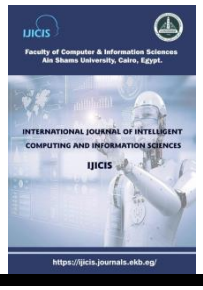
IJICIS, Vol.23, No.2, 28-49
DOI: 10.21608/ijicis.2023.189934.1251

International Journal of Intelligent Computing and Information Sciences

https://ijicis.journals.ekb.eg/



SOFTWARE DEFECT PREDICTION AT METHOD LEVEL USING ENSEMBLE LEARNING TECHNIQUES

Asmaa M Ibrahim*

Department of Software Engineering, Faculty of Computers and Information, Egyptian E-learning University, Assuit , Egypt
amahmoudibrahim@cis.eelu.edu.eg

Hicham Abdelsalam

Department of Computer Science, Faculty of Computers and Information Technology, Egyptian E-learning University, Cairo, Egypt
habdelsalam@cis.eelu.edu.eg

Islam A.T.F Taj-Eddin

Department of Information Technology, Faculty of Computers and Information , Assuit University, Assuit, Egypt
itajeddin@aun.edu.eg

Received 2023-01-26; Revised 2023-05-26; Accepted 2023-05-31

Abstract: *Creating error-free software artifacts is essential to increase software quality and potential re-usability. However, testing software artifacts to find defects and fix them is time consuming and costly, thus predicting the most error-prone software components can optimize the testing process by focusing testing resources on those components to save time and money. Much software defect prediction research has focused on higher granularity, e.g., file and package levels, and fewer have focused on the method level. In this paper, software defect prediction will be performed on highly imbalanced method-level datasets extracted from 23 open source Java projects. Eight ensemble learning algorithms will be applied to the datasets: Ada-Boost, Bagging, Gradient boost, Random Forest, Random Under sampling Boost, Easy Ensemble, Balanced Bagging and Balanced Random Forest. The results showed that the Balanced Random Forest classifier achieved the best results regarding recall and roc_auc values.*

Keywords: *Method-level software defects prediction, Ensemble Learning, Balanced Ensemble Learning, Imbalanced dataset, ELFF datasets.*

1. Introduction

The testing process is very crucial in the Software Development Life Cycle (SDLC), however, it consumes a significant amount of resources (roughly 40% of the overall development project costs) [1] consequently much research focused on improving the testing process to achieve the best resource

***Corresponding Author:** Asmaa M Ibrahim

Software Engineering Department, Faculty of Computer and Information, Egyptian E-learning University, Assuit , Egypt

Email address: amahmoudibrahim@cis.eelu.edu.eg

utilization. Machine learning techniques were used intensively in the field of software testing by predicting defective software components or classifying software components into defective e.g. containing one or more defects or non-defective e.g. defect-free, pursuing this approach, the limited testing resources can be dedicated to the software components predicted to be defective. The classification technique depends on software features or attributes to divide the inputs to the classification process into two classes: defective or nondefective. Those software metrics or attributes might be extracted from the code itself, such as the number of lines of code (NOLC), or from the process by which the software was created, such as the number of developers engaged in creating software.

Many studies focused on predicting software defects at the file or class level, where a file may contain thousands of lines of code and a class may contain tens of methods, implying that more effort is required to locate the defective classes or methods, and consequently predicting defects at the method level can be more efficient and resource-saving [2]. Nonetheless, fewer studies focused on the method-level defect prediction; Hall et al [3] found that from 2000 to 2012, only 12% of defect prediction models were performed at the method, function or procedure level. Luka et al.[4] tried to predict software defects at the method level, and they concluded that software defect prediction at the method level is still an open challenge. The reason why most of the research done in the area of software defect prediction focused on a class or a file level is the lack of reliable software defect datasets at the method level; to address this problem, Thomas Shippey et al.[5] provided the Finding Faults Using Ensemble Learners (ELFF) datasets containing software defect information related to the method level, which are used in this paper. Datasets consist of negative and positive samples e.g. non-fraudulent versus fraudulent transactions, non-diabetic versus diabetic person or non-defective software components vs. defective ones. The datasets are called balanced if they have a relatively equal number of both samples [6]; however, this is not the case in the real world problems where the positive class instances, which are usually the ones of interest, are much less represented than the negative class instances, causing the prediction model to ignore the less represented class. Many techniques have been used to overcome the imbalance of the datasets [7]; Resampling is one of the most commonly used techniques to handle this problem [8][9][10]. Resampling technique can be done either by under-sampling the majority class or over-sampling the minority class, so both classes will have relatively the same number of instances. Ensemble learning, also used to handle the imbalance of the datasets, refers to a group (or ensemble) of base learners or models who work together to improve the final prediction. Due to the significant variation or bias, a single model, also known as a base or weak learner, may not perform effectively on its own; when weak learners are combined, they can produce a strong learner, as their combined bias and variance are reduced, resulting in improved model performance. Bagging and Boosting are common types of ensemble learning methods. Bagging, also known as Bootstrap aggregating, is an ensemble learning strategy that depends on selecting a random sample of data from a training set with replacement meaning that the individual data points can be chosen multiple times, then weak learner(s) are trained individually after multiple data samples are collected. Depending on the type of task, regression, or classification, the average or majority of those predictions yield a more accurate estimate. Boosting on the other hand, is an ensemble learning strategy for minimizing training errors by combining a group of weak learners into a strong learner, a random sample of data is chosen, fitted with a model and then trained progressively

and each model attempts to compensate for the shortcomings of its predecessor, each iteration combines the weak rules from each classifier to generate a single, strong prediction rule, interested readers can find more details about ensemble learning techniques here [11].

In this paper, the gap created by the lack of software defect prediction at the method level is tried to be filled by conducting software defect prediction using the ELFF method level datasets, which contain 63 datasets created from 23 open source Java projects selected with restricted conditions, where each dataset contains 33 source code metrics.

2. Related work

In this section, related literature to software defect prediction at the method level, software defect prediction using ensemble learning techniques, and related work to address the problem of imbalanced datasets will be listed.

2.1.Related work to the software defect prediction at method level

Giger et al.[12] applied three approaches to conduct software defect prediction: using code metrics only, using change metrics only, and using a combination of them. They used precision, recall, and Area Under the Curve (Roc_AUC) to evaluate the prediction model in their work, and they achieved a recall of 85% and a precision of 95% by combining metrics and a random forest classifier. They also discovered that there is no significant difference between classification algorithms. Luca et al.[4] tried to evaluate the results in [12] by replicating the same methodology on different systems and time-spans; they found that the approach on [12] can be generalized with the same results, and then proposed a change to the methodology to overcome the limitations of the Giger et al. [12] approach. To overcome those limitations, Luke et al.[4] re-evaluated the performance of [12] method using data from subsequent releases, i.e., release-by-release validation, and applied the same metrics and the same machine learning algorithms and feature selection approaches. When compared to the random classifier, they discovered that the results dropped dramatically.

Hata et al.[2] also compared bug prediction at the method level, file level, and package level regarding effort-based evaluation using module histories using: first, code-related metrics; second, process metrics; and third; organizational metrics issues. They found that in 20% of the code, the number of bugs found at the method level exceeded those found at the file and package levels and concluded that method level prediction is more effective than package level and file level prediction when considering efforts.

Rainer Niedermayr et al.[13] tried the Inverse Defect Prediction (IDP) method; instead of predicting the most error-prone methods, they tried to predict the less likely error-prone methods or methods that are too trivial to contain bugs, and they used source code metrics to identify low- fault-risk methods.To identify a low-fault-risk method, they used the Association Rule and to overcome the imbalance of the dataset, they used the synthetic minority oversampling technique (SMOTE) technique. Their study showed that IDP can successfully identify methods that are not fault-prone; on

average, 31.7% of the methods (14.8% of the SLOC) matched by the strict classifier contain only 6.0% of all faults.

