0.25$ and $maxPD_{lj} > 0.5$, we take the combined probability as probability of l^{th} base learner, i.e., $[cPND_j, cPD_j] = [PND_{lj},\ PD_{lj}]$.
Case 3: If $\sigma_d > 0.25$ and $maxPD_{lj} > 0.5$, we take the predicted probabilities of l^{th} is as combined predicted probability $[PND_{lj},\ PD_{lj}] = [cPND_j, cPD_j]$.

The combination rule is of different types: average probabilities, the product of probabilities, majority voting, median, max/min probability. In BPDET, the average probability is applied as a combination rule, as shown in equation 3.1.
In Fig. 3.1. it has been shown that training data (D_{tr}) from D_1 to D_{10} randomly split into ten different data subsets with replacement so that mutual data point can be there to train each base learner effectively and training can occur without biased

TABLE 3.4: Comparison of ROC with various classification methods against BPDET.

Dataset	BPDET(mean)	SVM	Naive Bayes	Bayesian network	Multilayer perceptron	PART
CM1	**0.756**	0.519	0.658	0.689	**0.734**	0.721
JM1	**0.752**	0.522	0.679	0.701	0.690	**0.712**
KC1	**0.811**	0.518	0.790	**0.791**	0.771	0.747
KC2	**0.835**	0.499	**0.825**	0.824	0.828	0.753
KC3	**0.718**	0.500	**0.658**	0.584	0.654	0.607
MC1	**0.851**	0.512	0.709	**0.721**	0.690	0.695
MC2	**0.741**	0.500	**0.717**	0.616	0.706	0.663
PC1	**0.882**	0.563	0.641	0.694	0.752	**0.787**
MW1	**0.776**	0.495	0.734	**0.750**	0.648	0.618
PC2	**0.810**	0.500	**0.821**	0.791	0.744	0.686
PC3	**0.842**	0.543	0.756	0.773	0.780	**0.828**
PC4	**0.944**	0.505	0.836	0.827	**0.886**	0.848

results, more details given in [130].

$$(D_1, D_2, D_3, ..., D_{10} \subset D_{tr}) \wedge (D_i \cap D_j \neq \emptyset)$$

Those training subsets are then input into ten different classifiers, and those classifiers are IBK, MLP, SVM, RF, NB, Logistic Boost, PART, JRip, J48 consolidated, and Decision Stump.

We have chosen ten base classifiers for the EL phase, we have followed the metric-based classification selection criteria suggested by [135]. As we have used NASA dataset which are of Halsted, basic Halsted and, McCabe software metrics (see Table 2.1) then it will be adequate to choose the Statistical classifier (Naive Bayes), Decision tree approach (Decision stump, J48 consolidated, PART), Ensemble learning-based (Logistic Boost, Random forest), Neural network-based techniques (Multilayer

Table 3.5: Comparison of PRC with various EL method.

Dataset	BPDET(mean)	AdaBoost	Bagging	Random forest	Logistic Boost
CM1	**0.887**	0.873	0.877	0.880	**0.881**
JM1	**0.824**	0.797	**0.817**	0.816	0.799
KC1	**0.880**	0.860	**0.872**	0.866	0.864
KC2	**0.865**	0.829	**0.857**	0.851	0.854
KC3	**0.802**	0.729	**0.798**	0.790	0.756
MC1	**0.974**	0.966	0.965	**0.969**	0.951
MC2	**0.747**	0.622	0.670	**0.713**	0.648
PC1	**0.950**	0.926	0.943	**0.944**	0.936
MW1	**0.924**	**0.915**	0.911	0.915	0.909
PC2	**0.995**	0.985	**0.990**	0.985	0.989
PC3	**0.913**	-	0.902	0.901	**0.903**
PC4	**0.957**	0.868	0.940	0.942	**0.945**

perceptron), Nearest neighbor methods (Instance-based learning) and Support Vector Machine based classifier (SVM). We have also performed various other classification techniques to use as a base learner in voting techniques during our experiments, but they were not much efficient.

3.2.2 Experiment setup

This section will dicsuss the experimental setup which is required for experiments; we will discuss the dataset, the tools and its parameters, software metrics, and parameters for performance evaluation.

3.2.2.1 Datasets

We have used twelve different bug prediction datasets of NASA, from PROMISE repository [221] and tera-PROMISE repository. CM1, JM1 KC1, KC2, KC3, MC1, MC2, PC1, MW1, PC2, PC3, and PC4 are the various datasets that have been utilized for experiments. Some datasets had missing values and repeated instances, which can affect the result of the model. JM1 and MW1 have some missing values; KC1, KC2, PC1, PC2, PC3, JM1, and CM1 have repeated instances. The table 2.3 consists of attributes are several metrics, number of instances before and after preprocessing, number of non-faulty instances back and after preprocessing, and percentage of faulty instances.

3.2.2.2 Tools and its parameters

Experiments for BPDET are performed on the Weka-3.9.2 tool and Anaconda (python) version 5.2.0. The system configuration is an i7 octa-core 7^{th} generation processor, 8GB RAM, 1TB internal memory, and Windows 10 operating system. The voting is an ensemble learning algorithm have used for the experiment in Weka-3.9.2. The best result came after using 70% dataset used for training and 30% for testing data. The training datasets randomly divided into ten subsets with repetition, means data items are mutual between weak classifier. Ten-fold cross-validation was used in the experiment. Ten classifiers will be selected as seen in Fig. 3.1, "combination rule" is an average probability (see equation 3.1).

3.2.2.3 Software metrics

Twelve datasets used in the BPDET experiment as shown in the table 2.1, twenty-one software metrics which are essential for the operation are described in table 2.3. The basis Halsted metrics, Derived Halsted metrics, and McCabe metrics are the various software metrics used in experiments.

3.2.2.4 Performance evaluation parameters

The performance evaluation of the BPDET includes the area under the curve (AUC) of ROC (receiver operating characteristic curve), F-measure, Mathews correlation

coefficient (MCC), and precision-recall area (PRC). Sometimes the precision-recall curve is more useful than ROC because the ROC only gives an idea of how the classifier is performing in general because it considers the positive and negative classes equally. If someone is interested in how the classifier is engaged in a particular class, then PRC is more useful. As the MCC is more effective when classes are of different sizes, which means it is effective to address class imbalance problem. It also measures the biased nature of binary classifiers. The PRC curve also targets to class imbalance problem, as PRC is not considered "True Negative"; it measures the balance between the two classes. Let's consider a convention that the faulty module will be positive samples, and non-faulty modules will be negative samples.

