An empirical study toward dealing with noise and class imbalance issues in software defect prediction

Sushant Kumar Pandey\(^1\) \cdot Anil Kumar Tripathi\(^1\)

Accepted: 29 July 2021 / Published online: 13 August 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2021

Abstract
The quality of the defect datasets is a critical issue in the domain of software defect prediction (SDP). These datasets are obtained through the mining of software repositories. Recent studies claim over the quality of the defect dataset. It is because of inconsistency between bug/clean fix keyword in fault reports and the corresponding link in the change management logs. Class Imbalance (CI) problem is also a big challenging issue in SDP models. The defect prediction method trained using noisy and imbalanced data leads to inconsistent and unsatisfactory results. Combined analysis over noisy instances and CI problem needs to be required. To the best of our knowledge, there are insufficient studies that have been done over such aspects. In this paper, we deal with the impact of noise and CI problem on five baseline SDP models; we manually added the various noise level (0–80%) and identified its impact on the performance of those SDP models. Moreover, we further provide guidelines for the possible range of tolerable noise for baseline models. We have also suggested the SDP model, which has the highest noise tolerable ability and outperforms over other classical methods. The True Positive Rate (TPR) and False Positive Rate (FPR) values of the baseline models reduce between 20–30% after adding 10–40% noisy instances. Similarly, the ROC (Receiver Operating Characteristics) values of SDP models reduce to 40–50%. The suggested model leads to avoid noise between 40–60% as compared to other traditional models.

Keywords
Software testing \cdot Software fault prediction \cdot Class imbalance \cdot Noisy instance \cdot Machine learning \cdot Software metrics \cdot Fault proneness

1 Introduction
Software defect prediction (SDP) (Hall et al. 2011; Pandey et al. 2021, 2020a, b; Malhotra 2015) attempts to identify most likely fault-prone modules in the software project by utilizing software metrics (Radjenović et al. 2013; Shihab 2012; Schneider et al. 1992). It is always advisable to carefully and meaningfully execute testing of fault-prone modules rather than treating all the modules in a similar manner. SDP models make use of bug reports for representatives in old software that indicates faulty and non-faulty modules. Module’s metric information of a software system (Offutt 1992) is used for the training of defect prediction models. SDP models may make use of change-log of software configuration management documentation, as the change-log indeed reports the modules that experience change upon correction when faults being detected. SDP models that use well-known traditional classifiers as a classification technique to predict buggy or clean modules we called them classical SDP or traditional SDP models. All the classifiers that we have used for experimental purposes are classical SDP models. These SDP models and their variants are widely applied in the SDP domain, so they are also baseline methods in defect prediction. We have used five baseline methods for the experimental purpose discussed in Sect. 4.3.2.

The links between logs and bug reports may be inconsistent because of few reasons (Linberg 1999) and may also cause mislabeled data. Therefore, quite likely that an SDP model may be working with noisy data and leads to erroneous results. When the cardinality of one of the classes is much smaller than the other class, the dataset is said to be
imbalanced data. Seiffert et al. (2014) reported an analysis over combined study on both noise and class imbalance (CI) problems in software quality. They have conducted experiments over eleven classification techniques and seven sampling methods over public datasets. They concluded that few classifier combined with sampling methods that are most confronted over the noisy and imbalanced dataset. To the best of our knowledge, the combine interaction between CI and noisy instances still has limited research. Limitations of isolated studies evaluation between CI and noisy instances are presented as follows:

(a) It will not be easy to find the concurrent impact of both challenges over defect prediction models; both these problems degrade the performance of SDP techniques.

(b) Dealing with noisy instances only helps in suggesting the percentage of noisy instances a model can tolerate. In contrast, studies about the CI problem only help in recommending the rate of imbalance in the dataset that a predictive model can digest.

(c) The common approach that can conquer both the challenges together cannot proposed.

(d) The trade-off ratio between the percentage of noisy instances and the CI rate can not be explored.

After our empirical study, we have listed of few compelling motivational queries about the requirement of combined analysis between noise and CI problem in software defect prediction and are shown below.

(i) Many of the software practitioners and researchers claimed over the quality of defect prediction datasets, and Shepperd et al. (2013) claimed many instances were noisy in NASA data repository. Joint analysis of CI problem and noise can help to study the relative impact between classifier and sampling methods over noisy instances; as limited studies were reported over the interaction of sampling and classifiers over noisy data.

(ii) How classifiers interact with sampling methods? Do certain sampling methods outperform when simultaneously used with specific classification algorithms over noisy datasets?

(iii) A combined study of sampling methods & classifiers and their performance analysis over various SDP models is still unexplored at different noise levels.

Researcher either suggested an SDP that dealt with CI problem or noisy instance but not both, but we proposed an SDP model that address both issues. Apart from this, we analyzed the tolerable noise capability of existing SDP models, i.e., after adding noise, the performance of the SDP model remains unchanged. For the meaningful treatment of the study approach, we framed five research query (RQ) based on evaluation metrics that may guide the attempt to proposition of a model. These RQ justify the observations of our empirical study.

List of research queries (RQs) is as follows.

RQ-1 What are the effects of noise on True positive rate (TPR) and False positive rate (FPR) over classical SDP models?

RQ-2 To what extent the suggested model is resistant over the various level of noise compared to the other classical SDP models?

RQ-3 What is the range of tolerable noise in baseline defect prediction models?

RQ-4 How does the class imbalance problem affect the performance of various SDP models over different noise levels?

RQ-5 Compare the performance of proposed approach with other classical SDP models without applying sampling method.

All six RQs can explore the circumstances under which classical SDP models work over noisy data. Noise tolerant ability in defect prediction still has a scope of research. We have conducted similar experiments for noise handling like (Kim et al. 2011) done in their article; apart from this, we also dealt with CI issues. Tantithamthavorn et al. (2015) also explore the challenges of mislabeled data, which leads to inconsistent results. Cabral et al. (2019) suggested an approach JIT-SDP that makes defect predictions at the software change level, and they presume that the characteristics of the problem remain constant over time. The article makes the following contributions.

(i) Combined empirical studies between noise and CI problem. The article also evaluates the impact of these two problems in the performances of baseline methods.

(ii) The article also analyzed the various tolerance level of noise and CI problems over baselines methods.

(iii) Suggested SDP model that can tolerate maximum noise degree and circumvent class imbalance issues. The suggested approach is mainly a change in a buggy prediction model. These two are the most prominent challenges in any SDP technique. We have tested the significance level of the suggested approach using TPR, FPR, F-measure, Precision, and ROC compared with other traditional SDP models.

(iv) We conducted 864 experiments over 3 public datasets using 5 classifiers and 1 sampling method. We also dispense a few guidelines for noise tolerance level and CI issue in baseline SDP models. Those guidelines will assist in better SDP models in the future.
In the next Sect. 2, we have discussed the related work, followed by the background details in Sect. 3; then after we illustrate the experimental procedure and suggested approach in Sect. 4. In last sections, we have analyzed and discuss results of our experiments in Sect. 5. Afterward, we talk about threats and validity in Sect. 6. In the final Sect. 7, we present conclusions drawn from the article.

2 Related work

A given software module consists of source code and other software metrics, and SDP classifies the module either clean or buggy. SDP classifies a module as either clean or buggy, whereas a given software module consists of software features, e.g., source code metrics; few existing SDP methods are SVM (Elish and Elish 2008; Hou and Li 2009), Naïve Bayes (Ji et al. 2019; Pandey et al. 2018), Random forest (Zhou et al. 2019; Khanh et al. 2018), AdaBoost (Hu et al. 2008; Peng et al. 2011), J48 (Queiroz et al. 2016; Catal et al. 2007), etc. Recently, a few other ensemble learning (Peng et al. 2011; Yang et al. 2017; Laradji et al. 2015) and deep learning-based (Wang et al. 2016; Li et al. 2017; Pandey and Tripathi 2020) defect prediction architecture have been reported. Wei et al. (2019) suggested an interesting approach using dimension reduction in different software metrics, and they have suggested an approach using tangent-based SVM. There are several unsupervised and semisupervised machine learning methods that are also applied in the SDP domain. Abaei et al. (2015) proposed a semi-supervised-based approach using hybrid self-organizing map (HySOM), the model has the ability to predict defect-prone module in an unsupervised manner. They performed experiments using NASA dataset and found improvement over existing methods. Catal (2014) performed a comparative analysis between performances of various semi-supervised methods. A semi-supervised method is proposed by Lu et al. (2012); they found the proposed model significantly better over the random forest. Metric-driven software quality prediction model was proposed by Catal et al. (2010); their method can be used when defects are absent; it does not require information about a number of clusters before the clustering phase. They found the proposed model significantly outperforms existing methods. Li et al. (2012) proposed an SDP model called ACoFores, which addresses the problem of inadequate availability of historical dataset. They used the PROMISE dataset for experiments and found optimal results compared to other state-of-the-art methods. Similar work has been performed by Seliya and Khoshgoftaar (2007); they investigated Expectation–Maximization (EM) algorithm for software quality prediction. They used NASA dataset and found EM-based prediction model improves generalization performance. SDP techniques are suffering from two main challenges; thus, we have categories related work into that fashion. The quality of the data in first categories, and second about the class imbalance issue.

2.1 Quality of defect dataset

In few studies (Malhotra 2015; Ramler and Himmelbauer 2013), researchers had claimed the existence of some errors in large datasets, like field error, the rate of field error is nearly around 5% (Maletic and Marcus 2000; Wu 1995). Zhu and Wu (2004) tried to handle noisy data and also tried to overcome from an over-fitting (Liu and Khoshgoftaar 2004) problem. Shepperd et al. (2013) also raised the question on the quality of data of NASA repository, but still, a few powerful machine learning-based defect prediction models are available such as (Pandey et al. 2018; Catal and Dirì 2007; Zheng 2010; Li et al. 2019). There are two different types of noises in a defect dataset, and both of these noises (Shanthini et al. 2019) affect the performance over machine learning algorithms, first is class noise and second is feature noise. However, we have only considered class noise in this article. Class noise is an interchange of the class label from clean to buggy or buggy to clean or both, due to any consequences. This problem leads to inconsistent results. Bachmann et al. (2010) concluded that few defects are not found in commit logs of a dataset, and hence, they are also not visible in automated linking tools. Bird et al. (2009) found that more accomplished developers are more likely to direct links between issues report to code change. Rahman and Devanbu (2013) investigated the influence of SDP models by inducing artificially generated defect dataset. Catal et al. (2011a) conducted a study over class and noise detection; they proposed a detection algorithm based on software feature threshold values. Riaz et al. (2018) proposed a two-stage data preprocessing methods that incorporates the feature selection and noise filter execution; they employed K-nearest neighbor and ensemble learning in their proposed approach. Alan and Catal (2011) proposed an outlier detection approach using metrics threshold and class label; they employed NASA datasets to identify class outliers; they found the proposed model outperforms over baseline methods.

2.2 Class imbalance problem

The class imbalance issue may produce biased result toward the negative instance (Ali et al. 2015). A comprehensive study about the class imbalance in SDP is recently done by Song et al. (2018). Few studies (Bennin et al. 2019; Chen et al. 2018) compared results produced from imbalance and balance class labels, but there are few researchers who proposed some solutions that have increased some accuracy of SDP
models. Researchers (Sharma et al. 2018; Limsettho et al. 2018; Tong et al. 2018) have proposed random subsampling (Clarkson and Shor 1989), SMOTE (Chawla et al. 2002), class balancer (Laurikkala 2001), and spread subsampling techniques, which help to avoid class imbalance issue and provide unbiased results. Joon et al. (2020) performed a combined study over class imbalance, feature selection, and simple noise removal strategy over public datasets; they used precision, recall, f-score, roc, and accuracy as performance measures.