2.2.Related work to the software defect prediction using ensemble learning techniques

Sammar Mustafa et al.[14] performed the defect prediction at the class level with several voting techniques using weighted majority voting (WM), randomized weighted majority voting (RWM), cascading weighted majority voting (CWM), and cascading randomized weighted majority voting (CRWM). They used three types of metrics: static code metrics, change metrics, and a combination of both. The datasets used were extracted from four Java open source projects with a highly imbalanced distribution of instances. The change code metrics provided the highest results regarding model evaluation metrics, with an f- measure of 98.6%, an accuracy of 74.9% and an auc of 68.5%.

FYucalar et al.[15] tried to empirically study ten ensemble predictors compared to baseline predictor performance; they conducted their study on 15 NASA projects using the Weka workbench and evaluated the prediction model in terms of f-measure and area under the receiver operating characteristic curve (AUC). They concluded that: first, ensemble algorithms can achieve good results, second, increasing the number of base predictors in ensemble algorithms leads to better results, and third, the combination of ensemble algorithms reduces the false alarm rate.

Sweta Mehta & K. Sridhar Patnaik [16] applied several machine learning techniques, including ensemble learning algorithms such as: XGBoost, Stacking, Ada-Boost, and Gradient-Boost, to perform the software defect prediction on datasets from NASA and the PROMISE repository, to address the high dimensionality of the datasets, they used several feature selection techniques, while the problem of high imbalance in the dataset was addressed using the SMOTE oversampling approach.They concluded that the Stacking ensemble learning method and XGBoost provided the best results among the techniques used in their research.

Yakub Kayode Saheed et al.[17] implemented seven ensemble machine learning models for SDP: the Cat Boost, Light Gradient Boosting Machine (LGBM), Extreme Gradient Boosting (XgBoost),Boosted Cat Boost, Bagged Logistic Regression, Boosted LGBM,and Boosted XgBoost in addition to separate base model using Logistic Regression. They used in their research three public datasets from the PROMISE repository and concluded that the proposed ensemble method's performance was significantly better and more competitive than the individual base classifier Logistic Regression model for all defect datasets used in the research.

Abdullateef O. Balogun et al.[18] proposed a combination of synthetic minority oversampling technique (SMOTE) and ensemble classifiers (Bagging and Boosting) in addition to using Decision Tree (DT) and Bayesian Network (BN) algorithms as base classifiers for predicting software defects. Six datasets from the NASA repository were used with unbalance ratios ranging from 2% to 7% and with different numbers of features and modules. First SMOTE oversampling was applied to the datasets, then the four classifiers with 10-fold cross validation were applied to the oversampled datasets; their results showed that the use of SMOTE and homogeneous ensemble not only solved the

issue of class imbalance but also enhanced the base classifier prediction performance. It is worth mentioning that the way of re-sampling followed by [18] was criticized by [19], as the later paper considered that a common flaw usually happens in such situations where the entire dataset is over/under sampled before applying cross validation, this approach may lead to overly optimistic results, while the right way to over/under sample the data according to [19] is within the cross validation folds.

Inas Abuqaddom and Amjad Hudaib [20] Used SMOTE oversampling technique along with Cost Sensitive Learning (CSL) and the Ada-Boost algorithm to handle imbalanced dataset issues. They applied their approach to four datasets from the PROMISE repository with imbalance ratios ranging from 9.84% to 19.35%. They compared their approach to three other approaches: Decision Tree (DT), Ada-Boost with DT as its base classifier, and SMOTE-Ada-Boost. Their findings showed that their proposed approach improved the prediction of software defects.

Aljamaan and A.Alazba [21] investigated seven ensemble learning classifiers in defect prediction, which were: Random Forest, Extra Trees, Ada Boost, Gradient Boosting, Heist Gradient Boosting, XGBoost, and Cat-Boost. They performed their approach on 11 publicly available MDP NASA software defect datasets, and their results showed that Random Forest and Extra Trees classifiers performed the best while Ada-Boost performed the worst.

Saifan et al. [22] conducted two ensemble learning algorithms (Bagging and Boosting) and two single classifiers (KNN and SVM) along with four feature selection techniques, which were Principal Component Analysis (PCA), Pearson's correlation, Greedy Stepwise Forward Selection, and Information Gain (IG). They applied their approach to five datasets obtained from the PROMISE software repository and concluded that ensemble methods can improve a model's performance without any feature selection techniques.

Mohammad Zubair Khan [23] presented a Hybrid Ensemble Learning Technique (HELT) which was performed on eight datasets from the PROMISE repository with an imbalance ratio ranging from 2.1% to 35.2%. According to their study, the HELT technique included the features selection approach, and two algorithms (Classifier-Attribute-Eval and Ranker algorithms) from the Weka tool, then K-Mean clustering was applied to the datasets, and SMOTE oversampling was used to balance the data. Ada-Boost and Bagging ensemble techniques were used with Naive Bayes, Support Vector Machine and Random Forest classifiers were used as base classifiers for both ensemble techniques. Their results showed that Ada-Boost with SVM and Bagging SVM performed the best in terms of accuracy, recall, precision, and auc. The HELT approach also showed a higher accuracy rate compared to other machine learning techniques.

2.3.Related work to address the problem of imbalanced datasets

Most machine learning models performed in SDP face the problem of imbalanced datasets where defective instances are much less common than non-defective ones, which makes the prediction model biased towards the class with the highest number of instances, giving misleading results. Many

techniques were used to address the problem of imbalanced datasets, such as cost-sensitive classifiers, ensemble learning approaches, and re-sampling techniques.

Malhotra and J.Jain [24] conducted an investigation into software defect prediction at the class level using three datasets from three open-source Java projects from the PROMISE library. To overcome the problem of imbalanced datasets in their research, they used cost-sensitive classifiers. Their research adopted the J48 (decision tree) algorithm and compared it with three ensembles: Boosting-Based ABM1, Bagging-Based Bag, and Random Forest-based RSS. A subset of features was chosen using the Correlation Feature Selection (CFS) method. They concluded that using a cost-sensitive classifier performed better with machine learning algorithms than without it. The re-sampling technique, on the other hand, is a popular approach used in handling imbalanced datasets, KE Benin et al.[25] tried to investigate six re-sampling techniques: synthetic minority over sampling technique (SMOTE), borderline SMOTE ,safe-level SMOTE, Oversampling using adaptive synthetic (ADASYN), random over sampling (ROS), and random under sampling (RUS). The six approaches were conducted on 40 releases of 20 open source projects with an imbalance ratio between the ranges of 3.8 to 17.46%, and two types of metrics were used (process metrics and static code metrics). They used several machine learning algorithms and concluded that: first, resampling methods significantly improved the performance of the prediction model in terms of all model metrics except for auc (Area Under the Roc Curve); second the performance of resampling method depends on the imbalance ratio of the dataset (ratio of defects and clean instances); third, random under-sampling and border-line SMOTE provided more stable results across several performance measures and prediction models.

Abdullah Alsaeedi et al.[26] presented a study to compare several machine learning algorithms (Bagging, Support Vector Machines , Decision Tree , and Random Forest) using 10 NASA datasets; a SMOTE re-sampling technique was used to handle the imbalanced datasets. To evaluate the performance of different classifiers, accuracy, f-measure, and roc_auc metrics were used. Their outcomes showed that Random Forest, Ada-Boost with Random Forest, and Bagging with Decision Tree generally performed well, but they did not reach a decisive result about the best performing algorithm as some classifiers performed well on some datasets and poorly on others.