3.3 Result and discussion

This section will explain the performance of BPDET and compare it with ten baseline methods. We will also elaborate the performance of BPDET over different corruption rates. After that, we will address all three research queries that we have framed in section 3.1. In the last section 3.3.4, we will discuss the insightful discussion about BPDET.

We have compared the performance of BPDET with the latest baseline methods. The bold & underline text in tables (Table 3.1 to Table 3.8) of BPDET column are showing the best performance among all other SBP methods, whereas the bold text represents the second-best performance value of SBP models. In table 3.1 the comparison of MCC values with [249] and other ensemble learning methods. MCC values of BPDET is highest for CM1 (0.420), KC2 (0.382), MC2 (0.332), PC1 (0.382), MW1 (0.331), & PC2 (0.171) among every EL baseline method. Among 8 out of 12 data, BPDET outperforms concerning baseline and other EL methods.

Similarly, the table 3.2, MCC values of BPDET were compared with the various primary classifier based SBP models, the table 3.2 shown that the highest MCC of BPDET are for CM1 (0.420), JM1 (0.232), KC1 (0.333), KC2 (415), MC1 (0.142), MC2 (0.332), PC1 (0.382), MW1 (0.331), PC2 (0.171), & PC4 (0.469). In 10 out of 12 data BPDET surpass the MCC values with other traditional SBP models.

In Fig. 3.4(a), the MCC values of BPDET over every dataset are compared with the best four fault prediction models. It can be seen in the figure, that MCC of BPDET is better for CM1, JM1, KC1, KC2, MC2, PC1, MW1, & PC2, whereas for datasets

MC1 (0.3287), PC3 (0.3095) & PC4 (0.5758) the highest MCC value produced by SBP of [249]. AdBoost based SBP have the highest MCC for KC3 (0.344) dataset. The ROC is the foremost evaluation parameter to evaluate the accuracy of any prediction model. In table 3.3 and table 3.4 we compared the ROC of BPDET concerning ten other SBP techniques. In table 3.3 the ROC is compared with five baseline methods including few EL methods, from the table 3.3 it can be easily seen that ROC of BPDET is highest for datasets CM1, JM1, KC1, KC2 MC2 and PC1, MW1, PC3, PC4 and their respective values are 0.756, 0.752, 0.811, 0.835, 0.741, 0.882, 0.776, 0.842 and 0.944 respectively, whereas for KC3 (RF), MC1 (RF), PC2 (AdaBoost), 0.742, 0.883 and 0.818 respectively. In Fig.3.4(b), we have compared the ROC for various datasets with the best four-fault prediction models. BPDET surpasses ROC values compared with other techniques for CM1, JM1, KC1, KC2, MC2, PC1, MW1, PC3, and PC4.

Similarly, table 3.4 compared the various ROC of BPDET concerning different classifiers. ROC of BPDET is maximum in 11 out of 12 datasets, as in the bold & underline letter of table 3.4 indicates the maximum ROC of any of the SBP methods. Naive Bayes beats the performance of BPDET for the PC2 dataset, the ROC for PC2 data by Naive Bayes is 0.821, but the BPDET is the second-best performer with the 0.810 ROC value.

Table 3.5 shows the comparison of PRC values for all twelve different datasets of the BPDET model and compared with four EL methods. In the table, we can easily see that bold & underline text shows the highest PRC values and bold as second highest PRC values among all EL techniques. The PRC value of BPDET in all dataset is highest, i.e CM1, JM1, KC1, KC2, KC3, MC1, MC2, PC1, MW1, PC2, PC3, and PC4 and their corresponding values are 0.887, 0.824, 0.880, 0.865, 0.802, 0.974, 0.747, 0.950, 0.924, 0.995, 0.913, and 0.957 respectively. The second-best performer is Bagging; it beats other EL models in seven datasets (JM1, KC1, KC2, KC3, MC1, PC2, & PC3). BPDET has the highest PRC values compared with all ten SBP models, so BPDET is much more effective for class imbalance challenges with respect to existing techniques. Fig. 3.4(c) reports that PRC produced by BPDET for every dataset is maximum between all top models. After BPDET, Logistic Boost outperforms other EL-based methods.

In table 3.6 we have compared the PRC values produced by BPDET with five basic classification techniques; the PRC values of the BPDET model are highest among all classical classification techniques for all datasets. The PRC of values produced

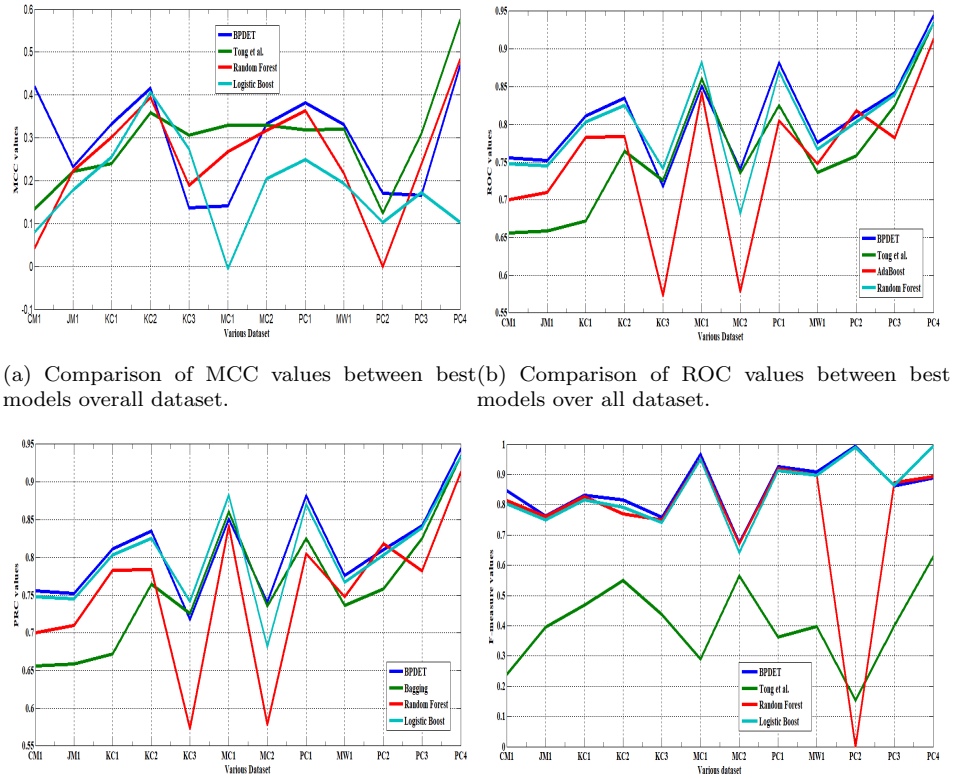
by the PART model is the second-best performer; it is effective in four datasets (CM1, JM1, PC1, & PC2), whereas MLP (KC2, PC3, & PC4) and Naive Bayes (KC3, MC1 & MC2) also a second-best performer for three datasets.