3 Background

The general framework of a software defect prediction model is shown in Fig. 1. Software repository consists of two segments (Fan et al. 2020); version control system (VCS), and issue tracking system (ITS) as shown in Fig. 1. Most of the time software practitioners use both of them because version control systems (source code repository system) are unable to store bugs. As figure reports the instance are generated from software repository. These instance are made up of software metrics, the data cleaning and other preprocessing are required to build the training set. Then, training set is fed into the trained/untrained model that can classify buggy or clean module. We will provide detailed discussion about preprocessing and trained model in the later sections. Before designing any prediction model, we need to create the prediction target, i.e., class label. Software modules consist of software entities such as file (Moser et al. 2008), component (Nagappan et al. 2006) or change (Kim et al. 2006); SDP model is intended to predict software module as either buggy/defective or clean/non-defective.

There are two types of defect prediction models, buggy file prediction and change buggy prediction. The detail description is given below.

3.1 Buggy files prediction

Identification of the buggy files in advance helps the development team leader to properly and optimally allocated resources, and it leads to minimizing the testing effort. As we know, some of the internal properties of a software system, such as software metrics, and have associates with the external property such a fault-proneness of a module. This kind of SDP model mainly identifies software features that are expressed in a defect dataset. This classification model learns from historical data and predicts the fault-prone modules in a test data. A lot of software features are responsible for this kind of SDP models such as resource metrics (Ram et al. 2010), process metrics (Kan 2002) and cyclomatic complexity metrics (Zhang et al. 2007).

3.2 Change buggy prediction

When some new changes are introduced in a software modules, the change buggy prediction predicts whether the changed software module is buggy or not, and it learns from change classification (CC). Let us say a module consist n files and suppose a new file is added to it, so there are total n + 1 files are present in the module. Now, this n + 1 files to the module may cause the software faulty. It mainly involves two source code revisions, an old revision, and a new revision. This change in several files is related to metadata, which includes author, change-log, date of commit, etc. After mining the change history, it can derive the co-change count, which indicates, for how many files changes, the system will remain clean or buggy. Kim et al. (2008) illustrated this process in their article.

To build any of the two types of SDP models as defined above requires class labels (buggy/clean) and various features. Model fitting using mislabeled data may cause incorrect results. In this direction, we proposed a change buggy
prediction model by using public data. In the next section, we will discuss its experimental details, performance measure, and build a useful SDP model which can tolerate noisy instance up to some extents.

4 Experimental procedure and suggested approach

In this section, we will illustrate the experimental details, dataset description, noise addition phenomenon, performance metrics, preprocessing, classification techniques, and the suggested approach.

4.1 Experimental setup

This section will discuss every aspect required in experiments. We have performed 864 experiments over three public datasets that are Scarab, Columba, and Eclipse. We have implemented our experiment on 8 GB RAM, 1 TB of the hard drive over windows 10 operating system. Python libraries that have been applied for experiments are Numpy, Scipy, Scikit-learn, Keras, Pandas, and Matplotlib. Validation of all 864 experiments is conducted over WEKA (Waikato Environment for Knowledge Analysis) tool (Witten and Frank 2000) to reverify experimental results.

4.1.1 Dataset description

The three public datasets are Columba, Scarab, and Eclipse that we have used in our experiments for the buggy change prediction model, as detailed shown in Table 1. These are classical datasets and have substantial training instances, as shown in Table 1, compared with other open-source datasets, which lead to satisfactory results. Although (Kim et al. 2011) uses these datasets in their experiments, they also conducted similar experiments by adding noise manually into the datasets, so we are extending their experiments.

4.1.2 Noise added in dataset

We assumed datasets that we are using are pure, i.e., there are no noisy instances. Kim et al. (2011) also considered similar assumptions over these three datasets. So we injected some percentage of noisy labels into it and then exercised its training using various SDP models. Now, with the interchange of the class label from buggy to clean and clean to buggy that introduced class noise in the defect dataset. Then, we evaluated the performance of various SDP models over different noise levels. To analyze the performance of various classical SDP models, we have added noise in the defect data from 0 to 80%. 0% means no noise added, whereas 10% means, 10% of the total instance has been selected and interchange of the target label.

4.2 Performance measure

To evaluate the performance of five SDP models, we used five performance metrics; those are True Positive Rate (TPR), False Positive Rate (FPR), Precision, F-measure, and area under ROC (receiver operating characteristic) curve. The brief details of these evaluation metrics are given below.

(a) True Positive Rate (TPR) It is the proportion of actual positive instances that are correctly classified; It is also known as Recall. More the TPR better will be the model.

\[
TPR = \frac{TP}{TP + FN}
\]

(b) False Positive Rate (FPR) It is the ratio between the number of negative instances wrongly classified as a positive instance (false positives) to the total number of actual negative instances (regardless of classification). Low the
FPR more effective the model is and vice-versa.

\[ FPR = \frac{FP}{FP + TN} \]

(c) **Precision** It is the ratio of relevant instances out of total instances. The relevant instances are those instances that are required for the classification.

\[ Precision = \frac{FP}{FP + TN} \]

(d) **F-measure** It is the harmonic mean of precision and recall (TPR). Its value lies between 0 to 1, 0 implies the worst result and 1 implies the best result. It is also known as f-score and F₁ score.

\[ F\text{-}measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \]

(e) **Receiver Operating Characteristics (ROC) curve** It is an area under the curve of TPR and FPR. Its value lies between 0 and 1. The value 0 shows that there is no correct classification, 0.5 shows random classification and 1 for 100% correctly classified instances. It is mainly for diagnostic ability of a binary classifier (Davis and Goadrich 2006; Shatnawi 2017; Catal et al. 2011a).

4.3 **SDP models for experiments**

We added noise levels from 0 to 80% and then trained various models. We analyze the performance of defect prediction models over different noise levels. The other major challenge in any defect datasets is a skewed distribution of a particular class causes class imbalance problem. To deal with such a problem, few researchers applied sampling methods that establish the balance between positive and negative classes. In the next sections, we explain the preprocessing methods and then various baseline models.

4.3.1 **Preprocessing techniques**

In the preprocessing stage, after data cleaning, the feature selection and sampling methods are the major steps. The steps involved in preprocessing are shown below.

1. **Sampling technique** Table 1 reports the datasets have skewed distribution and suffer from class imbalance challenge. To avoid CI problem, we have applied random undersampling methods (Kotsiantis and Pintelas 2003). We have also tried other sampling methods, e.g., class balancer (Chawla 2009), synthetic minority oversampling technique (Chawla et al. 2002), and spread subsample technique (Davies 1995), but achieved optimal results over the random sampling technique. We considered the algorithm from Charte et al. (2015), which deletes random samples of the majority class label (SetLabels). The full description of the algorithm is given in Charte et al. (2015). The main objective of this algorithm is to achieve uniform distribution of buggy and non-buggy class labels.

2. **Feature selection** As Table 1 reports, each dataset has a massive number of features. So we need to select relevant features for better analysis. We have used Information gain (Lee and Lee 2006) as a feature selection method and Ranker method as search technique (Chandrashekar and Sahin 2014; Roobaert et al. 2006) for feature ranking. It is mainly entropy-based method; it is defined as a amount of information provided by the selected item for categorization. It is calculated by how much an item’s information is important for classification, in order to compute the importance of lexical items for the classification problem. The ranker method uses conjunction with attribute evaluator (Entropy, Gain ratio, etc.). It has three parameters, \( P \), \( T \), and \( N \). The \( P \) (start state) specify the starting set of the attribute; specified attribute is ignored during ranking. \( T \) (threshold), the threshold is specified by which features are ignored. \( N \) (Number of selection); it specified the number of attributes selected.

4.3.2 **Classification techniques**

We have used five different classification techniques while applying the same preprocessing methods (discussed in Sect. 4.3.1). These classifiers build five different SDP models. The list of classification techniques is shown below.

(i) **Naive Bayes (NB)** It is a probabilistic classifier (Rish et al. 2001; Huang and Li 2011), which is derived from Bayes theorem. It is a family of algorithm which shares a mutual principle. Every pair of features is classified as independent of each other. The underlying assumptions are features makes an equal and independent contribution to the outcomes (Catal et al. 2011b). We have used random batch size = 100, set “doNotcheckCapabilities” and “kernelEstimator” as “False”. 

(ii) **Least Square Support Vector Machine (LSVM)** It is a supervised learning algorithm (Suykens and Vandewalle 1999). It can be used for both classification and regression problems. It is a binary classifier which creates \( n \)-dimensional hyperplane to classify the instances (Elish and Elish 2008). We have used radial basis function kernel in our experiments. We used random size = 100, catch size = 40, cost = 1, degree = 3, loss = 0.1, nu = 0.5, and seed = 1.

(iii) **J48** It is variation (Bhargava et al. 2013) of the C4.5 algorithm; it is a decision tree-based classification algorithm which used to create Univariate Decision Trees.
(UDT) (Catal and Diri 2007). The leaf node will decide the instance belongs to which category; it mainly calculate the information gain of each attribute and select the attribute with max info gain. We have used batch size = 100, we set “binarySplits” as “False”, “collapseTree” as “True”, no. of folds = 3, seed = 1, “unprunned” = “False”, and “useLaplase” = “False”.

(iv) AdaBoost It is short of Adaptive Boosting (Zheng 2010; Hu et al. 2008), which is mainly an ensemble learning technique. It combines different weak learners into one model and combines the results of each weak learner. That makes the classifier more powerful. As it is an ensemble learning technique; it overcomes the over-fitting problem. We have utilized weak classifier as “Decision stump”, “weightThresold” = 100, seed = 1, and “doNotcheckCapabilities” as “False”.

(v) Random Forest (RF) We have applied RF as a classifier in our proposed model. It is also an ensemble learning technique (Liaw and Wiener 2002). Algorithm 1 shows the pseudocode of RF learning algorithm. We considered this algorithm from Liaw and Wiener (2002), complete discussion about RF can be found in Liaw and Wiener (2002). There is a function in RF algorithm called “RandomizedTreeLearn” which mainly returns the learned tree. It is a decision tree-based learning algorithm; it is one of the most robust SDP model (Kaur and Malhotra2008; Catal and Diri 2009). We have used batch size = 100, “breakTiesRandomly” as “True”, no. of folds = 3, seed = 1, “unprunned” = “False”, “ComputerAttributeImportance” as “False”, “no. of slots” = 1, and seed = 1.