Shatnawi and Ziad [27] presented a guided oversampling technique in which a module is duplicated in the dataset depending on the number of defects, and then the resulting dataset is oversampled. Not only the guided oversampling approach was used, but also SMOTE oversampling was used with different values of minority class duplication (100%, 200%, 300%, 400% and 500%). The prediction models are first applied to the original dataset without resampling and then to the processed dataset after resampling. They found that the re-sampled datasets with all re-sampling techniques outperformed the datasets with no re-sampling in terms of all prediction model metrics. They also found that their proposed duplicated and re-sampled technique provided better results with and without re-sampling and they also found that for the five classifiers used, their technique had better performance for duplicated without re-sampling datasets, meaning that their proposed technique outperformed the well-known SMOTE .

Thanh Tung Khuat and My Hanh Le.[28] used the random under-sampling technique to address the

problem of imbalanced datasets and used several machine learning algorithms through three steps: first, a sampling step in which majority class samples in the original dataset are split into bins using random under sampling, second, a training step in which each base classifier is trained on the balanced datasets that resulted from the previous step, and third, final classifier is constructed using the results of the base classifier and the majority voting rule. They conducted their study on seven datasets from the PROMISE repository of software defect databases, with a ratio of defective instances ranging from 11.36% to 33.59%. Using f-measure, they found that their ensemble model performed better on re-sampled datasets than non-resampled datasets and outperformed base classifiers on both resampled and non-resampled datasets.

Hamad Alsawalqah et al.[29] used a hybrid technique of SMOTE oversampling and ensemble classifiers to perform software defect prediction on four public benchmark datasets from the PROMISE repository, with a ratio of defective modules ranging from 9.83% to 19.35%. In the first stage of their study, they applied to the datasets three commonly used classifiers: Naive Bayes (NB), Multilayer Perceptron (MLP), and C4.5 decision trees, with the 10-fold cross validation technique. The second stage was to apply three ensemble classifiers (Random Forest, Bagging, and Ada- Boost) to enhance the prediction results from the previous step using Decision Tree as a base classifier that performed the best in the previous stage, and the final step was to combine the ensemble technique with the best results (Ada-Boost) with SMOTE. The amount of oversampling using SMOTE ranged from 20% to 200%. Their results showed that their proposed approach had better quality results than the other classification algorithms.

Haotian Yang and Min Li [30] presented a hybrid approach of SMOTE-Tomek re-sampling and the XGBoost algorithm and applied their technique to 10 NASA datasets and compared their approach with different re-sampling techniques and classification algorithms. They concluded that their approach performed better than the other approaches in terms of auc and f-measure.

3. Research gaps and contributions

Bowes et al.[31] used the ELFF tool provided by the ELFF paper authors to predict faults at an industrial level by plugging the tool into the IntelliJ IDE, which enables developers to perform regular defect prediction on their Java code. ELFF datasets were also used in a regression machine learning model where the number of defects in the newer software version was predicted based on bug-related information obtained from the older version [32]. ELFF class-level datasets were also used in [33] to investigate time and defect velocity in relation to the defect density of a class using the defect density correlation technique.

F. Yang et al.[34] proposed a software defect prediction at the level of method call sequences, while Thomas Shippey et al.[35] proposed a semi-supervised model named DPCAG for the defect prediction task. From the previous literature review, it is found that ELFF method-level datasets were not used in predicting within-project software defects based on the ELFF datasets bug related metrics, nor were they used to investigate the impact of ensemble learning techniques on the severe imbalance in the dataset. The contribution of this paper will be:

1. Investigating the use of ELFF method-level datasets to perform software defect prediction using ELFF software metrics.
2. Investigating the impact of four ensemble learning algorithms (Ada-Boost, Gradient-Boost, Bagging, and Random Forest) and their balanced versions (Random Under-Sampling Boost, Easy Ensemble, Balanced Bagging, and Balanced Random Forest) on the imbalance of the datasets and consequently on the software defect prediction model built on those datasets.

4. Dataset and features

To provide a reliable dataset at the method level, Thomas Shippey et al. [5] provided the Finding Faults Using Ensemble Learners (ELFF) datasets derived from 23 open source Java projects selected on multiple criteria by mining the Boa Source-Forge september 2013 open source dataset. The criteria used to select the 23 projects were:

1. To be written in Java.
2. To be an SVN (subversion version control system) project.
3. Have a Sourceforge defect tracker.
4. Have a Sourceforge tracking system.
5. Have at least 100 fixed defects.

Choosing Java and SVN projects because the research team's tool was designed to extract defects from projects with those criteria, while selecting projects with a certain number of defects was intended to achieve a certain degree of balance in the datasets, as a high degree of unbalancing will mislead the prediction model and cause bias towards the class with a larger number of instances. The software metrics in the ELFF datasets are source code metrics for 69 versions of the 23 projects, and they were extracted per release, unlike the approach followed by Giger et al.[12], where software metrics related to defects were extracted after a specified time span, which is considered a limitation by Luca et al.[4].

Each method in the ELFF datasets has 33 metrics (see Table 1). ELFF datasets are also highly imbalanced, with a much lower percentage of defective methods than non-defective ones. Table 3. lists the percentage of defective instances in the ELFF datasets. In any classification process, the first three metrics (package, class, and method) will be omitted because it is trivial that they will not affect the outcome of the classification process, besides not overwhelming the prediction model with unnecessary data. The rest of the metrics are the well-known source code metrics, such as Halstead's metrics, which are famous static code metrics; interested readers can find more information in [36]. The datasets also contain two additional metrics: Flag and Sequence. Flag determines the type of method: (setter, getter, abstract...), Sequence on the other hand describes the listing of the method e.g. identifier ,block.... The classification process will be applied to 63 datasets out of 69, as six datasets in the ELFF datasets have no defective methods. Compared to previous work that performed defect prediction at the method level, the ELFF datasets has the largest number of features regarding source code metrics, while on the other hand, Giger et al[12] and Luka et al[4] used other types of metrics in the dataset they worked on, Table 2. summarizes the distinction between three papers that investigated the software defect prediction at the method level.

Table 1: ELFF dataset software metrics.

Metric (Software Feature)	Definition
Package	Method's package
Class	Method's Class
Method	Method Name
CC	Cyclomatic complexity It is a quantitative measure of the number of linearl independent paths through a Program is code
NOCL	The number of comment Lines
NOS	The number of statements
HLTH	Halstead length of method
HVOC	Halstead vocabulary of method
HEFF	Halstead effort, the mental effort required to develop or maintain a program
HBUG	Halstead prediction of a number of bugs
CREF	The number of classes referenced.
XMET	The number of external methods.
LMET	Local methods called using the method
NLOC	The number of lines of code
NOC	The number of comments
NOA	The number of arguments
MOD	The number of modifiers
HDIF	Halstead difficulty to implement a method
VDEC	The number of variables declared
EXCT	The number of exceptions thrown using the method
EXCR	The number of exceptions referenced using the method
CAST	The number of class casts
TDN	Total depth of nesting
HVOL	Halstead volume
NAND	The total number of operands
VREF	The number of variables referenced
NOPR	The total number of operators
MDN	Method: Maximum depth of nesting
NEXP	The number of expressions
LOOP	The number of loops (for,while)
FLAG	Type of method Getter, setter, abstract...
SEQUENCE	Listing of method , e.g (method identifier block)
DEFECTIVE	True for buggy method, False for non-buggy method

Table 2: A comparison of this paper's proposal to previous works related to defect prediction at Method Level