TABLE 3.6: Comparison of PRC with various classification method.

Dataset	BPDET (mean)	SVM	Naive Bayes	Bayesian network	Multilayer perceptron	PART
CM1	**0.887**	0.828	0.859	0.865	0.879	**0.880**
JM1	**0.824**	0.698	0.783	0.795	0.789	**0.796**
KC1	**0.880**	0.745	0.860	**0.861**	0.851	0.835
KC2	**0.865**	0.674	0.852	0.856	**0.860**	0.799
KC3	**0.802**	0.698	**0.778**	0.725	0.778	0.742
MC1	**0.974**	0.956	**0.968**	0.967	0.966	0.965
MC2	**0.747**	0.547	**0.735**	0.628	0.722	0.667
PC1	**0.950**	0.885	0.894	0.904	0.921	**0.925**
MW1	**0.924**	0.857	0.904	**0.911**	0.886	0.889
PC2	**0.995**	0.971	0.981	0.984	0.989	**0.991**
PC3	**0.913**	0.830	0.883	0.892	**0.893**	0.870
PC4	**0.957**	0.787	0.901	0.902	**0.933**	0.909

The highest PRC value of BPDET is for the PC2 (0.995) dataset, and after that MC1 (0.974) data, it reflects the idea that MC1 and PC1 have more minor class imbalance problem. Whereas MC2 (0.747) and KC3 (0.802) have the lowest PRC values, so comparable, they can have low-class imbalance issues.

F-measure is also helpful for the evaluation of accuracy for SBP models. In table 3.7 F-measure is compared to baseline and four EL models, whereas in table 3.8 compare the F-measure with five classifications based SBP models with respect to BPDET. In both the tables, BPDET has the highest F-measure for all twelve datasets. In 10 out of 12 datasets BPDET outperformed compared with baseline and EL methods i.e., CM1, JM1, KC1, KC2, KC3, MC1, MC2, PC1, MW1, and PC2 are 0.847, 0.763, 0.832, 0.816, 0.759, 0.967, 0.675, 0.926, 0.909, and 0.994, respectively. RF outperform on PC3, PC4 projects with f-score of 0.873 and 0.894, respectively; even RF



(a) Comparison of MCC values between best models overall dataset.

(b) Comparison of ROC values between best models over all dataset.

(c) Comparison of PRC values between best models over all dataset.

(d) Comparison of F-measure values between best models over all dataset.

FIGURE 3.4: Comparison of MCC, ROC, PRC, F-measure between best fault prediction techniques over all dataset.

is the second-best performer after BPDET in six datasets (JM1, KC2, KC3, MC1, MC2, & PC1). In Fig. 3.4(d) we have compared the F-measure of BPDET with the four highest f-score models; it can be seen that BPDET have maximum F-measure values for ten datasets compared with other techniques.

In Fig.3.4, we have summarized every performance evaluation metrics for the bug prediction model, which shows the efficiency of BPDET compared with other baseline or existing methods.

Multilayer perceptron, PART, and AdaBoost also have much higher f-score values than the other methods. After comparing the evaluation parameters of BPDET with four EL-baseline, and five classical SBP techniques, we have analyzed the performance metrics of each dataset. We examined MCC, ROC, PRC, and F-measure one by one.

TABLE 3.7: Comparison of F-measure over baseline and EL methods with BPDET. Note** "-" symbol implies model cannot produce output.

Dataset	**BPDET(mean)**	Tong et al. [249]	**AdaBoost**	**Bagging**	**Random forest**	**Logistic Boost**
CM1	**0.847**	0.239	-	**0.841**	0.814	0.802
JM1	**0.763**	0.395	0.727	0.754	**0.760**	0.749
KC1	**0.832**	0.468	0.806	0.824	**0.826**	0.815
KC2	**0.816**	0.555	**0.810**	0.752	0.769	0.792
KC3	**0.759**	0.438	0.741	0.752	**0.749**	0.742
MC1	**0.967**	0.289	-	-	**0.952**	0.950
MC2	**0.675**	0.565	0.601	0.659	**0.671**	0.643
PC1	**0.926**	0.362	-	0.917	**0.919**	0.914
MW1	**0.909**	0.3969	**0.901**	0.891	0.898	0.898
PC2	**0.994**	0.152	-	-	-	**0.991**
PC3	0.863	0.400	-	0.854	**0.873**	**0.864**
PC4	0.889	0.6288	0.868	0.886	**0.894**	**0.892**

TABLE 3.8: Comparison of F-measure with various classification method. Note** "-" symbol implies model cannot give output.