**Algorithm 1:** Pseudo code of Random forest.

```
Initial condition: Training set T = (x_1, y_1),(x_2, y_2),...,(x_n, y_n), let number of features be F and number of tree be B.
function RandomForest(T, F)
    h ← ∅ for i = 1 to B do
        T' ← Boot strap sample from T;
        h_i ← RandomizedTreeLearn(S, F);
        h = h u h_i
    return h;
function RandomizedTreeLearn(T, F);
    at every node
    f ← is small subset of F;
    Split of best best feature in f;
    return Learned Tree
end function
```

Before applying the learning technique, we split the dataset into a training set and testing set. Where 70% for the training data and 30% for testing data, we have also performed other split ratios but got optimal results on a 70–30% ratio. Then, we have used tenfold cross-validation (Dor and

**Algorithm 2:** Experimental procedure.

```
p ← 20 /* No. of times each experiment performs */
q ← 10 /* No. of fold in cross validation of dataset */
d_i ← d_1, d_2, & d_3 /* No. of datasets used for experiment */
c_j ← c_1, c_2, c_3, c_4, & c_5 /* No. of classifier used for experiment */

foreach p times experiment do
    /* dataset selection */
    foreach data d_i do
        SelData ← Select n instance from each data d_i
    for instance i do
        buggy ← non-buggy & non-buggy ← buggy
        /* changing class label */
        // Preprocessing
    foreach instance i do
        f_1 ← Random sampling /* Sampling technique used for experiments */
        f_2 ← Info gain /* Feature selection */
    foreach a fold do
        TestData ← SelData[fold]
        TrainData ← d_i - TestData
        end
        // classifier selection
        foreach classifier c_j do
            TrainData ← attribute(TrainData)
            /* attribute selection */
            Classifer = c_j(TrainData) /* classifier
            operate on training data */
        end
        Performance P_1 ← c_j[TestData] /* PRC, MCC, ROC and F-measure */
    end
    // classifier applied without adding noise
    foreach d_i do
        SelData ← Select m instance of every data d_i
    foreach instance i do
        apply f_1 & f_2
    end
    /* preprocessing */
    foreach c_j do
        foreach data d_i do
            SelData ← n instance from data d_i
        foreach classifier c_j do
            TrainData ← attribute(TrainData)
            Classifer = c_j(TrainData)
        end
        Performance P_2 ← c_j[TestData] /* TPR, FPR, Presision, F-measure, & ROC */
    end
    Compare P_1 and P_2 /* compare performance of models with and without adding noise */
```
Zhou 2007) on training set in each classifier. It avoids the possibility of an over-fitting problem (Moore 2001) in the classification model.

4.4 Suggested approach

We have given the sequence of procedures regarding experiments in Algorithm 2. The Underlying architecture of the suggested approach is shown in Fig. 2; it reflects each phase of the suggested model. Noise is added using the mislabeling of the class label, as shown in Fig. 2. We have injected various noise level in a dataset, and we have tested the endure noise level in change buggy prediction model. We have applied information gain as an attribute selection method and ranking method as a search method to rank the attribute and select the most relevant attribute (see Sect. 4.3.1).

We have utilized random undersampling as a sampling technique to address the class imbalance problem. We have also tried a few other (SMOTE, Class Balancer, Spread Sub-Sampling, etc.) well known sampling techniques, but the best results came from random undersampling technique.

After preprocessing, we have split the dataset into training set and testing set, 70% for training, and 30% for testing (see Fig. 2). After that, we have applied the tenfold cross-validation technique on training data. Cross-validation Dor and Zhou (2007) also avoids the over-fitting Moore (2001) and makes the better prediction model. To avoid random bias, each experiment has been performed ten times and taken the mean value of each performance measure. The basic architectural view of the suggested approach shown in Fig. 2. Random Forest is applied as a classifier, as shown in Fig. 2. We have performed similar preprocessing step for the pure set, i.e., 0% noise and compared the performance $P_1$ and $P_2$, as shown in the Algorithm 2, here $P_1$ performance at 10–80% noise level and $P_2$ at 0% noise level.

In the next section, we will discuss the performance of various SDP models after adding different noise levels from 0 to 80%. Besides, we also test the tolerable noise in the suggested architecture without applying the sampling technique for all five baseline methods.

5 Results and analysis

We have experimented with, and without noisy instances in datasets, and the noise has been added from 10 to 80% in all three datasets. We have also conducted experiments with and without applying the sampling method at various noise levels and evaluating the performances. In this section, we will address all five research queries (see Sect. 1) and also justify the conclusion corresponding to experimental results. Justifications of every RQs are shown below.

5.1 What are the effects of noise on True positive rate (TPR) and False positive rate (FPR) over classical SDP models?

Table 2 and Fig. 3 represents the TPR of all baseline models over the various noise levels, whereas Table 3 and Fig. 4 report the FPR of various SDP models over different noise levels. On Eclipse data, the RF-based model exceeds its performance over other defect prediction models. On pure data, the TPR value is 0.975, whereas the lowest TPR value produced by the proposed model is 0.844 at 60 and 80%. The worst performance processed by NB-based model over pure Eclipse data is 0.854. The AdaBoost was the most variant model when the noise level increased from 0 to 80%, and its TPR values started decreasing from 0.9 to 0.489.

As we can see in Table 2 and Fig. 3 the TPR of all baseline models over the various noise levels. Whereas Table 3 and Fig. 4 report the FPR of various SDP models over different noise levels. On Eclipse data, the RF-based model exceeds its performance over other defect prediction models. On pure data, the TPR value is 0.975, whereas the lowest TPR value produced by the proposed model is 0.844 at 60 and 80%. The worst performance processed by NB-based model over pure Eclipse data is 0.854. The AdaBoost was the most variant model when the noise level increased from 0 to 80%, and its TPR values started decreasing from 0.9 to 0.489.

FPR value of all five defect prediction models is shown in Table 3. The lowermost FPR value for pure Columba data (see Fig. 4b) is 0.209, and it produced by the RF-based model. SVM-based technique has the highest FPR value for pure data, which is 0.583. The FPR value of NB-based model increases up to 0.377 at 40% noise, then started decreasing. Similarly, for the J48 defect prediction model, it starts increasing to 0.212 at 30% noise level then started dropping. The suggested model is most resistant, as FPR values decrease when noise level increases. AdaBoost has the worst resistance because it starts decreasing up to 10% noise then increasing.

SDP models for the Scarab dataset are more robust, as shown in Table 3. The RF-based defect prediction model has lowest FPR value (see Fig. 4c), i.e., 0.11 at 0% noise. The worst performance processed by NB-based technique at the same noise degree with 0.273 FPR value. The suggested model is most tolerable as its value is close to 0.11–40% noise. Even the J48-based model also has high noise tolerant ability with 0.153 FPR value at a 40% noise degree, which is close to 0.132 at 0%.
Fig. 2 Underlying architecture of Random forest (suggested) based SDP model
Table 2: True positive rate values of all datasets at different noise levels of various defect prediction models

| Dataset | Method | 0%       | 10%      | 20%      | 30%      | 40%      | 50%      | 60%      | 70%      | 80%      |
|---------|--------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Columba | NB     | 0.677/0.635 | 0.613/0.589 | 0.669/0.608 | 0.656/0.696 | 0.660/0.623 | 0.666/0.631 | 0.672/0.588 | 0.649/0.594 | 0.653/0.618 |
|         | SVM    | 0.744/0.706 | 0.688/0.661 | 0.677/0.626 | 0.675/0.614 | 0.669/0.619 | 0.640/0.571 | 0.635/0.582 | 0.678/0.611 | 0.536/0.639 |
|         | J48    | 0.851/0.711 | 0.847/0.661 | 0.827/0.699 | 0.805/0.636 | 0.824/0.653 | 0.821/0.626 | 0.835/0.659 | 0.823/0.667 | 0.830/0.698 |
|         | RF     | 0.893/0.748 | 0.876/0.712 | 0.874/0.699 | 0.851/0.696 | 0.874/0.680 | 0.848/0.695 | 0.855/0.682 | 0.872/0.698 | 0.877/0.719 |
|         | AdaBoost | 0.731/0.724 | 0.691/0.685 | 0.646/0.643 | 0.608/0.606 | 0.659/0.641 | 0.626/0.595 | 0.591/0.604 | 0.643/0.623 | 0.646/0.671 |
| Eclipse | NB     | 0.854/0.970 | 0.815/0.804 | 0.709/0.636 | 0.693/0.584 | 0.612/0.563 | 0.636/0.520 | 0.619/0.508 | 0.649/0.520 | 0.619/0.508 |
|         | SVM    | 0.910/0.940 | 0.860/0.838 | 0.795/0.760 | 0.742/0.674 | 0.669/0.586 | 0.618/0.486 | 0.643/0.534 | 0.678/0.549 | 0.643/0.534 |
|         | J48    | 0.930/0.955 | 0.910/0.857 | 0.889/0.798 | 0.874/0.716 | 0.843/0.671 | 0.813/0.662 | 0.821/0.680 | 0.859/0.730 | 0.829/0.682 |
|         | RF     | 0.975/0.940 | 0.944/0.856 | 0.923/0.710 | 0.873/0.675 | 0.863/0.675 | 0.847/0.624 | 0.844/0.616 | 0.871/0.653 | 0.844/0.616 |
|         | AdaBoost | 0.90/0.955  | 0.874/0.860 | 0.789/0.766 | 0.662/0.669 | 0.584/0.566 | 0.510/0.496 | 0.489/0.680 | 0.612/0.590 | 0.489/0.498 |
| Scarab  | NB     | 0.725/0.715 | 0.732/0.704 | 0.710/0.682 | 0.724/0.700 | 0.688/0.693 | 0.659/0.623 | 0.616/0.541 | 0.70/0.608 | 0.641/0.606 |
|         | SVM    | 0.779/0.724 | 0.852/0.721 | 0.833/0.692 | 0.80/0.688 | 0.757/0.678 | 0.722/0.675 | 0.747/0.590 | 0.714/0.630 | 0.762/0.666 |
|         | J48    | 0.867/0.724 | 0.855/0.740 | 0.841/0.689 | 0.858/0.711 | 0.852/0.674 | 0.841/0.713 | 0.826/0.638 | 0.831/0.706 | 0.862/0.670 |
|         | RF     | 0.890/0.783 | 0.890/0.768 | 0.883/0.747 | 0.895/0.767 | 0.885/0.743 | 0.885/0.727 | 0.87/0.681 | 0.878/0.699 | 0.883/0.695 |
|         | AdaBoost | 0.783/0.756 | 0.728/0.713 | 0.715/0.707 | 0.779/0.710 | 0.680/0.700 | 0.691/0.682 | 0.653/0.619 | 0.689/0.646 | 0.660/0.651 |