Comparisons/Paper	**Method-Level Bug Prediction**	**Re-evaluating Method-Level Bug Prediction.**	**Software Defect Prediction at Method Level Using Ensemble Learning Techniques**
Reference	Giger et al [12]	Luka Et al [4]	This paper
Year	2012	2018	2023
Classifiers	Bayesian Network Support Vector Machine Random Forest J48(Decision tree)	Bayesian Network Support Vector Machine Random Forest J48(Decision tree)	Bagging,Ada-Boost, Gradient Boost, Random Forest, Random Under Sampling Boost,Easy Ensemble, Balanced Bagging and Balanced Random Forest
Number Of Java projects	21	21	23
Date of extracting defects related data	At the end of a time frame.	By release and at the end of a time frame	By release
Type of metrics	Source code, Change metrics	Source code ,Change metrics	Source code metrics
Number of source code metrics	9	9	29
Evaluating metrics	precision, recall, roc_auc	precision, recall, f-measure, roc_auc	precision, recall, roc_auc

5. Methodology

5.1. Selecting the proper machine learning algorithm to build the prediction model on the ELFF datasets

Ensemble learning algorithms were used in the literature to perform software defect prediction in subsection 2.2 as well as to handle the problem of imbalanced datasets as shown in subsection 2.3.In this paper, ensemble learning algorithms with their corresponding balanced versions in the Scikit-Learn Python library are used. Table 4 lists a short description of the eight algorithms.[1]

5.2. Selecting the model metrics

To evaluate the performance of the model, a few terms needed to be reviewed:

1. True positive (TP): positive instances classified as positive (defective methods classified as defective).
2. False positive (FP): negative instances classified as positive (non-defective methods classified as defective).
3. False negative (FN): positive instances classified as negative (defective methods classified as non-defective).
4. True negative (TN): negative instances classified as negative. (non-defective methods classified as non-defective).

[1] https://scikit-learn.org/stable/modules/Ensemble.html

The following model metrics are used to evaluate model performance:

1. Precision is the fraction of true positives retrieved among the total retrieved instances Precision = true positive / (true positive + false positive).
2. Recall is the fraction of true positives that were retrieved over the total number of true positives in the dataset. Recall = true positive / (true positive + false negative).
3. ROC Area is a ROC curve (receiver operating characteristic curve) that is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: the true positive rate and the false positive rate.

Table 3: Percentage of the defective methods for each dataset in the ELFF datasets

Dataset Name	Defective%	Dataset Name	Defective%
metricsFlagsDefects-autoplot2012	1.241	metricsFlagsDefects-jmol9	5.312
metricsFlagsDefects-cdk1.1	1.906	metricsFlagsDefects-jmol10	2.009
metricsFlagsDefects-cmusphinx3.6	0.321	metricsFlagsDefects-jmri2.2	1.0718
metricsFlagsDefects-cmusphinx3.7	0.214	metricsFlagsDefects-jmri2.4	7.238
metricsFlagsDefects-controltier3	0.872	metricsFlagsDefects-jmri2.6	1.194
metricsFlagsDefects-drjava2008	6.940	metricsFlagsDefects-jmri2	0.930
metricsFlagsDefects-drjava2009-	4.544	metricsFlagsDefects-jppf4.1	1.393
metricsFlagsDefects-drjava2010	2.145	metricsFlagsDefects-jppf4.2	1.394
metricsFlagsDefects-eclemma2.1	3.477	metricsFlagsDefects-jppf4	2.727
metricsFlagsDefects-eclemma2	1.0262	metricsFlagsDefects-jppf5.1	0.599
metricsFlagsDefects-ejdt3	1.220	metricsFlagsDefects-jppf5	3.0394
metricsFlagsDefects-genoviz5.4	9.690	metricsFlagsDefects-jtds23072009	1.440
metricsFlagsDefects-genoviz6.1	10.296	metricsFlagsDefects-jump1.5	0.980
metricsFlagsDefects-genoviz6.2	3.689	metricsFlagsDefects-jump1.6	0.732
metricsFlagsDefects-genoviz6.3	2.595	metricsFlagsDefects-jump1.7	0.647
metricsFlagsDefects-genoviz6	10.175	metricsFlagsDefects-jump1.8	0.713
metricsFlagsDefects-htmlunit2008	9.0987	metricsFlagsDefects-jump1.9	1.845
metricsFlagsDefects-htmlunit2009	1.324	metricsFlagsDefects-omegat3.1	0.882
metricsFlagsDefects-htmlunit2010	3.349	metricsFlagsDefects-omegat3.5	1.645
metricsFlagsDefects-jedit5.2	0.178	metricsFlagsDefects-omegat3.6	1.536
metricsFlagsDefects-jikesrvm2	1.1100	metricsFlagsDefects-runawfe3.6	0.015
metricsFlagsDefects-jikesrvm3.1	0.437	metricsFlagsDefects-runawfe4.1	2.942
metricsFlagsDefects-jikesrvm3	2.917	metricsFlagsDefects-saros1.0.6	3.787
metricsFlagsDefects-jitterbit1.1	2.0486	metricsFlagsDefects-tango2008	0.560
metricsFlagsDefects-jitterbit1.2	0.296	metricsFlagsDefects-unicore1.2	4.211
metricsFlagsDefects-jmol2	2.928	metricsFlagsDefects-unicore1.3	2.166
metricsFlagsDefects-jmol3	2.593	metricsFlagsDefects-unicore1.4	7.878
metricsFlagsDefects-jmol4	5.9384	metricsFlagsDefects-unicore1.5	1.728
metricsFlagsDefects-jmol5	5.772	metricsFlagsDefects-unicore1.6	5.0453
metricsFlagsDefects-jmol6	13.143	metricsFlagsDefects-xaware5.1	0.592
metricsFlagsDefects-jmol7	10.227	metricsFlagsDefects-xaware5	1.621
metricsFlagsDefects-jmol8	4.592		

5.3. Data preparation

Processing data before applying a prediction model is a vital step in machine learning techniques, if

data is poorly processed, misleading results will be obtained no matter how good the prediction model is.

In this paper proposal, three steps will be applied to prepare data before applying the prediction model: dropping id-like metrics, metrics, replacing missing values and normalization:

1. Dropping id-like features (package, class and method) (see Table 1) as they will not affect the classification process and to save processing resources as well.
2. Replacing missing values: real-world datasets may not contain all values for all instances due to problems in recording data or obtaining certain values. The Pandas Python library provides several approaches to fill in missing values. In this paper, backward filling will be used, which works by filling missing values with the next set of data points .[2]
3. Normalization: Features (software metrics) in the ELFF dataset differ in range and distribution, causing features with high values to have a higher impact on the classification results. To unify the range of all numeric data from 0 to 1, normalizing is used.

Table 4: A short description of the algorithms used in this paper

Algorithm	Classification: Strategy
Ada-Boost	A meta-estimator that starts by fitting a classifier to the original dataset, then fits multiple copies of the classifier to the same dataset, but adjusts the weights of poorly classified instances so that succeeding classifiers focus more on difficult cases.
Bagging	An ensemble meta-estimators that fit base classifiers to random subsets of the original dataset and then aggregate their individual predictions (either by voting or averaging) to generate a final prediction.
Gradient Boosting	GB permits the optimization of any differentiable loss function and constructs an additive model in a forward stage-wise manner. Each stage involves fitting n classes regression trees on the loss function's negative gradient, such as a binary or multi class log loss.
Random Forest	A meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
Random Under-Sampling Boost	Ada-Boost's learning technique incorporates random undersampling where each iteration of the Boosting method, the problem of class balancing is mitigated by randomly undersampling the sample.
Easy Ensemble	The classifier is a collection of Ada-Boost learners that were trained on a variety of balanced bootstrap examples. Random Under-Sampling is used to achieve the balancing.
Balanced Bagging	An extra balance Bagging classifier. where Bagging is implemented with an additional step to balance the training set at the appropriate moment using a provided sampler.
Balanced Random Forest	A balanced random forest randomly under-samples each bootstrap sample to balance it.