Dataset	**BPDET(mean)**	**SVM**	**Naive Bayes**	**Bayesian network**	**Multilayer perceptron**	**PART**
CM1	**0.847**	**0.833**	0.828	0.718	0.825	0.831
JM1	**0.763**	0.739	0.751	0.710	0.741	**0.752**
KC1	**0.832**	0.787	0.820	0.739	**0.825**	0.817
KC2	**0.816**	0.703	0.801	0.797	**0.810**	0.802
KC3	**0.759**	-	0.745	0.735	0.738	**0.726**
MC1	**0.967**	**0.945**	0.889	0.857	0.939	0.937
MC2	**0.675**	-	0.657	**0.662**	0.642	0.615
PC1	**0.926**	0.914	0.895	0.791	0.917	**0.918**
MW1	**0.919**	0.881	0.864	0.871	0.900	**0.909**
PC2	**0.994**	-	0.983	0.943	**0.990**	0.985
PC3	**0.863**	**0.858**	0.573	0.760	0.845	0.855
PC4	**0.889**	0.824	0.861	0.787	**0.890**	0.884

TABLE 3.9: Comparison of total training time(seconds) taken by the various ensemble learning methods and some base classifier with respect to BPDET.

Dataset	**BPDET**	**AdaBoost**	**Bagging**	**Random Forrest**	**Logistic Boost**	**SVM**	**Naive Bayes**	**Multi layer perceptron**	**PART**
CM1	12.64	0.051	0.031	0.091	0.021	0.072	0.0001	1.331	0.0021
JM1	918.52	0.32	1.691	5.66	0.291	130.38	0.032	24.52	1.775
KC1	40.051	0.052	0.151	0.561	0.051	0.731	0.0001	4.775	0.130
KC2	7.461	0.021	0.032	0.122	0.010	0.063	0.0001	1.811	0.021
KC3	8.51	0.022	0.021	0.041	0.010	0.021	0.0001	1.210	0.123
MC1	75.11	0.122	0.171	0.361	0.083	1.341	0.012	11.532	0.012
MC2	4.46	0.020	0.022	0.042	0.010	0.011	0.0001	0.770	0.0101
PC1	16.53	0.061	0.061	0.212	0.031	0.342	0.0001	2.370	0.032
MW1	23.513	0.13	1.5	0.31	0.12	0.006	0.01	2.29	0.08
PC2	163.22	0.44	4.4	1.06	0.25	0.316	0.03	35.41	0.35
PC3	58.23	0.08	1.9	0.50	0.08	0.086	0.01	11.62	0.17
PC4	61.25	0.09	1.8	0.37	0.06	0.069	0.01	8.18	0.9

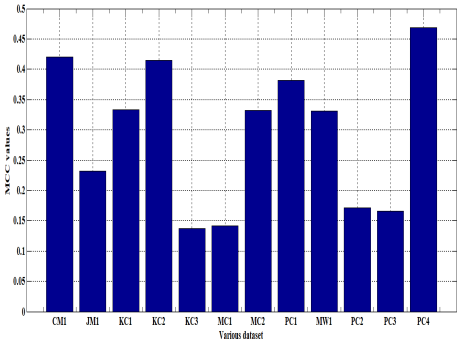
TABLE 3.10: Comparison of MCC, ROC, PRC, f-score on k-fold cross validation over BPDET

Data	**5-fold**				**8-fold**				**10-fold**				**12-fold**				**15-flod**			
	MCC	*ROC*	*PRC*	*f-score*	*MCC*	*ROC*	*PRC*	*f-score*	*MCC*	*ROC*	*PRC*	*f-score*	*MCC*	*ROC*	*PRC*	*f-score*	*MCC*	*ROC*	*PRC*	*f-score*
CM1	0.391	0.855	0.887	0.746	0.411	0.760	0.891	0.854	0.420	0.756	0.887	0.847	0.231	0.750	0.886	0.850	0.156	0.781	0.898	0.856
JM1	0.202	0.752	0.824	0.759	0.223	0.748	0.781	0.765	0.221	0.752	0.824	0.763	0.214	0.756	0.826	0.763	0.201	0.753	0.825	0.760
KC1	0.321	0.819	0.882	0.830	0.328	0.814	0.883	0.830	0.333	0.811	0.880	0.832	0.331	0.818	0.882	0.831	0.317	0.821	0.885	0.828
KC2	0.427	0.829	0.861	0.819	0.427	0.835	0.857	0.819	0.415	0.835	0.865	0.816	0.457	0.830	0.860	0.867	0.435	0.836	0.856	0.822
KC3	0.112	0.720	0.803	0.753	0.131	0.725	0.804	0.74	0.137	0.718	0.802	0.759	0.226	0.727	0.795	0.780	0.160	0.717	0.797	0.764
MC1	0.008	0.832	0.975	0.964	0.009	0.866	0.978	0.965	0.142	0.851	0.974	0.967	0.006	0.847	0.976	0.965	0.068	0.825	0.974	0.966
MC2	0.409	0.767	0.771	0.725	0.279	0.714	0.668	0.725	0.332	0.741	0.747	0.675	0.321	0.790	0.800	0.676	0.367	0.737	0.762	0.699
PC1	0.329	0.889	0.949	0.921	0.312	0.874	0.949	0.919	0.382	0.882	0.950	0.926	0.374	0.876	0.948	0.926	0.336	0.876	0.946	0.922
MW1	0.288	0.725	0.915	0.908	0.333	0.752	0.920	0.913	0.331	0.776	0.924	0.909	0.323	0.747	0.916	0.912	0.369	0.792	0.928	0.917
PC2	0.098	0.768	0.994	0.994	0.167	0.854	0.994	0.995	0.171	0.810	0.995	0.994	0.145	0.871	0.995	0.994	0.134	0.798	0.994	0.994
PC3	0.173	0.837	0.910	0.864	0.157	0.830	0.913	0.869	0.166	0.842	0.913	0.863	0.114	0.828	0.908	0.857	0.207	0.851	0.918	0.868
PC4	0.441	0.941	0.954	0.884	0.477	0.942	0.956	0.891	0.469	0.944	0.957	0.889	0.476	0.947	0.958	0.890	0.506	0.943	0.957	0.866