Fig. 3: True positive rate all three datasets
|                | 0%           | 10%          | 20%          | 30%          | 40%          | 50%          | 60%          | 70%          | 80%          |
|----------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| **Columba**    |              |              |              |              |              |              |              |              |              |
| NB             | 0.355/0.355  | 0.345/0.371  | 0.347/0.395  | 0.353/0.367  | 0.377/0.408  | 0.355/0.384  | 0.317/0.389  | 0.318/0.365  | 0.347/0.363  |
| SVM            | 0.583/0.656  | 0.611/0.646  | 0.477/0.587  | 0.416/0.498  | 0.429/0.498  | 0.384/0.461  | 0.383/0.444  | 0.443/0.548  | 0.536/0.605  |
| J48            | 0.216/0.428  | 0.207/0.418  | 0.192/0.411  | 0.212/0.387  | 0.187/0.390  | 0.180/0.349  | 0.168/0.345  | 0.189/0.378  | 0.193/0.373  |
| RF             | 0.209/0.500  | 0.206/0.472  | 0.170/0.411  | 0.180/0.367  | 0.150/0.381  | 0.157/0.317  | 0.146/0.322  | 0.154/0.378  | 0.185/0.418  |
| AdaBoost       | 0.488/0.537  | 0.487/0.538  | 0.549/0.479  | 0.436/0.494  | 0.466/0.482  | 0.412/0.444  | 0.402/0.439  | 0.412/0.489  | 0.554/0.547  |
| **Eclipse**    |              |              |              |              |              |              |              |              |              |
| NB             | 0.228/0.470  | 0.232/0.287  | 0.231/0.295  | 0.220/0.343  | 0.284/0.278  | 0.233/0.306  | 0.257/0.323  | 0.245/0.311  | 0.257/0.323  |
| SVM            | 0.778/0.840  | 0.683/0.773  | 0.598/0.689  | 0.452/0.582  | 0.395/0.497  | 0.312/0.442  | 0.294/0.348  | 0.297/0.414  | 0.294/0.384  |
| J48            | 0.458/0.471  | 0.282/0.480  | 0.228/0.400  | 0.164/0.285  | 0.143/0.248  | 0.144/0.257  | 0.133/0.238  | 0.111/0.227  | 0.133/0.238  |
| RF             | 0.225/0.321  | 0.261/0.596  | 0.191/0.422  | 0.186/0.318  | 0.137/0.318  | 0.119/0.296  | 0.126/0.298  | 0.106/0.304  | 0.126/0.298  |
| AdaBoost       | 0.792/0.471  | 0.5/0.550    | 0.524/0.617  | 0.636/0.482  | 0.464/0.527  | 0.404/0.405  | 0.403/0.238  | 0.417/0.433  | 0.403/0.401  |
| **Scarab**     |              |              |              |              |              |              |              |              |              |
| NB             | 0.273/0.282  | 0.272/0.30   | 0.315/0.347  | 0.296/0.347  | 0.346/0.339  | 0.352/0.395  | 0.390/0.451  | 0.307/0.388  | 0.354/0.389  |
| SVM            | 0.221/0.277  | 0.150/0.280  | 0.176/0.395  | 0.208/0.336  | 0.256/0.357  | 0.3/0.443    | 0.29/0.496   | 0.294/0.379  | 0.238/0.337  |
| J48            | 0.132/0.227  | 0.145/0.260  | 0.164/0.315  | 0.151/0.302  | 0.153/0.333  | 0.188/0.326  | 0.185/0.389  | 0.168/0.295  | 0.138/0.330  |
| RF             | 0.111/0.217  | 0.111/0.233  | 0.124/0.269  | 0.109/0.255  | 0.121/0.287  | 0.147/0.325  | 0.140/0.381  | 0.123/0.306  | 0.118/0.308  |
| AdaBoost       | 0.217/0.245  | 0.272/0.288  | 0.309/0.319  | 0.245/0.337  | 0.362/0.359  | 0.47/0.463   | 0.433/0.501  | 0.326/0.345  | 0.342/0.353  |
To what extent the suggested model is resistant over the various level of noise compared to the other classical SDP models?

TPR, FPR, and Precision values of the defect prediction model are directly co-related with noise resistance. Figure 3b reports RF-based SDP models have least deviated, i.e., the TPR value at 0% noise is 0.893, and 80% is 0.877, which are close to each other. Even for the other models, TPR values fluctuated. We can see in Fig. 3a that RF-based model has TPR range from 0.975 (0% noise) to 0.844 (80% noise). Although till 20% of noise level, its value is 0.923, which is close to 0.975. The proposed model over Scarab data has the highest tolerable capability, as shown in Fig. 3c and Table 2, at 0 and 70% the TPR value is 0.890, and 0.878, respectively, approximate equal value. Whereas no other methods have that much ability to tolerate this amount of noise, and they showed inconsistent results. The FPR and precision are productive metrics to measure the efficiency of SDP models. Table 3 reports that the RF-based defect prediction model has the least FPR value over all three datasets. FPR values for pure Eclipse, Columba, and Scarab datasets are 0.225, 0.209, and 0.111, respectively. Even the deviation curve of FPR values at various noise levels is shown in Fig. 4. The FPR value for Columba data at 30% noise is 0.180, which is close to 0.209 at 0% noise; it implies model can tolerate noise up to 30%, as shown in Fig. 4b. The FPR value of Eclipse and Scarab data processed by the proposed model at 0% noise is 0.458 and 0.111, respectively, as shown in Table 3. The FPR curve deviation for Eclipse data produced by the RF-based model has last deviation, as shown in Fig. 4a. Whereas the FPR curve deviation of Scarab data is shown in Fig. 4c, we can see that the RF-based model has the least deviation, whereas NB-based model has the most deviated curve. FPR values for Scarab data produced by the proposed model at 0 and 40% are 0.11, and 0.121, respectively, which is close to each other; it implies tolerable deviation till 40% noise.

The precision values of classical SDP models over various noise levels for all three datasets are shown in Table 5. We can see in Fig. 5b, the precision of RF-based model at 0 and 40% noise is 0.894, and 0.875, respectively. These two values are close to each other, which implies the model tolerates noise.
Fig. 5  Precision of all three datasets

up to 40%. Whereas precision values pure Eclipse and Scarab data produced by the suggested model are 0.970 and 0.890, respectively. The precision value of Eclipse at 20% noise processed by the RF-based model is 0.923, which is close to 0.970 at 0%; it indicates, the model remained unchanged until 20% noise. The precision value of Scarab data at 80% noise is 0.883, which is approximately equal to 0.890. Precision values deviation for Eclipse and Scarab dataset can be better viewed in curves Fig. 5a, c, respectively. The J48-based model also has high noise tolerant rate for Scarab data, and the precision value is 0.854 at 50% noise level, which is close to 0.870–0% noise.

5.3 What is the range of tolerable noise in baseline defect prediction models?

After performing experiments, we can conclude that for all five SDP models, the range of tolerable noise is different. Figures from Figs. 3, 4, 5, 6 and 7 show the TPR, FPR, Precision, F-score, and ROC, respectively, of various defect prediction models under different noise condition. We have analyzed each classifier. The tolerable noise range in NB-based model is from 20 to 30% because TPR values (see Table 2 and Fig. 3) and FPR (Table 3 and Fig. 4) values remain unchanged at 30% noise level; it indicates model is stable and tolerable up to 30% noise. Even precision (Fig. 5) and ROC (Fig. 7) of NB-based SDP for every dataset are fallen after adding noise more than 30%. F-measure (Fig. 6) and TPR (Fig. 3) values continuously fall down, but up to 30% of noise level, TPR, and f-score values are approximately equal, indicates performance breakdown point at 30% noise. FPR values (Fig. 4) are increased when the noise level rises. Still, from 0 to 30%, FPR values are close to each other; it indicates the models are uniformly performed up to 30% noise, but after that model becomes misclassifying the actual class; which leads to degrade in the performance.

SVM is also an effective SDP model, Figs. 5, and 7 show the precision, and ROC values, respectively. We can easily see the effectiveness of SVM over every datasets and deviation of SVM over different noise conditions. We can see that precision is fallen over Eclipse and Columbia datasets after adding different noise level, but for the Scarab data,
the precision first gradually increases than started decreasing. Whereas ROC rises in the early phase for both Eclipse and Scarab datasets. ROC values produced by the SVM defect prediction model over Eclipse dataset increase up to 70% noise and then started decreasing indicates SVM-based model is highly noise tolerable over Eclipse data. For the Scarab data, the SVM-based model degrades its ROC values after a 10% noise level, as shown in Fig. 7. It stipulates the SVM model is not stable over the Scarab dataset. Whereas for Columbia data, the 20% noise is significant, which means there are no hard changes in ROC values. SVM-based method is efficient to tolerate noise up to 40% for Eclipse data. The TPR values are continuous, falling down for all 3 datasets, as shown in Fig. 3, whereas f-score values are changing stochastically for Columbia and Scarab datasets, and started reducing at every noise level for Eclipse data as reported in Fig. 6. FPR values (Fig. 4) for Eclipse and Columbia datasets gradually decrease when noise increases, but for Scarab data; it increases till 70%, and then, there is a sudden decrease. It indicates the SVM is stable and more tolerable over Scarab data.

The J48 algorithm uniformly performs when the noise level is between 30–40%, because TPR, F-score, ROC, and precision values are approximately equal for all the three datasets. The FPR is suddenly started decreasing when noise is more added in the Eclipse data, as we can see in Fig. 4c; it indicates that J48-based model is inefficient to tolerate noise in Eclipse data after 30% noise. Whereas FPR values increase when the noise level increases, which makes results unpredictable.

AdaBoost-based SDP model is performing least effective over each data when noise increases as shown in Figs. 6 and 7. When the noise level is increased to 10%, the TPR and precision started decreasing for all three datasets. Even FPR values decrease for the Eclipse data as shown in Fig. 4. Although in Columbia, and Scarab datasets, FPR values increase when the noise level increases; it implies the Adaboost is inefficient to tolerate noise over these datasets; so Adaboost can tolerate maximum noise up to 30%.

In our suggested approach, i.e., RF-based SDP model, the ROC, and precision were almost unchanged till 60–70% as shown in Figs. 7 and 5. F-score and TPR values started
decreasing as the noise level increases, but from 40 to 60% of noise, the TPR and f-score are unaffected. The FPR in Fig. 4a started decreases when noise increases, whereas in Fig. 4b, c the FPR increases as per noise increases. For all three datasets, the noise tolerates capacity by the proposed approach is 30–40%.

Table 4 reports the f-score of all methodologies at various noise degree. The maximum f-score processed by the proposed model for pure Columbia, Eclipse, and Scarab datasets is 0.889, 0.698, and 0.889, respectively. Figure 6 shows the deviation curve about f-score from various SDP models over different noise degree.

We can see in Fig. 6b that RF and SVM-based SDP have the least, and most deviated curve, respectively. In all noise level scenarios, the RF-based SDP has approximately equal f-score, their respective f-score values at 10, 20, 30, 40, 50, 60, 70, and 80%, noise levels are 0.871, 0.872, 0.849, 0.873, 0.847, 0.855, 0.871, and 0.874, respectively. The f-score is still more than 0.85 after 40% of noise; this can be because of the sample space of buggy and clean instance are approx equal.

f-score values of Eclipse data on various noise states are shown in Fig. 6a. We can see in Table 4 and Fig. 6c, the highest f-score value for pure Eclipse data processed by NB-based SDP model, is 0.974, followed by RF with 0.968 f-score value. The lowermost f-score of Eclipse data is 0.870 which is produced by AdaBoost-based SDP model. The least deviated curve is of proposed model and J48-based SDP models, whereas the most deviated curve is of NB-based SDP technique, as shown in Fig. 6a. The f-score value of RF-based SDP method at 10, 20, 30, 40, 50, 60, 70 and 80% noise levels is 0.940, 0.919, 0.919, 0.869, 0.861, 0.841, 0.844, 0.847, and 0.844, respectively. The f-score is still more than 0.8 after 50% of noise. When the noise increases, the sample of actual buggy/clean instances degrades. Hence, the sample space of both classes becomes approximately equal, so the model leads to a good fitted model and outperforms at the high noise level.