[2] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html

5.4. Cross validation technique

Each algorithm will be applied with ten folds of cross-validation, where the data is first divided into k segments or folds of equal size; following that, k iterations of training and validation are carried out, with each iteration holding out a different fold of the data for validation while utilizing the remaining (k-1) folds for learning.

6. Results and discussion

Figure.1 illustrates the model metrics resulting from applying the Ada-Boost classifier on ELFF method-level datasets, which are (except for roc_auc) way below random classifier values. The zero value appears a lot for all model metrics, which is the worst value for a model metric. Recalling from Subsection 5.2 that a low precision value implies a high false positive rate and a low recall value implies a high false negative rate, the Ada-Boost classifier identifies non-defective methods as defective and identifies defective methods as non-defective with high rates.

In Figure. 2, the model metrics resulting from applying the Gradient Boost classifier are shown. Despite the fact that the results are slightly better than the previous algorithm, they are still way below the random classifier.

Bagging classifier application to ELFF method-level datasets results appear in Figure. 3, where the model metrics values are getting better than two previous algorithms (except for roc_auc), but still not fair results.

On the other hand, Figure. 4 shows results for the Random Forest algorithm, where precision values got better than previous classifiers while recall did not get noticeably better.

Moving to the second approach in this research, which is using the balanced versions of the ensemble learning algorithms in the Sci-kit-learn Python library, Figure. 5 shows the model metrics values of Random Under-Sampling Boost, where recall values noticeably increased and exceeded precision values, meaning that false negative values dropped (the number of defective methods classified as non-defective), indicating an enhancement in the model's ability to identify defective methods as defective.

Results of applying the Balanced-Bagging classifier are shown in Figure .6, The low values of precision achieved by this classifier indicate a high false positive value. On the other hand, recall values increased above the random classifier. Also, the model's ability to distinguish between two classes (defective and non-defective instances) increased as the values of roc_auc increased.

The third algorithm in the balanced ensemble learning algorithms used in this paper is the Easy Ensemble classifier, whose results are shown in Figure. 7, where fair values for recall and roc_auc appeared but with very low values for precision.

Finally, Figure .8 shows the values of applying a Balanced Random Forest classifier on ELFF datasets, where the best results were achieved among all algorithms used in this paper in terms of recall and roc_auc, which means an increase in the model ability to identify defective instances as defective and an increase in the model ability to distinguish between defective and non-defective methods.

Figure. 9 summarizes the performance of the eight ensemble learning algorithms used in this paper.It shows that the non-balanced versions of the ensemble learning algorithms achieved the worst performance regarding recall values, but on the other hand, they achieved better precision results than the balanced ensemble learning versions. It also shows that precision values in the four balanced ensemble learning algorithms dropped dramatically while recall values increased,which might be caused by re-sampling techniques implemented by the balanced ensemble learning algorithms where instances from the majority class (non-defective methods) were deleted, causing a balance in the datasets. Results in Figure. 9 indicate that the roc_auc values were roughly close in both balanced and unbalanced versions of the ensemble learning algorithms. Low precision value in practice means that testing resources will be wasted on methods unlikely to have faults; on the other hand, low recall value means that a high number of defect-prone modules will go undiscovered, implying low software quality.

7. Conclusion

Method-level software defect prediction was an open challenge due to the lack of reliable datasets. The ELFF datasets were created specifically to fill in that gap in that area of research. In this paper, the SDP model was built on the ELFF datasets using two ensemble learning approaches: ensemble learning algorithms (Ada-Boost, Gradient Boost, Bagging, and Random Forest) and balanced ensemble learning algorithms (Random under-sampling Boost, Easy Ensemble, Balanced Bagging, and Balanced Random Forest).

1. As for the first contribution (investigating the use of ELFF method level datasets to perform software defect prediction using the ELLF software metrics):
   The first four ensemble algorithms achieved low values for all model metrics, especially recall and roc_auc, which were mostly below the random classifier.
   Implementing the balanced versions of the ensemble learning algorithms enhanced recall and roc_auc values noticeably but caused the value of precision to severely drop, which is imputed to the re-sampling strategy implemented by the balanced versions of the ensemble learning algorithms, where instances from the majority (non-defective methods) were discarded, causing information related to them to be lost.
   The low precision value means wasted resources on unlikely defective instances, and the low recall value implies undiscovered defective methods.
   So if the priority of the SDP model built on the ELFF datasets is to identify defective methods rather than save resources, then using Balanced Random Forest algorithm will be effective due to its fair recall and roc_auc values, but if saving testing resources is of high importance, then using that prediction model may not be practical due to its low precision value.

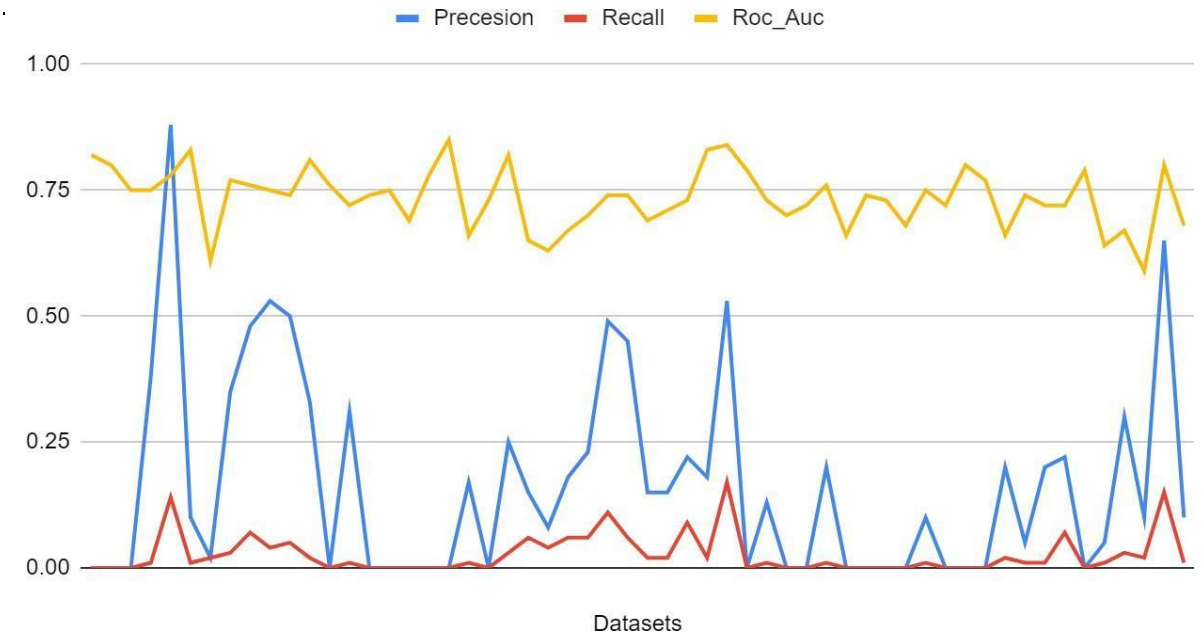


Figure.1 : Model evaluation metrics values of applying Ada-Boost classifier to ELFF Datasets