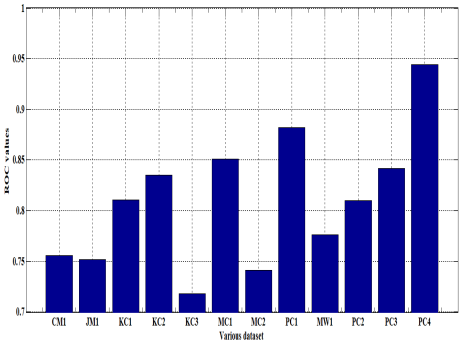
Fig. 3.5 shows the bar graph of various performance measures produced by BPDET for all examined datasets; it has been demonstrated in the bar-graph of Fig. 3.5(a) that MCC value for data PC4 (0.469) and CM1 (0.42) is the highest, and lowest for KC3 (0.137) and MC1 (0.142).
In Fig. 3.5(b), the ROC of all twelve datasets are examined and shown. The highest ROC value yield by BPDET are 0.944 and 0.887 for PC4 and PC1, respectively, whereas 0.741 and 0.752 are the lowest ROC values for MC2 and CM1 datasets, respectively.
Fig. 3.5(c) shows the PRC values of BPDET of all datasets, as the figure reports that the PRC value of PC2 (0.995) and MC1 (0.974) are the highest, whereas MC2 (0.747) and KC3 (0.802) projects are have lowest values.
In Fig. 3.5(d) represents the F-measure values of BPDET over various datasets, for PC2 (0.994) MC1 (0.967) the F-measure are highest and for MC2 (0.675) and KC3 (0.759) are the lowest among all datasets.

3.3.1 Effect of corruption rate over performance of BPDET

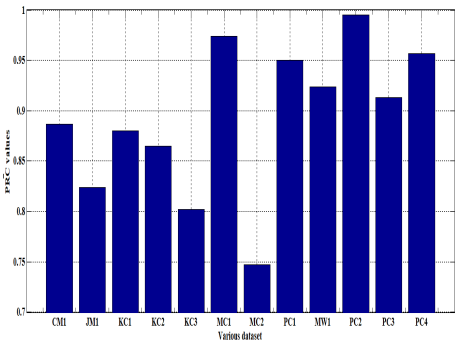
The variation of corruption rate, as discussed in section 3.2.1.3, it affects the performance of the BPDET model. Evaluation metrics values are different at various corruption rates, and they can be easily detectable by conducting experiments at different corruption rates. To get a deeper analysis of the BPDET model, we must have considered the effect of the corruption rate (σ_d). In Fig. 3.7, we have summarized the effect of the corruption rate of BPDET over performance metrics.
Table 3.11 shows the adjustment variations of defective rate σ_d with positive parameters λ and δ. We have considered the threshold more than 0.5, so after calculating the value of $e^{\lambda*(\sigma_d-\delta)}$ by using equation 3.5, i.e., equal to 0.54. The value of σ_d from 0 to 1 with 0.1 value of difference. Total adjusted values of λ and δ are shown in the table 3.11. The box-plot graph of λ and δ are shown in Fig. 3.6(a) and Fig. 3.6(b). The corruption rate variation causes dissimilar performance results; it can be easily trackable at different corruption rates. Fig. 3.7 shows the difference in performance metrics on every dataset when the corruption rate varies. In Fig. 3.7(a) we have plotted the MCC values at 0 to 0.9 value of corruption rate. Whereas in Fig. 3.7(b) we have compared the variation of ROC w.r.t corruption rates for all twelve datasets. In Fig. 3.7(c) and Fig. 3.7(d) shown the different results of PRC values and F-measure values for all datasets at 0 to 0.9 corruption rate.



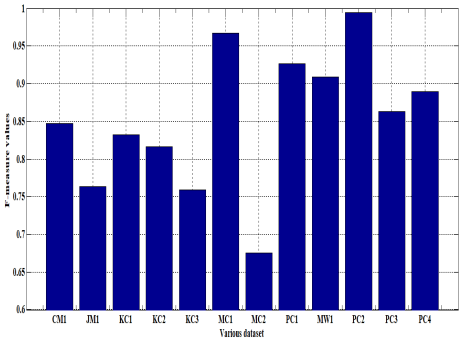
(a) BPDET result: MCC over various dataset.



(b) BPDET result: Area under curve of ROC over various dataset.



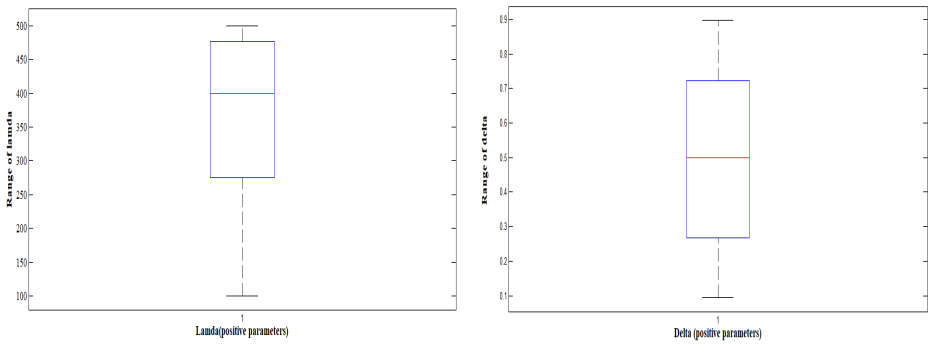
(c) BPDET result: PRC values over various dataset.



(d) BPDET result: F-measure values over various dataset.

FIGURE 3.5: Comparison of MCC, ROC, PRC, F-measure of BPDET over all 12 dataset.

In Fig. 3.8, we have shown the boxplot graph of performance metrics of each dataset at different corruption rates. The graph also shows the range values (minimum to maximum) of all four performance metrics for every dataset. In Fig. 3.8(a), the boxplot graph of MCC values is shown, Fig. 3.8(b) shows the ROC range at different corruption rate, Fig. 3.8(c) and Fig. 3.8(d) shows the PRC and F-measure of each dataset.



(a) Boxplot of lamda (positive parameter), the range of lamda that used in the experiment.

(b) Boxplot of delta (positive parameter), the range of delta that used in the experiment.