In Table 4, we can see the f-score of Scarab data by various methodologies under different noise conditions. The maximum f-score value is 0.889 for the Scarab data, and the RF-based SDP model produces it. After that J48-based model with 0.867 f-score value. Figure 6c shows the devia-
| Model  | F-measure values of all datasets at different noise levels of various defect prediction models |
|--------|---------------------------------------------------------------------------------------------|
|        | 0%          | 10%         | 20%         | 30%          | 40%         | 50%          | 60%          | 70%          | 80%          |
|        | Columba     | SVM         | J48         | RF           | AdaBoost    | Eclipse      | RF           | AdaBoost    | Scarab       |
|        |             |             |             |              |             |             |              |             |              |
| NB     | 0.688/0.651 | 0.624/0.6   | 0.672/0.614 | 0.657/0.621  | 0.657/0.626 | 0.671/0.578  | 0.652/0.596  | 0.660/0.626 |
| SVM    | 0.682/0.623 | 0.6/0.554   | 0.620/0.527 | 0.649/0.568  | 0.636/0.565 | 0.627/0.544  | 0.628/0.565  | 0.644/0.540  |
| J48    | 0.851/0.707 | 0.846/0.661 | 0.827/0.681 | 0.805/0.636  | 0.824/0.628 | 0.821/0.653  | 0.835/0.659  | 0.823/0.664  | 0.830/0.691 |
| RF     | 0.889/0.718 | 0.871/0.684 | 0.872/0.681 | 0.849/0.684  | 0.873/0.669 | 0.847/0.692  | 0.855/0.682  | 0.871/0.686  | 0.874/0.699 |
| AdaBoost | 0.709/0.696 | 0.666/0.640 | 0.570/0.618 | 0.603/0.570  | 0.600/0.583 | 0.591/0.556  | 0.591/0.558  | 0.638/0.593  | 0.599/0.609 |
| Eclipse |             |             |             |              |             |             |              |             |              |
| NB     | 0.974/0.965 | 0.832/0.821 | 0.730/0.666 | 0.707/0.60   | 0.619/0.579 | 0.642/0.530  | 0.621/0.512  | 0.652/0.530  | 0.621/0.512 |
| SVM    | 0.879/0.752 | 0.825/0.789 | 0.746/0.684 | 0.697/0.587  | 0.634/0.509 | 0.613/0.434  | 0.635/0.509  | 0.668/0.512  | 0.635/0.509 |
| J48    | 0.924/0.952 | 0.909/0.850 | 0.886/0.786 | 0.822/0.713  | 0.841/0.678 | 0.813/0.658  | 0.821/0.678  | 0.858/0.726  | 0.821/0.678 |
| RF     | 0.968/0.921 | 0.940/0.843 | 0.919/0.680 | 0.869/0.658  | 0.861/0.658 | 0.840/0.614  | 0.844/0.606  | 0.870/0.637  | 0.844/0.606 |
| AdaBoost | 0.870/0.952 | 0.862/0.845 | 0.799/0.745 | 0.666/0.593  | 0.492/0.522 | 0.428/0.422  | 0.370/0.678  | 0.611/0.588  | 0.370/0.397 |
| Scarab |             |             |             |              |             |             |              |             |              |
| NB     | 0.723/0.711 | 0.727/0.698 | 0.704/0.690 | 0.721/0.673  | 0.682/0.688 | 0.663/0.627  | 0.618/0.554  | 0.697/0.607  | 0.634/0.599 |
| SVM    | 0.779/0.724 | 0.852/0.721 | 0.832/0.692 | 0.880/0.648  | 0.756/0.672 | 0.765/0.653  | 0.742/0.554  | 0.711/0.624  | 0.762/0.664 |
| J48    | 0.867/0.724 | 0.855/0.740 | 0.841/0.689 | 0.857/0.710  | 0.852/0.675 | 0.840/0.712  | 0.826/0.636  | 0.832/0.706  | 0.862/0.670 |
| RF     | 0.889/0.783 | 0.889/0.768 | 0.882/0.745 | 0.895/0.761  | 0.885/0.739 | 0.884/0.720  | 0.870/0.675  | 0.878/0.698  | 0.883/0.693 |
| AdaBoost | 0.783/0.755 | 0.728/0.713 | 0.709/0.700 | 0.776/0.696  | 0.670/0.681 | 0.646/0.646  | 0.621/0.548  | 0.677/0.656  | 0.658/0.646 |
Table 5  Precision values of all datasets at different noise levels of various defect prediction models

|                  | 0%     | 10%    | 20%    | 30%    | 40%    | 50%    | 60%    | 70%    | 80%    |
|------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Columba          |        |        |        |        |        |        |        |        |        |
| NB               | 0.713/0.698 | 0.676/0.654 | 0.679/0.629 | 0.661/0.693 | 0.656/0.620 | 0.674/0.631 | 0.684/0.615 | 0.684/0.638 | 0.680/0.658 |
| SVM              | 0.752/0.654 | 0.698/0.590 | 0.695/0.596 | 0.680/0.602 | 0.681/0.616 | 0.648/0.572 | 0.635/0.581 | 0.683/0.585 | 0.679/0.584 |
| J48              | 0.850/0.705 | 0.846/0.662 | 0.828/0.693 | 0.804/0.636 | 0.824/0.675 | 0.821/0.629 | 0.835/0.659 | 0.824/0.663 | 0.831/0.690 |
| RF               | 0.894/0.730 | 0.877/0.696 | 0.875/0.693 | 0.853/0.693 | 0.875/0.677 | 0.848/0.695 | 0.855/0.682 | 0.872/0.693 | 0.879/0.710 |
| AdaBoost         | 0.709/0.696 | 0.670/0.659 | 0.642/0.625 | 0.602/0.590 | 0.706/0.665 | 0.656/0.614 | 0.598/0.636 | 0.636/0.605 | 0.627/0.658 |
| Eclipse          |        |        |        |        |        |        |        |        |        |
| NB               | 0.911/0.971 | 0.868/0.854 | 0.782/0.732 | 0.735/0.627 | 0.632/0.608 | 0.656/0.547 | 0.641/0.531 | 0.661/0.550 | 0.641/0.531 |
| SVM              | 0.910/0.852 | 0.843/0.784 | 0.790/0.686 | 0.741/0.583 | 0.685/0.511 | 0.629/0.438 | 0.660/0.487 | 0.692/0.493 | 0.660/0.487 |
| J48              | 0.923/0.951 | 0.901/0.846 | 0.886/0.781 | 0.872/0.711 | 0.842/0.675 | 0.813/0.658 | 0.821/0.679 | 0.859/0.728 | 0.821/0.679 |
| RF               | 0.970/0.95 | 0.943/0.846 | 0.923/0.680 | 0.872/0.659 | 0.864/0.659 | 0.849/0.612 | 0.845/0.603 | 0.871/0.641 | 0.845/0.603 |
| AdaBoost         | 0.870/0.951 | 0.860/0.852 | 0.811/0.745 | 0.672/0.590 | 0.489/0.496 | 0.487/0.478 | 0.700/0.679 | 0.611/0.588 | 0.700/0.654 |
| Scarab           |        |        |        |        |        |        |        |        |        |
| NB               | 0.733/0.719 | 0.746/0.687 | 0.714/0.687 | 0.723/0.705 | 0.686/0.692 | 0.672/0.635 | 0.621/0.557 | 0.704/0.612 | 0.656/0.618 |
| SVM              | 0.779/0.724 | 0.856/0.721 | 0.833/0.691 | 0.799/0.686 | 0.756/0.676 | 0.770/0.664 | 0.747/0.573 | 0.719/0.633 | 0.762/0.668 |
| J48              | 0.870/0.724 | 0.855/0.740 | 0.841/0.690 | 0.850/0.710 | 0.852/0.676 | 0.854/0.712 | 0.826/0.635 | 0.832/0.706 | 0.862/0.670 |
| RF               | 0.890/0.783 | 0.890/0.768 | 0.883/0.748 | 0.895/0.767 | 0.885/0.744 | 0.885/0.721 | 0.870/0.677 | 0.879/0.700 | 0.883/0.698 |
| AdaBoost         | 0.783/0.756 | 0.728/0.713 | 0.720/0.712 | 0.781/0.723 | 0.679/0.716 | 0.704/0.680 | 0.661/0.640 | 0.713/0.656 | 0.663/0.656 |
|                | Columba      | Eclipse      | Scarab       |
|----------------|--------------|--------------|--------------|
|                | 0%           | 10%          | 20%          | 30%          | 40%          | 50%          | 60%          | 70%          | 80%          |
|                | 0%           | 10%          | 20%          | 30%          | 40%          | 50%          | 60%          | 70%          | 80%          |
| **NB**         | 0.717/0.687  | 0.694/0.640  | 0.699/0.655  | 0.706/0.696  | 0.683/0.652  | 0.710/0.660  | 0.721/0.647  | 0.719/0.649  | 0.692/0.652  |
| **SVM**        | 0.580/0.525  | 0.544/0.507  | 0.589/0.520  | 0.629/0.558  | 0.620/0.560  | 0.628/0.555  | 0.626/0.569  | 0.617/0.532  | 0.573/0.517  |
| **J48**        | 0.843/0.650  | 0.845/0.648  | 0.830/0.748  | 0.811/0.632  | 0.847/0.630  | 0.828/0.653  | 0.847/0.669  | 0.840/0.664  | 0.833/0.661  |
| **RF**         | 0.951/0.793  | 0.941/0.756  | 0.949/0.748  | 0.937/0.745  | 0.948/0.737  | 0.940/0.760  | 0.950/0.750  | 0.946/0.748  | 0.951/0.769  |
| **AdaBoost**   | 0.748/0.714  | 0.704/0.687  | 0.636/0.651  | 0.630/0.660  | 0.658/0.649  | 0.656/0.617  | 0.623/0.635  | 0.682/0.657  | 0.649/0.663  |
|                | 0.851/0.748  | 0.818/0.765  | 0.789/0.696  | 0.771/0.663  | 0.711/0.658  | 0.741/0.637  | 0.729/0.648  | 0.751/0.642  | 0.729/0.648  |
| **SVM**        | 0.566/0.532  | 0.589/0.535  | 0.598/0.535  | 0.645/0.546  | 0.637/0.544  | 0.653/0.522  | 0.674/0.575  | 0.691/0.568  | 0.674/0.575  |
| **J48**        | 0.793/0.742  | 0.848/0.650  | 0.858/0.70  | 0.854/0.712  | 0.859/0.748  | 0.859/0.753  | 0.876/0.749  | 0.895/0.767  | 0.876/0.746  |
| **RF**         | 0.967/0.964  | 0.951/0.846  | 0.951/0.783  | 0.953/0.789  | 0.944/0.789  | 0.931/0.743  | 0.937/0.739  | 0.944/0.670  | 0.937/0.739  |
| **AdaBoost**   | 0.831/0.742  | 0.801/0.806  | 0.732/0.685  | 0.570/0.648  | 0.638/0.648  | 0.547/0.546  | 0.549/0.749  | 0.581/0.461  | 0.549/0.558  |
|                | 0.792/0.772  | 0.785/0.750  | 0.744/0.717  | 0.780/0.733  | 0.726/0.697  | 0.709/0.665  | 0.647/0.589  | 0.724/0.669  | 0.733/0.671  |
| **SVM**        | 0.779/0.724  | 0.851/0.721  | 0.828/0.683  | 0.796/0.676  | 0.750/0.661  | 0.736/0.616  | 0.728/0.546  | 0.710/0.625  | 0.762/0.665  |
| **J48**        | 0.889/0.724  | 0.874/0.744  | 0.862/0.712  | 0.858/0.685  | 0.853/0.677  | 0.839/0.682  | 0.859/0.649  | 0.865/0.703  | 0.875/0.668  |
| **RF**         | 0.960/0.869  | 0.963/0.840  | 0.950/0.830  | 0.968/0.839  | 0.956/0.814  | 0.959/0.792  | 0.946/0.636  | 0.957/0.787  | 0.961/0.783  |
| **AdaBoost**   | 0.865/0.830  | 0.802/0.793  | 0.794/0.771  | 0.837/0.772  | 0.750/0.730  | 0.694/0.710  | 0.685/0.651  | 0.721/0.741  | 0.729/0.729  |
Fig. 8  Boxplot over range of True positive rate (TPR) value at various noise level for all three dataset

Table 6 and Fig. 7 report the ROC of all five models under various noise levels. The maximum ROC value for pure Columba data is 0.951, which is produced by the RF-based model, as shown in Fig. 7b. Then, after J48-based SDP has ROC value, i.e., 0.843. Lowest ROC produced by SVM-based method with 0.58 ROC value. NB and AdaBoost-based SDP model has a moderate performance with 0.717 and 0.748 ROC values, respectively.