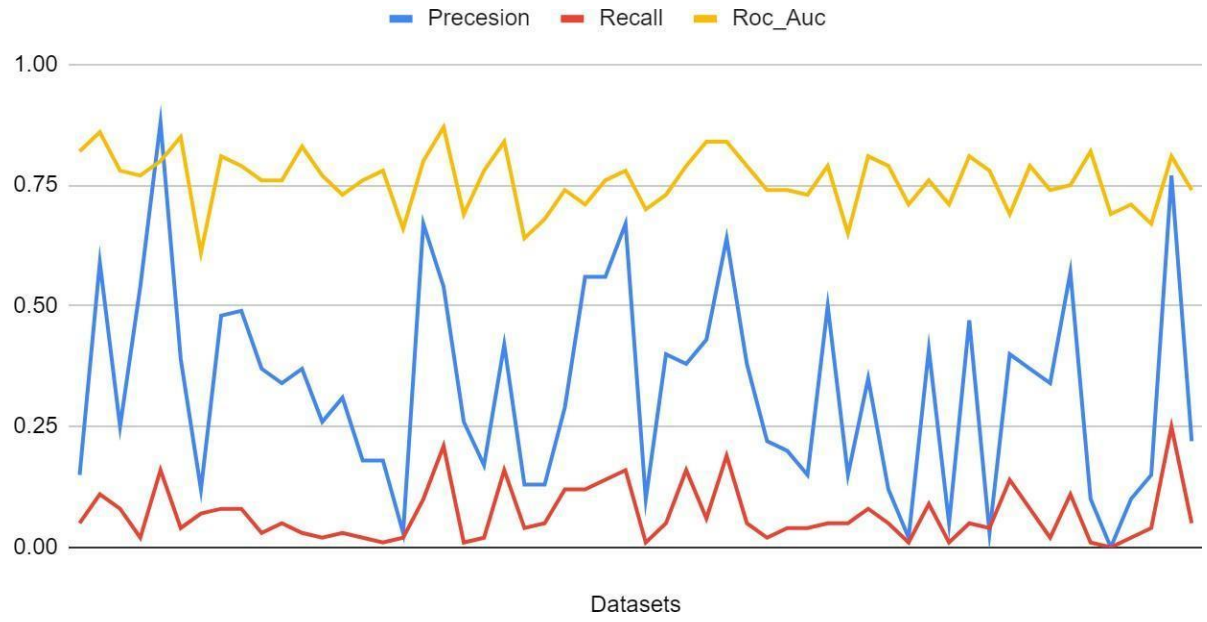


Figure.2: Model evaluation metrics values of applying Gradient-Boost classifier to ELFF Datasets

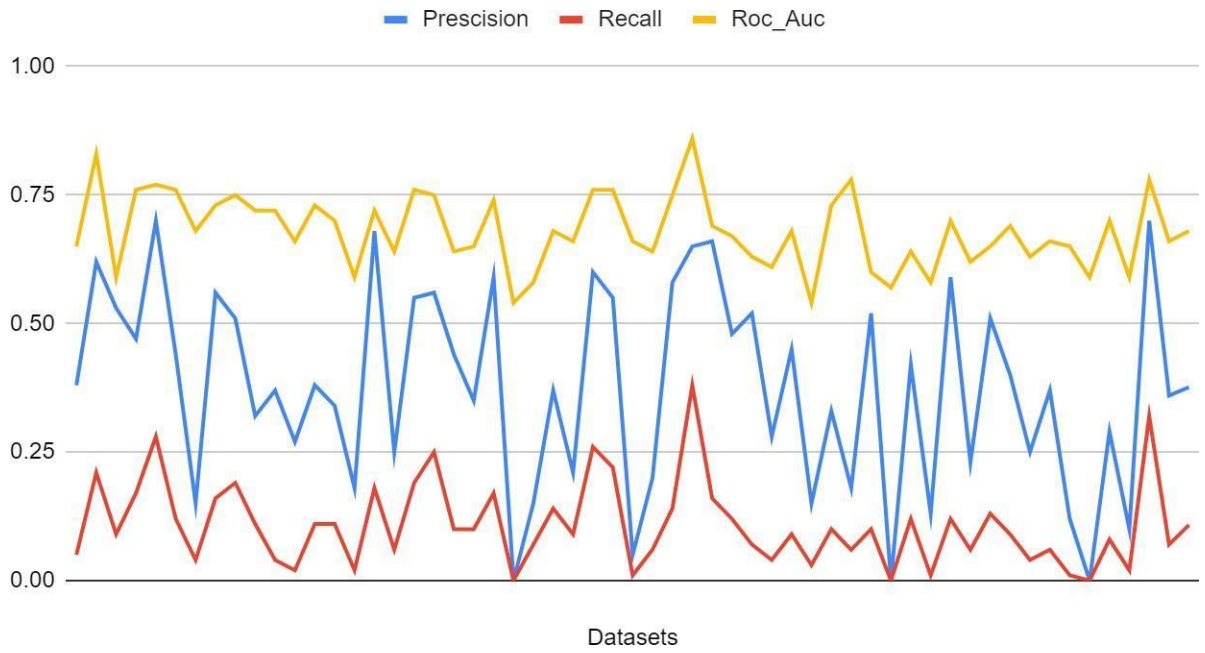


Figure.3: Model evaluation metrics values of applying Bagging classifier to ELFF Datasets

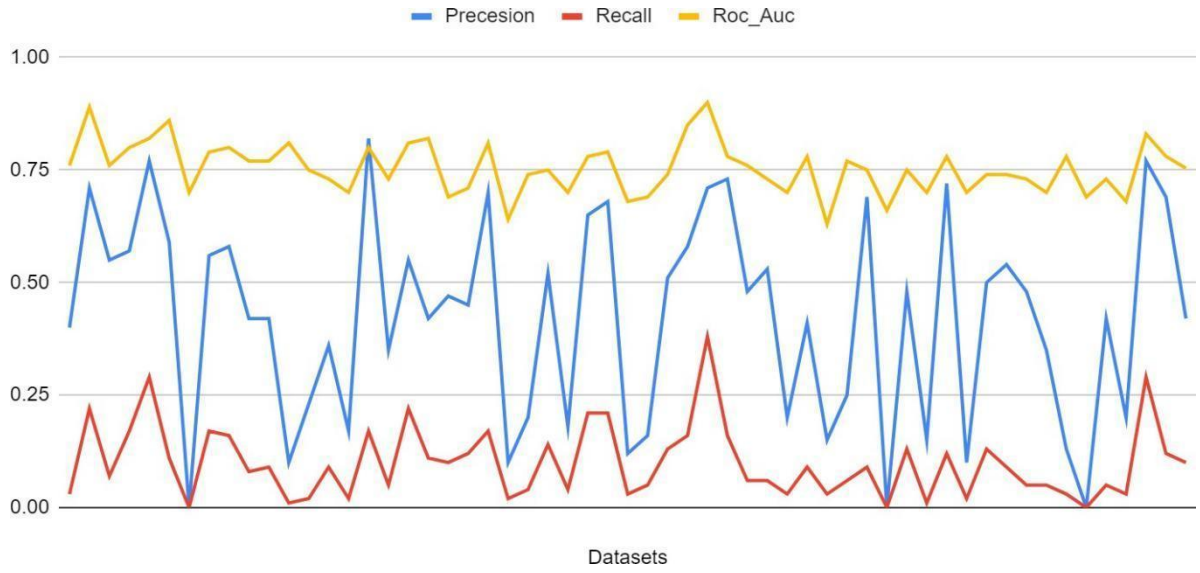


Figure.4: Model evaluation metrics values of applying Random Forest classifier to ELFF Datasets