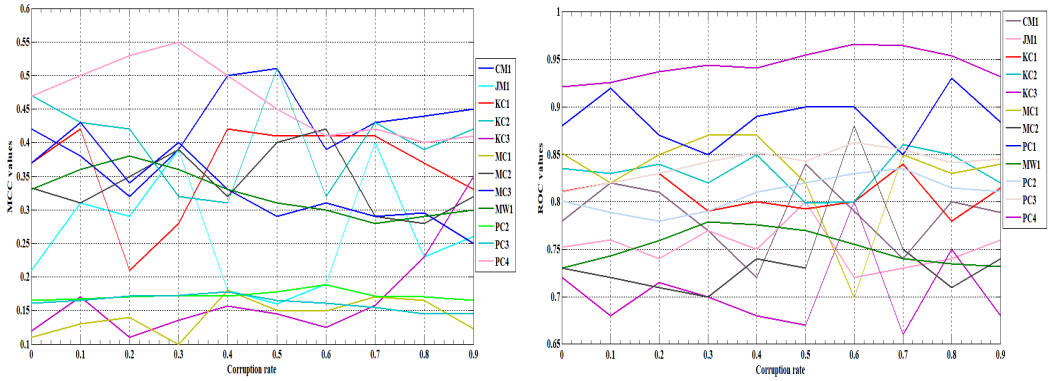
FIGURE 3.6: Boxplot of positive parameters λ and δ for corruption rate(σ_d).

3.3.2 Research queries discussion

The research problems which were asked in section 3.1, we are going to address those questions. Each research query is related to the performance of the proposed approach, defective datasets, the class imbalance & over-fitting problem, and time comparison with other SBP models. The justification of every research query (RQ) is given below.

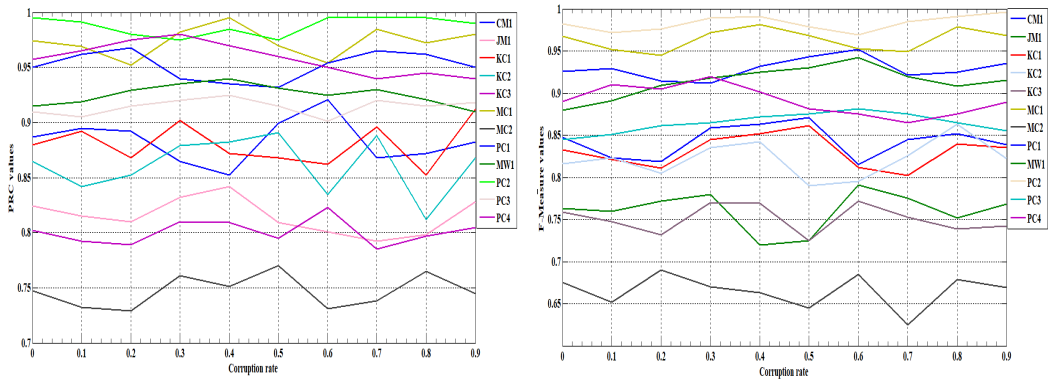
3.3.2.1 Effectiveness of BPDET in terms of performance metrics compared with other fault prediction model

The BPDET model is an effective SBP model, as we have seen earlier in section that MCC, ROC, PRC, and F-measure of BPDET are compared with baseline methods, i.e., four EL techniques and five latest classical SBP methods. The tables 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, and 3.9 shows the comparative analysis of MCC, ROC, PRC, and F-measure of EL with baseline model and various traditional techniques, respectively. Whereas Fig. 3.5(a), 3.5(b), 3.5(c), and 3.5(d) shows the effectiveness of BPDET model with respect to MCC, ROC, PRC, and F-measure, respectively. In Fig. 3.4, we have compared the state-of-the-art prediction models using the performance metrics. The graphs reports in Figures. 3.4(a) 3.4(b)), 3.4(c), 3.4(d) are performance metrics of top four SDP models, in terms of MCC, ROC, PRC and F-measure, respectively.



(a) Variation of MCC values of BPDET with respect to curruption rate for every dataset.

(b) Variation of ROC values of BPDET with respect to curruption rate for everydataset.



(c) Variation of PRC values of BPDET with respect to curruption rate for every dataset.

(d) Variation of F-measure values of BPDET with respect to curruption rate for every dataset.

FIGURE 3.7: Variation of MCC, ROC, PRC and F-measure of BPDET with respect to corruption rate for every dataset.

3.3.2.2 How useful BPDET model compared with traditional methods regarding the class imbalance and over-fitting problem

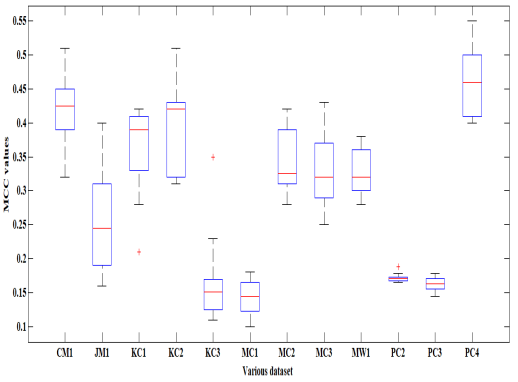
Over-fitting is defined as the model that performs better while in training and worst in testing.

(i) Data noise/ data inconsistent

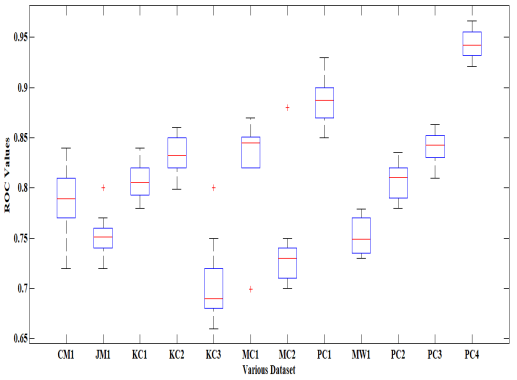
(ii) Deficient training data

(iii) Construction of complex mode

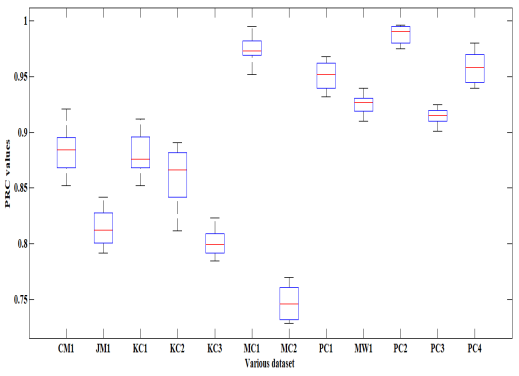
The ensemble learning method is one of the techniques to avoid over-fitting, [25] suggested EL is beneficial to avoid over-fitting in ML-based prediction models. We