The variation of ROC values for Eclipse data is shown in Fig. 7a. The maximum and minimum ROC values for pure Columba dataset are 0.969 (RF), and 0.566 (SVM), respectively. In Table 6, we can see that RF-based SDP outperform at every noise level as compared with other baseline models. ROC produced by RF-based technique at 10, 20, 30, 40, 50, 60, 70, and 80% noise level are 0.960, 0.963, 0.950, 0.968, 0.956, 0.959, 0.946, 0.957 and 0.961, respectively.

The ROC of Scarab data at various noise degree is shown in Fig. 7c. The maximum ROC value at 0% noise is 0.960, which is processed by the proposed model, followed by J48-based model with 0.889 with ROC value. The ROC curve generated by RF-based SDP is almost uniform with the least deviation compared with other methodologies, indicated RF outperforms over every SDP model at various noise levels.

The boxplot range of performance measures of various SDP models over 0–80% noise levels is shown in Figs. 8, 9, 10, 11 and 12. Figure 8 reports the boxplot range of TPR value at different noise level for all three datasets. We can see in Fig. 8a, the range of TPR value for NB-based SDP model is from 0.608 (20% noise) to 0.677 (0% noise), for SVM 0.536 (80% noise) to 0.744 (0% noise), for J48 0.805 (30% noise) to 0.851 (0% noise), for RF 0.848 (50% noise) to 0.975 (0% noise), and for AdaBoost 0.489 (60% noise) to 0.731 (0% noise). Similarly for Eclipse dataset, the boxplot range for NB-based SDP is 0.612 (40% noise) to 0.854 (0% noise), for SVM 0.536 (80% noise) to 0.744 (0% noise), for J48 0.805 (30% noise) to 0.851 (0% noise), for RF 0.848 (50% noise) to 0.893 (0% noise), and for AdaBoost 0.591 (60% noise) to 0.731 (0% noise). Similarly for Eclipse dataset, the boxplot range for NB-based SDP is 0.612 (40% noise) to 0.854 (0% noise), for SVM 0.618 (50% noise) to 0.910 (0% noise), for J48 0.813 (50% noise) to 0.930 (0% noise), for RF 0.844 (60% noise) to 0.975 (0% noise), and for AdaBoost 0.489
Fig. 8b shows the boxplot range of TPR over various SDP models with corresponding noise value for Eclipse dataset. As Fig. 8 reports, the range of NB-based SDP is from 0.612 (40% noise) to 0.854 (0% noise), range of SVM-based model is from 0.618 (50% noise) to 0.910 (0% noise), range of J48-based model is from 0.813 (50% noise) to 0.930 (0% noise), RF-based defect prediction model is from 0.844 (60% noise) to 0.975 (0% noise), and the range of AdaBoost is from 0.489 (60% noise) to 0.9 (0% noise). It indicated that most of the models optimally performed at 0% noise, whereas performance degrades as noise increases. As the quartile range of TPR value produced by Adaboost over Eclipse data is maximum as shown in Fig. 8b, that indicates the model is unstable and misclassified instance. Figure 8c shows the boxplot range of TPR for Scarab dataset by four baseline models at various noise levels. TPR range by NB-based SDP is from 0.616 (60% noise) to 0.732 (10% noise), the range of SVM-based SDP is from 0.714 (70% noise) to 0.852 (10% noise), range of J48-based model is from 0.826 (60% noise) to 0.867 (0% noise), for RF-based SDP, the range is from 0.870 (60% noise) to 0.890 (0% noise), and for Adaboost-based technique, the range lies between 0.660 (80% noise) to 0.783 (0% noise).

Figure 9 shows the boxplot range of FPR over various SDP models at different noise level for all three datasets. The boxplot range of FPR values over Columba data is shown in Fig. 9a. The FPR range of NB-based model lies from 0.317 (60% noise) to 0.395 (20% noise). As the quartile range of FPR value produced by SVM-based method over Eclipse and Columba datasets is maximum, as shown in Fig. 9a, b, respectively, that indicates the model is unstable and misclassified instance. SVM-based model lies from 0.383 (60% noise) to 0.611 (10% noise), J48-based technique the FPR value lies from 0.168 (60% noise) to 0.216 (0% noise), RF-based defect prediction model the value lies from 0.146 (60% noise) to 0.261 (10% noise), and for AdaBoost defect prediction model FPR values lies between 0.402 (60% noise) and 0.792 (0% noise). The FPR range for Eclipse data is shown in Fig 9b, the range of FPR values for NB-based model lies from 0.220 (30% noise) to 0.284 (40% noise), for SVM-based model FPR range lies from 0.294 (60% noise) to 0.778 (0% noise), for J48-based technique FPR value lies from 0.111 (70% noise) to 0.458 (0% noise), for RF-based model FPR value lies between 0.106 (70% noise) and 0.261 (10% noise), and for Adaboost-based SDP method FPR lies from 0.403 (60% noise) to 0.792 (0% noise). Figure 9c shows the FPR for Scarab dataset at various noise level, for NB defect
perdition model the FPR range is from, 0.272 (10%) to 0.392 (60%), SVM-based model FPR lies from 0.176 (20%) to 0.294 (70%), J48-based technique lies between 0.132 (0%) to 0.188 (50%), RF-based model lies from 0.109 (30%) to 0.147 (50%), and for AdaBoost model FPR range is from 0.217 (0%) to 0.470 (50%). The quartile range of FPR value produced by Adaboost-based SDP model over Scarab data is maximum, as shown in Fig. 9c; it reports that the model is least stable and misclassified actual instances.

Figure 10 shows the boxplot range of F-measure over all three dataset produced by five baselines methods under various noise levels. Figure 10a reports the boxplot range of f-score produced by defect prediction methods. The f-score range for Columba data produced by NB, SVM, J48, RF, and AdaBoost-based SDP models is 0.614 (20%) to 0.68 (0% noise), 0.565 (40% noise) to 0.682 (0% noise), 0.821 (50% noise) to 0.851 (0% noise), 0.847 (50% noise) to 0.889 (0% noise), and 0.570 (20% noise) to 0.709 (0% noise); similarly for Eclipse data 0.616 (40% noise) to 0.974 (0% noise), 0.613 (50% noise) to 0.879 (0% noise), 0.813 (50% noise) to 0.924 (0% noise), 0.844 (60% noise) to 0.968 (0% noise), 0.370 (60% noise) to 0.870 (0% noise) as shown in Fig. 10b. The quartile range of f-score value processed by Adaboost-based defect prediction model over Scarab and Eclipse dataset is maximum, as shown in Fig. 10b, c, respectively. It indicates that models are least stable and misclassified the actual buggy instances. The boxplot range for f-score over Scarab data is shown in Fig. 10c, and the range for NB, SVM, J48, RF, and AdaBoost-based SDP models is 0.618 (60%) to 0.727 (0%), 0.711 (70%) to 0.852 (10%), 0.832 (70%) to 0.867 (0%), 0.870 (60%) to 0.895 (30%), and 0.621 (60%) to 0.783 (0%), respectively.

Figure 11 shows the boxplot range for precision by all four SDP models; the values were calculated at 0–80% noise level. Figure 11a shows the boxplot range of precision over Columba data. The range of precision for NB, SVM, J48, RF, and AdaBoost-based SDP over Columba dataset is 0.629 (20% noise) to 0.713 (0% noise), 0.616 (40% noise) to 0.752 (0% noise), 0.804 (30% noise) to 0.850 (0% noise), 0.848 (50% noise) to 0.894 (0% noise), and 0.598 (60% noise) to 0.709 (0% noise), respectively. Similarly for Eclipse dataset, the values lie from 0.632 (40% noise) to 0.911 (0% noise), 0.629 (50% noise) to 0.910 (0% noise), 0.813 (50% noise) to 0.923 (0% noise), 0.845 (60% noise) to 0.970 (0% noise), and 0.487 (50% noise) to 0.870 (0% noise), respectively, as shown in Fig. 11b. The boxplot range of FPR value produced
by Adaboost-based model over Eclipse data is maximum, as shown in Fig. 11b; it signifies that the model is unstable and misclassified true instances. The range for Scarab dataset when NB, SVM, J48, RF, and AdaBoost-based SDP model where applied is 0.612 (60% noise) to 0.746 (0% noise), 0.719 (70% noise) to 0.856 (0% noise), 0.826 (60% noise) to 0.870 (0% noise), 0.870 (60% noise) to 0.890 (0% noise), and 0.661 (60% noise) to 0.783 (0% noise), respectively, as shown in Fig. 11b. It indicates that, when the model is unstable or overfitted the range of precision values will be high. When the model is good fit, the range is shorter. The shortest range of values is of RF-based SDP model, so it is highly stable and good fit model.

Figure 12 reports the boxplot range of ROC over datasets with various noise level produced by five baselines methods. The boxplot range for Columba data is shown in Fig. 12a, and the ROC range for Columba data produced by NB, SVM, J48, RF, and AdaBoost-based SDP models is from 0.655 (20% noise) to 0.719 (70% noise), 0.544 (10% noise) to 0.629 (30% noise), 0.811 (30% noise) to 0.847 (60% noise), 0.937 (30% noise) to 0.951 (0% noise), and 0.623 (60% noise) to 0.748 (0% noise), respectively. Similarly for Eclipse data boxplot range of ROC are 0.711 (40% noise) to 0.851 (0% noise), 0.691 (70% noise) to 0.566 (0% noise), 0.895 (70% noise) to 0.793 (0% noise), 0.931 (50% noise) to 0.967 (0% noise), 0.547 (50% noise) to 0.831 (0% noise), respectively, as shown in Fig. 12b. The quartile range of ROC value processed by Adaboost-based defect prediction model over Scarab and Eclipse dataset is maximum, as shown in Fig. 11b, c, respectively. It suggests that the models are least stable and misclassified the actual true buggy instances. Figure 12c presents the boxplot range of ROC value over Scarab dataset for NB, SVM, J48, RF, and AdaBoost-based models are 0.647 (60% noise) to 0.792 (0% noise), 0.710 (70% noise) to 0.851 (10% noise), 0.839 (50% noise) to 0.946 (60% noise) to 0.968 (30% noise), and 0.685 (60% noise) to 0.964 (50% noise), respectively.