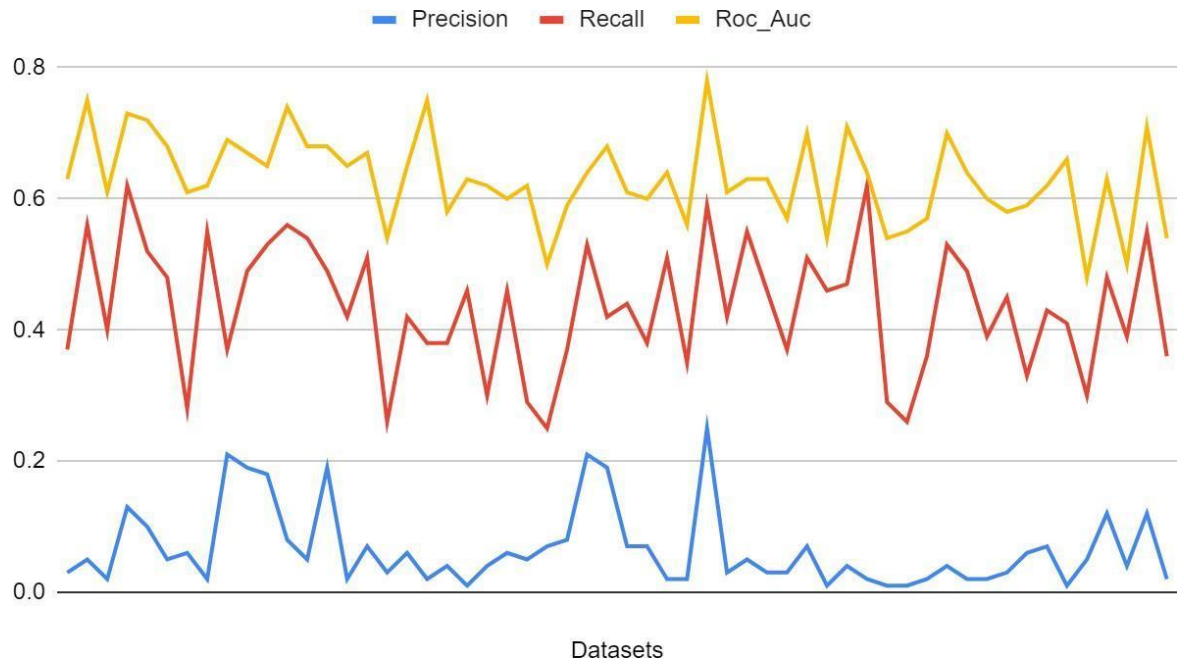


Figure.5: Model evaluation metrics values of applying RUS Boost classifier to ELFF Datasets

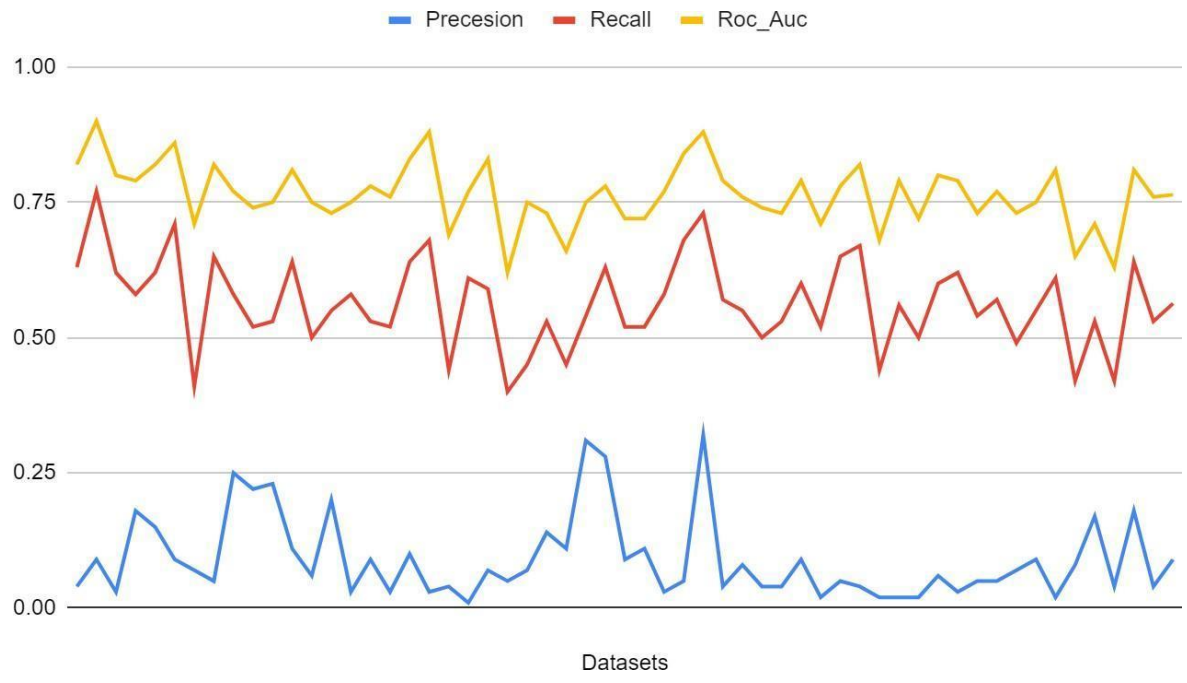


Figure.6: Model evaluation metrics values of applying Balanced Bagging classifier to ELFF Datasets

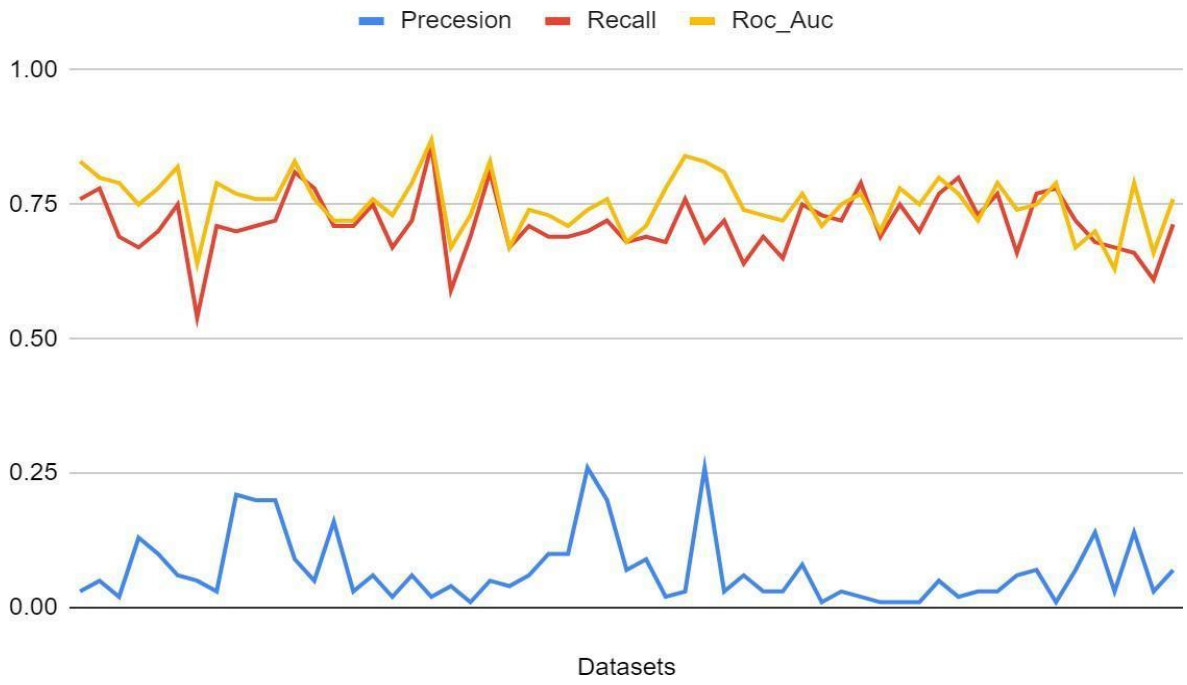


Figure.7: Model evaluation metrics values of applying Easy Ensemble classifier to ELFF Datasets