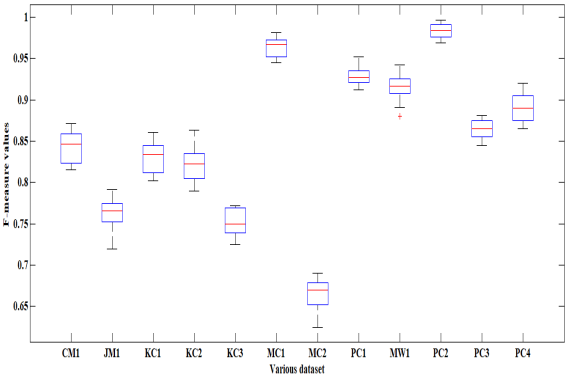
(a) Boxplot of MCC over various dataset. It shows the variation of MCC when the σ_d varies.



(b) Boxplot of ROC over various dataset. It shows the variation of ROC when the σ_d varies.



(c) Boxplot of PRC over various dataset. It shows the variation of MCC when the σ_d varies.



(d) Boxplot of F-measure over various dataset. It shows the variation of F-measure when the σ_d varies.

FIGURE 3.8: Boxplot representation of MCC. ROC, PRC and F-measure for every dataset when the corruption rate(σ_d) varies.

have applied regularization techniques in our proposed model; the dropout [236] and early stop [271] are also efficient to avoid over-fitting. Averaging the classifier result can also protect the model from over-fitting. According to [66], the average of the classifier's result can able to reduce the over-fitting problem, as we have used average probability (equation 3.1) as a combination rule.

EL not only protects against over-fitting but also fights for the class imbalance problem. Re-sampling and cost-sensitive learning are other methods to avoid class imbalance problem; that is the main reason to apply two layers of EL stage. Other EL methods such as AdaBoost, Bagging [240, 53] are not effective for the class imbalance problem. BPDET employed two other EL techniques as a base learner, and these Random Forrest and logistic boost. If some EL methods are used as a base learner in the BPDET model, they can efficiently reduces class imbalance problem.

The main reasons of BPDET to avoid class imbalance and over-fitting problems are 2 layers of EL method, regularization techniques, and combination rule. The first layer of EL is trying to reduce these two challenges. The combined rule, which is the average probability [66] also helps to minimize these two challenges. MCC and PRC values of the proposed model are higher than state-of-the-art techniques, as discussed earlier. It indicates that the proposed model avoids these two issues.

3.3.2.3 How much training time taken by the BPDET model

Training time is an important aspect of measuring an ML-based predictive model; the total time is the sum of staked denoising auto-encoder (SDA) phase, training, and testing of an ensemble learning phase of the BPDET model. Total time is the summation of all these times.
Total execution time(sec) = SDA time + EL(Training(D_{tr}) + Testing(D_{ts})) time.
So; Total training time = DL phase (Training) + EL phase (Training) = Total execution time - Testing time
Table 3.11 shows the total training time, i.e., training of EL phase and SDA phase, whereas sampling time (SMOTE) and training time of other SBP techniques over various datasets. As we can see, the training time by BPDET for each dataset is very high, it is because of the model with two complex phases.
As shown in Fig. 3.9, JM1 has the maximum training time taken, i.e., 918.52 sec, whereas PC2 has the second maximum training time with 163.22 sec, which is very lesser as compare to JM1; the main reason is that it has 10886 instances, out of which 2106 defective instances (19.35%) and 8779 non-defective instances (80.65%). MC2 and KC2 have the lowest training time, i.e., 4.46 sec and 4.461 sec, respectively. The time taken by the BPDET model is high for every dataset with respect to other techniques because of the following reasons.

(a) The deep learning phase takes time

(b) Ten different classifier in a BPDET model (EL model)

(c) Second layer of EL in BPDET

TABLE 3.11: Adjustment of parameters(λ, δ) with defective rate to achieve threshold T(σ_d).

σ_d	λ	δ
0.1	100	0.0946
0.2	200	0.1973
0.3	300	0.292
0.4	350	0.3984
0.5	400	0.4988
0.6	450	0.5988
0.7	475	0.698
0.8	480	0.7987
0.9	500	0.898

3.3.3 Significant analysis

In this subsection, we have conducted a non-parametric test on BPDET and other SFP models using Wilcoxon Rank-Sum Test [73]. We have considered four different average values of MCC, ROC, PRC, and f-score as a performance measure, the first, second, third, and fourth row of Table 3.12 are average values of MCC, ROC, PRC, and f-score of different SBP models. We have taken four various means values of BPDET in S1 and best performing SBP models in S2. In sample one (S1), we input these metrics values, whereas, in sample two (S2), we gave second best-performed values of these corresponding metrics, for example: if an SBP "A" has the second-highest ROC values but third highest MCC values, so we took only the ROC values of the model A. The sum of ranks for S1 is: $R_1 = 156$, and the sum of ranks of the S2 is: $R_2 = 120$. Hence, the test statistic is $R = R_1 = 156$. The following null and

TABLE 3.12: Two different sample of BPDET and other best models. The full description of the table in the section 3.3.3.

S1 (BPDET)	**S2 (other techniques)**
0.294	0.260 (Tong et al.)
0.819	0.761 (Tong et al.)
0.894	0.881 (RF)
0.856	0.837 (Logistic Boost)
0.304	0.261 (Tong et al.)
0.820	0.767(AdaBoost)
0.899	0.880 (Bagging)
0.850	0.827 (PART)
0.297	0.262 (Naive Bayes)
0.821	0.763 (Logistic Boost)
0.904	0.870 (MLP)
0.852	0.824 (PART)

alternative hypotheses need to be tested: H_0 = Median (Difference) = 0
H_a : Median (Difference) > 0

TABLE 3.13: Ranking of values from samples. The full description of the table in the section 3.3.3.