**5.4 How does the class imbalance problem affect the performance of various SDP models over different noise levels?**

We have also conducted experiments over datasets without applying sampling techniques at various noise levels. The second column of each Table from Tables 2, 3, 4, 5 and 6 reports the performance metrics without using the sampling
method at a different noise level. Its easy to analyze from Table 2 that for each non-sampling-based SDP model over all three datasets. The TPR value is less at various noise levels compared with sampling-based SDP models. Similarly, the FPR produced by SDP methods in which the sampling technique is not utilized, and they have higher FPR values compared with sampling-based methods, as shown in Table 3. f-score values of every model in which sampling technique is not applied have lesser compared with sampling-based SDP techniques over every dataset. The precision and ROC values are even lower at different noise levels produced by imbalanced SDP models compared with sampling-based SDP models, as shown in Tables 5 and 6, respectively. Although at high noise level (60–80%), the sampling-based SDP model misclassifies the actual class in some rare cases, which causes the worst performance. Since in very few cases, the non-sampling-based SDP outperforms because the cardinality of buggy instance is more than a clean instance; it implies the model is overfitted toward buggy instances. But sampled-based SDP models do not overfitted at any noise level.

5.5 Compare the performance of proposed approach with other classical SDP models without applying sampling method.

As we discussed earlier, all three datasets are imbalanced. In this section, we compared the performance of imbalanced baseline methods with an imbalanced suggested approach over every dataset. The TPR value of RF without applying the sampling method is higher than all non-sampling classical methods, as shown in Table 2. The maximum TPR value by RF without using sampling technique on pure Columba, Eclipse, and Scarab datasets is 0.748 at 0%, 0.940 at 0%, and 0.783, respectively, whereas the highest TPR value by other classical technique without applying sampling-based model is 0.724 (AdaBoost at 0%), 0.970 (SVM at 0%), and 0.756 (AdaBoost at 0%), respectively, as shown in Table 2. Lowest FPR value reported by RF not having applied sampling technique on Columba, Eclipse, and Scarab datasets are 0.317 at 50%, 0.296 at 50%, and 0.217 at 0% noise, respectively. Whereas the minimum FPR value reported by other classical models without applying the sampling-based model are 0.355 (SVM at 0%), 0.238 (J48, AdaBoost at 60%), and 0.227 (J48 at 0%), respectively, as shown in Table 3.
The maximum value of F-measure reported by RF when the sampling method is not applied over pure Columba, Eclipse, and Scarab datasets is 0.718, 0.921, and 0.783, respectively, as shown in Table 4. Although the maximum F-measure value reported by other traditional models without applying the sampling technique are 0.707 (J48 at 0%), 0.965 (SVM at 0%), and 0.755 (AdaBoost at 0%), respectively. We have also reported the precision value of the classical model and suggested approach in Table 5. The uppermost precision value reported by RF when sampling is not utilized over pure Columba, Eclipse, and Scarab datasets are 0.720, 0.951, and 0.783, respectively. Whereas the maximum precision value reported by other traditional models over pure datasets when no sampling method applied is 0.705 (J48), 0.971 (SVM), and 0.756 (AdaBoost), respectively. The uppermost ROC value produced by RF without utilizing sampling technique over pure Columba, Eclipse, and Scarab datasets is 0.793, 0.964, and 0.744, respectively, as reported in Table 6. Although the maximum F-measure value produced by other traditional models without applying sampling technique is 0.748 (J48 at 20%), 0.806 (AdaBoost at 10%), and 0.830 (AdaBoost at 0%), respectively. As reported above, in most of the cases, the classical classifiers outperform at 0% noise level. As all the 3 datasets are imbalanced, so it leads to an overfitted model and produces biased results. But RF avoids overfitting (Ali et al. 2012) up to some extent.

5.6 Insightful discussion

When the noise level increases in datasets, the learning technique started misclassified the actual class, and the performance of classical SDP models degrades. As the noise level increases, the number of actual class degrades, and the model becomes predicting the wrong class as an actual class. Although when the sampling method not applied over traditional baseline models, due to the imbalance dataset, the classifiers started overfitting and leads to unsatisfactory outperformed results. AdaBoost is an ensemble learning (EL) method; the EL methods mainly split the dataset and combined the results. EL is also avoids overfitting problem (Rätsch et al. 1998). In a few cases, AdaBoost outperformed over RF-based model when sampling technique is not applied; such results are unbiased.

When we applied the sampling technique, the proposed model outperforms over each SDP model approximately in every noise level. In very few cases, the AdaBoost and SVM surpass the performance. When noise level increases, the classical SDP models started degrading its performances, because the actual class started reducing and model start predicting the notional classes. RF and J48 are tree-based models, and the leaf node represents the class. RF provides an improvement over other trees model by way of small tweaking that decorrelates the tree. At every split in the RF, the algorithm is not even allowed to consider a majority of the available predictors (possible square root of the full set). RF method uses the square root of total predictors causes better results when the noise level increases. It also offers efficient estimates of the test error without incurring the cost of repeated model training associated with cross-validation, so it is sufficient to avoid notional class and predict the actual class.

6 General discussion and threats to validity

We conducted a significant test using the Wilcoxon Rank-Sum test (Lam and Longnecker 1983) the noise versus clean performance of the proposed model and other SDP models at different noise levels for all three datasets. In Table 7, we have listed the ROC value of the proposed model and other optimal SDP models at various noise level. In Table 7, we reported corresponding ROC values of the proposed model and other optimal baselines methods at particular noise levels. Obuchowski (1997) suggested that nonparametric testing using ROC is effective over other evaluation metrics. We have taken two samples, in the first sample ($S_1$), we have listed ROC values of the proposed model with increasing order of noise level from 0 to 80%, whereas in Sample two ($S_2$), we have listed the ROC of most optimal SDP model at that noise level in the same order of noise. The hypothesis $H_0$ is the median (difference) between two samples is 0, and hypothesis $H_1$ is the median (difference) > 0. The sample size $n_1 = n_2 = 27$. Based on the information provided, the significance level is $\alpha = 0.005$, and the critical value for a right-tailed test is $z_c = 2.58$. The rejection region for this right-tailed test is $R = z : z > 2.58$, where $R$ is the rank sum of sample $n_1$, and $n_1$ is 1082. We got $z = 5.873$ since it is observed that $z = 5.873 > z_c$; its concluded that the null hypothesis is rejected. Therefore, there is enough evidence to claim that the population median of differences is greater than 0, at the 0.005 significance level.

Few threats to the validity of these experiments are follows.

- We have collected an open-source dataset for our experiments, the types of noise present in open source dataset and software available in a large organization may be different because of data acquisition by different trained employees. It will be better if private industries reveal their dataset so that it can be tested over noise resistance and class imbalance problem.
- We have used public dataset as a pure dataset, but there can be some instances which are not correctly linked, and some defect items are not adequately lined by SCM. It is also possible that few defects may not be recorded by a bug tracking system.
Table 7  **Comparison of ROC of Random forest-based SDP with respect to other best SDP model at different noise level**

| Noise Level (%) | Columba | Eclipse | Scarab |
|-----------------|---------|---------|--------|
| 0%              | RF 0.951 | RF 0.967 | RF 0.960 |
|                 | Other 0.748 (AB) | Other 0.851 (NB) | Other 0.889 (J48) |
| 10%             | 0.941 | 0.951 | 0.963 |
| 20%             | 0.949 | 0.951 | 0.950 |
| 30%             | 0.937 | 0.953 | 0.968 |
| 40%             | 0.948 | 0.944 | 0.956 |
| 50%             | 0.940 | 0.931 | 0.959 |
| 60%             | 0.950 | 0.937 | 0.946 |
| 70%             | 0.946 | 0.944 | 0.957 |
| 80%             | 0.951 | 0.937 | 0.961 |

Note here AB is AdaBoost

- We have not considered feature noise in our study, and this noise also impacts the performance of an SDP model.
- As we have randomly added noise in the public dataset by changing class labels, but it can be possible that sound can follow the specified pattern. That pattern can be because of poorly managed data during development.
- It is challenging to perform significant analysis between all five performance measures. It needs a multi-variant significant nonparametric test.
- We have used TPR, FPA, F-measure, Precision and ROC performance measures which have been widely used in SDP (Pandey et al. 2020a; Nam et al. 2013; Peters et al. 2013), another threats to validity to our conclusion.
- We performed Wilcoxon signed-rank test to investigate the performances made by various approaches; it is a classical method to validate significant improvements over these methods.
- In future, we plan to reduce threats by performing experiments over other diverse datasets.

7 Conclusion and future work

Noise and class imbalance problems are the two significant challenges in SDP. We have performed 864 experiments over 3 public datasets and analyzed the noise endure for well-known SDP models. We have manually added noise into it from 0 to 80%. We have used 4 baseline SDP methods and trained them using these noisy datasets. We have used random sampling to avoid the class imbalance problem. We also suggested an approach that can tolerate maximum noise and still outperforms over baseline methods. We have also compared the performance without applying sampling methods. We found the proposed approach surpasses the performance over baseline technologies with noisy instances and with imbalanced data. We have also provided a few guidelines. Additionally, we have concluded a few points that are listed below.

(i) We have applied Random sub-sampling as a sampling technique which provides the most effective results compared with other sampling techniques.
(ii) Random forest outperforms compared with other state of the art techniques. RF has high noise tolerate rate (30–40%) compared with other methodologies.
(iii) AdaBoost is least capable, and it has very lesser noise dealing capacity, i.e., from 10 to 20% only.
(iv) J48 is also approximately active as random forest and has a higher level of noise dealing capacity in the range of 30–40%.
(v) The TPR and FPR of RF have the least deviation; however, SVM and AdaBoost have high variation toward the noise. J48 and NB have an average difference after noise is added.
(vi) The f-score and ROC of RF are consistently similar in every noise scenario for all three data. SVM and NB have a high deviation when noise is added. J48 and AdaBoost have moderate deviation.
(vii) Naive Bayes and SVM are moderately active and have an intermediate level of noise tolerance ability, Naive Bayes has up 30%, and SVM has up to 40% noise bear level.

We have used public datasets; software industries should reveal their project data so that better data sources can be available for research purposes. Noise dealing algorithms need to be suggested because no such algorithm is present to deal with noise in defect data items.

There is a scope of ensemble learning in software bug tracking systems; it can outperform with state-of-the-art techniques. There is still deep learning-based model is not available till now because of lesser number of instance in a dataset, by applying data augmentation, we can make our training set bigger so that deep learning-based architecture can easily apply. Even deep learning architecture can be used as a feature selection method. Cross defect bug tracking systems can also be helpful for different types of software
systems, and we must be careful while combining other metrics and their datasets because it can create redundancy, which affects the performance of the learning model.

Acknowledgements Authors would like to thanks IIT-BHU for providing such a vital research platform for students researchers and faculties. We also want to thanks faculty members and students of the CSE department in IIT-BHU for their valuable comments and motivation.

Declarations

Conflict of interest The authors declare that there is no conflict of interest regarding the publication of this paper.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Abăei G, Selamat A, Fujita H (2015) An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. Knowl Based Syst 74:28–39
Alan O, Catal C (2011) Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets. Expert Syst Appl 38(4):3440–3445
Ali J, Khan R, Ahmad N, Masood I (2012) Random forests and decision trees. Int J Comput Sci Issues (IJCSI) 9(5):272
Ari A, Mariyam SS, Ralescu AL (2015) Classification with class imbalance problem: a review. Int J Adv Soft Comput Appl 7(3):176–204
Bachmann A, Bird C, Rahman F, Devanbu P, Bernstein A (2010) The missing links: bugs and bug-fix commits. In: Proccedings of the eighteenth ACM SIGSOFT international symposium on foundations of software engineering. ACM, pp 97–106
Bennin KE, Keung JW, Monden A (2019) On the relative value of data resampling approaches for software defect prediction. Empir Softw Eng 24(2):602–636
Bhargava N, Sharma G, Bhargava R, Mathuria M (2013) Decision tree analysis on j48 algorithm for data mining. Proc Int J Adv Res Comput Sci Softw Eng 3(6)
Bird C, Bachmann A, Aune E, Duffy J, Bernstein A, Filkov V, Devanbu P (2009) Fair and balanced?: Bias in bug-fix datasets. In: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering. ACM, pp 121–130
Cabral GG, Minku LL, Shihab E, Mujahid S (2019) Class imbalance evolution and verification latency in just-in-time software defect prediction. In: 2019 IEEE/ACM 41st international conference on software engineering (ICSE). IEEE, pp 666–676
Catal C (2014) A comparison of semi-supervised classification approaches for software defect prediction. J Intell Syst 23(1):75–82
Catal C, Diri B (2007) Software defect prediction using artificial immune recognition system. In: Proceedings of the 25th conference on IASTED international multi-conference: software engineering. ACTA Press, pp 285–290
Catal C, Diri B (2009) Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Inf Sci 179(8):1040–1058
Catal C, Diri B, Ozumut B (2007) An artificial immune system approach for fault prediction in object-oriented software. In: 2nd International conference on dependability of computer systems (DepCoS-RELCOMEX’07). IEEE, pp 238–245
Catal C, Sevim U, Diri B (2010) Metrics-driven software quality prediction without prior fault data. In: Electronic engineering and computing technology. Springer, Berlin, pp 189–199
Catal C, Alan O, Balkan K (2011a) Class noise detection based on software metrics and ROC curves. Inf Sci 181(21):4867–4877
Catal C, Sevim U, Diri B (2011b) Practical development of an eclipse-based software fault prediction tool using Naive Bayes algorithm. Expert Syst Appl 38(3):2347–2353
Chandrashekar G, Sahin F (2014) A survey on feature selection methods. Comput Electr Eng 40(1):16–28
Charte F, Rivera AJ, del Jesus MJ, Herrera F (2015) Addressing imbalance in multilabel classification: measures and random resampling algorithms. Neurocomputing 163:3–16
Chawla NV (2009) Data mining for imbalanced datasets: an overview. In: Data mining and knowledge discovery handbook. Springer, Berlin, pp 875–886
Chawla NV, Bowyer KW, Hall LO, Philip KW (2002) Smote synthetic minority over-sampling technique. J Artif Intell Res 16:321–357
Chen L, Fang B, Shang Z, Tang Y (2018) Tackling class overlap and imbalance problems in software defect prediction. Softw Qual J 26(1):97–125
Clarkson KL, Shor PW (1989) Applications of random sampling in computational geometry. ii. Discret Comput Geom 4(5):387–421
Davies D (1995) Parallel processing with subsampling/spreading circuitry and data transfer circuitry to and from any processing unit
Davis J, Goadrich M (2006) The relationship between precision-recall and roc curves. In: Proceedings of the 23rd international conference on machine learning. ACM, pp 233–240
Dor O, Zhou Y (2007) Achieving 80% ten-fold cross-validated accuracy for secondary structure prediction by large-scale training. Proteins Struct Funct Bioinform 66(4):838–845
Elish KO, Elish MO (2008) Predicting defect-prone software modules using support vector machines. J Syst Softw 81(5):649–660
Fan Y, Alencar da Costa D, Lo D, Hassan AE, Shanping L (2020) The impact of mislabeled changes by SZZ on just-in-time defect prediction. IEEE Trans Softw Eng
Hall T, Beecham S, Bowes D, Gray D, Steve Counsell (2011) A systematic literature review on fault prediction performance in software engineering. IEEE Trans Softw Eng 38(6):1276–1304
Hou S, Li Y (2009) Short-term fault prediction based on support vector machines with parameter optimization by evolution strategy. Expert Syst Appl 36(10):12383–12391
Hu W, Hu W, Maybank S (2008) Adaboost-based algorithm for network intrusion detection. IEEE Trans Syst Man Cybern Part B (Cybern) 38(2):577–583
Huang Y, Li L (2011) Naive Bayes classification algorithm based on small sample set. In: 2011 IEEE international conference on cloud computing and intelligence systems, pp 34–39
Ji H, Huang S, Wu Y, Hui Z, Zheng C (2019) A new weighted Naïve Bayes method based on information diffusion for software defect prediction. Softw Qual J 1–46
Joon A, Kumar TR, Kumar K (2020) Noise filtering and imbalance class distribution removal for optimizing software fault prediction using best software metrics suite. In: 2020 5th International conference on communication and electronics systems (ICCES). IEEE, pp 1381–1389
Kan SH (2002) Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc
Kaur A, Malhotra R (2006) Application of random forest in predicting fault-prone classes. In: 2008 International conference on advanced computer theory and engineering. IEEE, pp 37–43
Khang DH, Pham T, Wee NS, Tran T, Grundy J, Ghose A, Kim T, Kim C-J (2018) A deep tree-based model for software defect prediction. arXiv preprint arXiv:1802.00921
Kim S, Zimmermann T, Pan K, James Jr E et al (2006) Automatic identification of bug-introducing changes. In: 21st IEEE/ACM inter-
An empirical study toward dealing with noise and class imbalance issues in software defect...

Pandey SK, Tripathi AK (2020) BCV-predictor: a bug count vector predictor of a successive version of the software system. Knowl Based Syst 197:105924

Pandey SK, Mishra RB, Tripathi AK (2018) Software bug prediction prototype using Bayesian network classifier: a comprehensive model. Procedia Comput Sci 132:1412–1421

Pandey SK, Mishra RB, Tripathi AK (2020) BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. Expert Syst Appl 144:113085

Pandey SK, Rathee D, Tripathi AK (2020) Software defect prediction using K-PCA and various kernel-based extreme learning machine: an empirical study. IET Softw 14(7):768–782

Pandey SK, Mishra RB, Tripathi AK (2021) Machine learning based methods for software fault prediction: a survey. Expert Syst Appl 114595

Peng Y, Kou G, Wang G, Wu W, Shi Y (2011) Ensemble of software defect predictors: an AHP-based evaluation method. Int J Inf Technol Decis Mak 10(01):187–206

Peters F, Menzies T, Marcus A (2013) Better cross company defect prediction. In: 2013 10th Working conference on mining software repositories (MSR). IEEE, pp 409–418

Queiroz R, Berger T, Czarnecki K (2016) Towards predicting feature defects in software product lines. In: Proceedings of the 7th international workshop on feature-oriented software development. ACM, pp 58–62

Rätsch G, Onoda T, Müller KR (1998) An improvement of adaboost to avoid overfitting. In: Proceedings of the international conference on neural information processing. Citeseer

Radjenović D, Heričko M, Torkar R, Živković A (2013) Software fault prediction metrics: an systematic literature review. Inf Softw Technol 55(8):1397–1418

Rahman F, Devanbu P (2013) How, and why, process metrics are better. In: 2013 35th International conference on software engineering (ICSE). IEEE, pp 432–441

Ram N, Gerstorf D, Fauth E, Zarit S, Malmberg B (2010) Aging, disablement, and dying: using time-as-process and time-as-resources metrics to chart late-life change. Res Hum Dev 7(1):27–44

Ramlert R, Himmelbauer J (2013) Noise in bug report data and the impact on defect prediction results. In: 2013 Joint conference of the 23rd international workshop on software measurement and the 8th international conference on software process and product measurement. IEEE, pp 173–180

Riaz S, Arshad A, Jiao L (2018) Rough noise-filtered easy ensemble for software fault prediction. IEEE Access 6:46886–46899

Rish I et al. (2001) An empirical study of the naive bayes classifier. In: IJCAI 2001 workshop on empirical methods in artificial intelligence, vol 3, pp 41–46

Roobaert D, Karakoulas G, Chawla NV (2006) Information gain, correlation and support vector machines. In: Feature extraction. Springer, Berlin, pp 463–470

Schneider GM, Martin J, Tsai W-T (1992) An experimental study of fault detection in user requirements documents. ACM Trans Softw Eng Methodol (TOSEM) 1(2):188–204

Seiffert C, Khoshgoftaar TM, Van Hulse J, Folleco A (2014) An empirical study of the classification performance of learners on imbalanced and noisy software quality data. Inf Sci 259:571–595

Shanthini A, Vinodhini G, Chandrasekaran RM, Supraja P (2019) A taxonomy on impact of label noise and feature noise using machine learning techniques. Soft Comput 23(18):8597–8607

Sharma S, Bellinger C, Krawczyk B, Zaiane O, Japkowicz N (2018) Synthetic oversampling with the majority class: A new perspective on handling extreme imbalance. In: 2018 IEEE international conference on data mining (ICDM). IEEE, pp 447–456
Shatnawi R (2017) The application of roc analysis in threshold identification, data imbalance and metrics selection for software fault prediction. Innov Syst Softw Eng 13(2):201–217
Shepperd M, Song Q, Sun Z, Mair C (2013) Data quality: Some comments on the NASA software defect datasets. IEEE Trans Softw Eng 39(9):1208–1215
Shihab E (2012) An exploration of challenges limiting pragmatic software defect prediction. PhD thesis
Song Q, Guo Y, Shepperd M (2018) A comprehensive investigation of the role of imbalanced learning for software defect prediction. IEEE Trans Softw Eng
Suykens JAK, Vandewalle J (1999) Least squares support vector machine classifiers. Neural Process Lett 9(3):293–300
Tan M, Tan L, Dara S, Mayeux C (2015) Online defect prediction for imbalanced data. In: 2015 IEEE/ACM 37th IEEE international conference on software engineering, vol 2. IEEE, pp 99–108
Tantithamthavorn C, McIntosh S, Hassan AE, Ibara A, Matsumoto K (2015) The impact of mislabelling on the performance and interpretation of defect prediction models. In: 2015 IEEE/ACM 37th IEEE international conference on software engineering, vol 1. IEEE, pp 812–823
Tong H, Liu B, Wang S (2018) Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Inf Softw Technol 96:94–111
Wang S, Liu T, Tan L (2016) Automatically learning semantic features for defect prediction. In: 2016 IEEE/ACM 38th international conference on software engineering (ICSE). IEEE, pp 297–308
Wei H, Hu C, Chen S, Xue Y, Zhang Q (2019) Establishing a software defect prediction model via effective dimension reduction. Inf Sci 477:399–409
Witten IH, Frank E (2000) Weka. Machine Learning Algorithms in Java, pp 265–320
Wu X (1995) Knowledge acquisition from databases. Intellect books
Yang X, Lo D, Xia X, Sun J (2017) TLEL: a two-layer ensemble learning approach for just-in-time defect prediction. Inf Softw Technol 87:206–220
Zhang H, Zhang X, Gu M (2007) Predicting defective software components from code complexity measures. In: 13th Pacific rim international symposium on dependable computing (PRDC 2007). IEEE, pp 93–96
Zheng J (2010) Cost-sensitive boosting neural networks for software defect prediction. Expert Syst Appl 37(6):4537–4543
Zhou T, Sun X, Xia X, Li B, Chen X (2019) Improving defect prediction with deep forest. Inf Softw Technol
Zhu X, Wu X (2004) Class noise vs. attribute noise: a quantitative study. Artif Intell Rev 22(3):177–210

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.