Sample	**Values**	**Rank**
S2	0.260	1
S2	0.261	2
S1	0.294	3
S1	0.297	4
S1	0.304	5
S2	0.761	6
S2	0.763	7
S2	0.767	8
S1	0.819	9
S1	0.820	10
S1	0.821	11
S2	0.824	12
S2	0.827	13
S2	0.837	14
S1	0.850	15
S1	0.852	16
S1	0.856	17
S2	0.870	18
S2	0.880	19
S2	0.881	20
S1	0.894	21
S1	0.899	22
S1	0.904	23

We observed that both sample sizes are greater than 10, then we can use a normal approximation. The following z-statistic l be used, based on the information provided, the significance level is $\alpha = 0.025$, and the critical value for a right-tailed test is $z_c = 1.96$. The rejection region for this right-tailed test is $R = z : z > 1.96$.
Test Statistics: z-statistic is computed as follows:

$$z = \frac{R - n_1(n_1 + n_2 + 1)/2}{n_1 n_2 (n_1 + n_2 + 1)/12} = 0.739$$

Here n_1, and n_2 are sample sizes, the decision about the null hypothesis: Since it is observed that $z = 0.739 \leq z_c = 1.96$, it is then concluded that the null hypothesis is rejected. Using the p-value approach: The p-value is 0.2301, and since $p = 0.2301 \geq 0.025$, it is concluded that the null hypothesis is rejected, so H_o is rejected. Therefore, there is not enough evidence to claim that the sample median of differences is greater than 0 at the 0.025 significance level.

3.3.4 Insightful discussion

The BPDET model is significantly efficient to perform over each twelve NASA datasets, but we have also found BPDET is not the best predictive model on a few datasets concerning evaluation metrics, as its values are lesser than the other techniques. In this section, we will intuitively discuss the flaws of BPDET. In table 3.1, the MCC value is maximum for datasets MC1, PC3, PC4 and their respective values are 0.329, 0.309, 0.576 respectively by [249], whereas for KC3 data the maximum MCC is 0.344 by AdaBoost. Table 2.3 indicates that the faulty instance of MC1 is too low, i.e. 2.30%, similarly the PC2 have only 2.15% of faulty instance, which is highly imbalanced, as from equation 2.18 we can see that when the value of False Negative is high, then MCC will be lesser, so it can be a reason of the lesser value of MCC. Whereas in KC3, total instance is 194, the defective instance is 36 (see table 2.3) approx 18.55% defective. There are fewer instances, so maybe these can be one of the reasons, as False Negative rate can be more so, MCC will be less. Similarly, the Table 3.3 shows that ROC of KC3 (0.718), MC1 (0.851), PC2 (0.810) of BPDET model are also lesser compared with other SBP techniques. As discussed, MC1 & PC2 are highly imbalanced, and KC3 has a lesser number of instances; maybe these can be reasons for it. One more essential consideration is overfitting in the Random forest algorithm, as BPDET uses ten base classifiers with lesser chances of overfitting. EL method is effective in handling the over-fitting issue. Maximum F-measure values are of PC3 (0.873) and PC4 (0.894) produced by the RF-based SBP method, which surpasses the performance of BPDET. PC3 and PC4 have more number of software metrics and lesser number of the instance so data sparsity arises, which can cause poor performance by BPDET.

We have also analyzed the stability of the BPDET using the k-fold cross-validation test. In Table 3.10 we have listed the MCC, ROC, PRC and f-score values produced by the BPDET model over 5, 8, 10, 12 and 15 fold cross-validation on every datasets. As the table 3.10 reports, for each dataset, the values of MCC, ROC, PRC, and f-score are approx equal till 12-fold cross-validation. After the 12 fold cross-validation, the values of performance metrics started decreasing, i.e., there is a small difference in the performance on 12-fold and 15 fold cross-validation. Although at 15 fold, the values slightly reduce, which indicates BPDET started losing its stability. If the value of evaluation metrics is still approximately equal, which implies the higher stability of the model, BPDET performs similarly.

3.4 Threats to validity

There are so many threats that can affect the result of the experiment. Let us categorize those threats as internal and external threats.
The research proposal presented in this chapter depends on selected systems, which do not consider all the industrial domain possibilities. Many researchers questioned [80, 220] the quality of NASA datasets. Due to these many reasons, there can be no proper representation of the size of classes which can be treated as external threats that affect the accuracy of the proposed model.
According to some past studies [148, 12] performance measure parameters such as *Accuracy, Balance, F-measure, G-measure, G-mean1, G-mean2 PD, PF* have been used to analyze the performance of the binary classifier based SBP model, but we have used MCC, ROC, PRC, and F-measure for better analyses of the SBP model.

3.5 Summary

Software bug prediction is an important aspect of software reliability to ensure high-quality software systems. We proposed a novel SBP model based on deep learning and ensemble learning techniques. Our proposed model BPDET has two stages: the deep learning stage and the ensemble learning stage. We have applied Staked denoising auto-encoder(SDA) into BPDET to extract the deep representation of metrics from traditional software metrics. The deep learning phase mainly addresses the class imbalance and over-fitting problems, and two stages of EL also target these two issues. We have chosen twelve NASA public datasets for experiments, applied 10 fold cross-validation, and performed each experiment up to thirty times. We analyzed the performance of BPDET concerning MCC, ROC, PRC, F-measure and compared it with baseline methods, EL-based, and traditional SBP methods. After analysis, we found that performance metrics on most of the dataset for BPDET model is higher than the existing state-of-the-art techniques. We have also tested the model over 5, 8, 10, 12, and 15 fold cross-validation to measure the stability. We have found that the values MCC, PRC, ROC, and f-score are approximately equal till 12 fold cross-validation and started reducing after it. We have also validated the BPDET using Wilcoxon rank-sum test at alpha 0.025 and rejected the null hypothesis, i.e., H_0 = Median (Difference). The experiment shows that deep

learning is effective for extracting the deep representation of conventional software metrics and avoiding class imbalance & over-fitting problems. Deep representation combined with ensemble learning to build a better SBP model concerning baseline and traditional techniques.

In this work, we only identify the module or class is buggy or not; estimating the number of faults in every module provides more precise information. We now extend our work over predicting a number of defects in every module. Predicting the number of defects in each module is more useful because it can optimally divide testing resources. The module or class consists of more bugs that may require more testing effort.