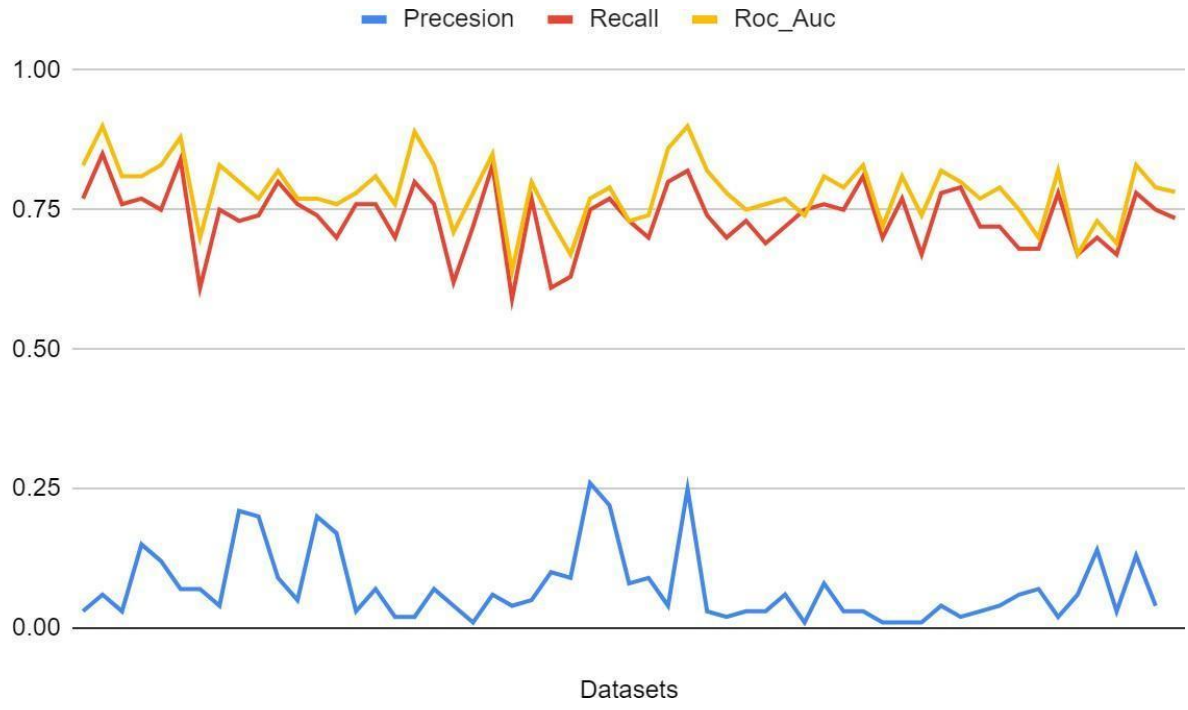
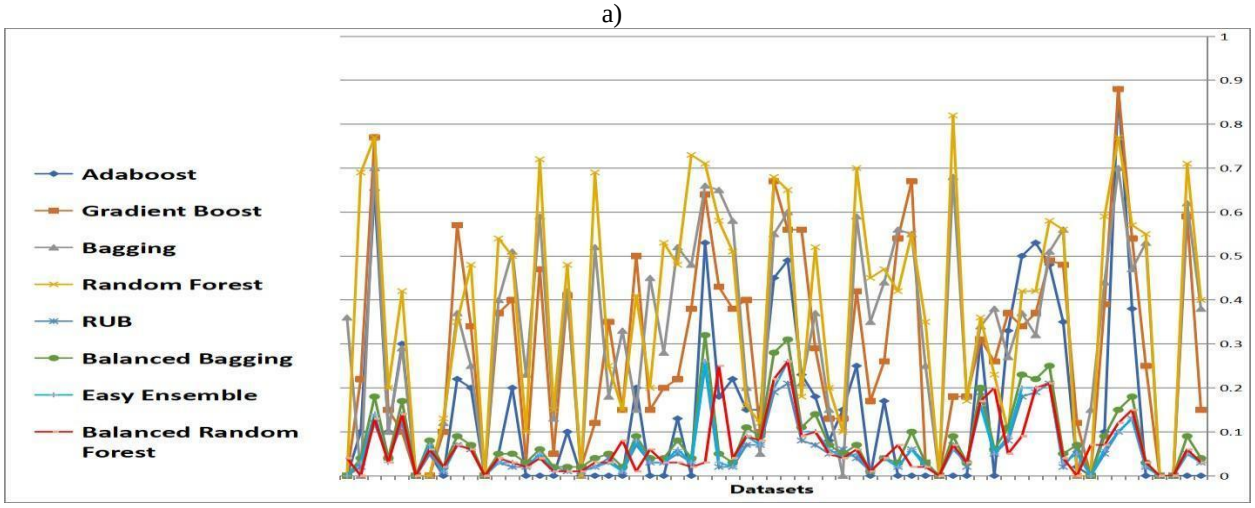
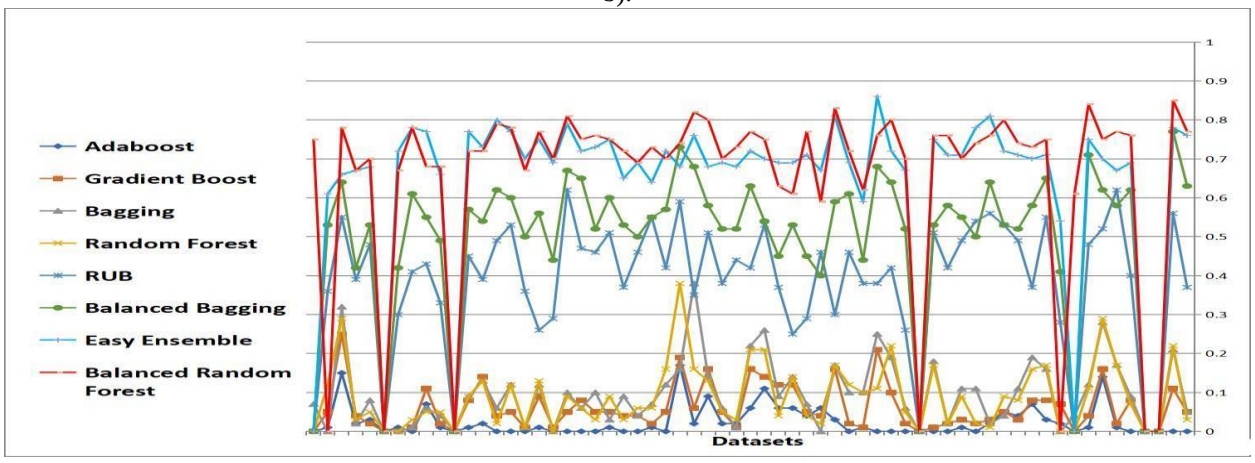


Figure.8: Model evaluation metrics values of applying Balanced Random Forest classifier to ELFF Datasets

a)



b).



c).

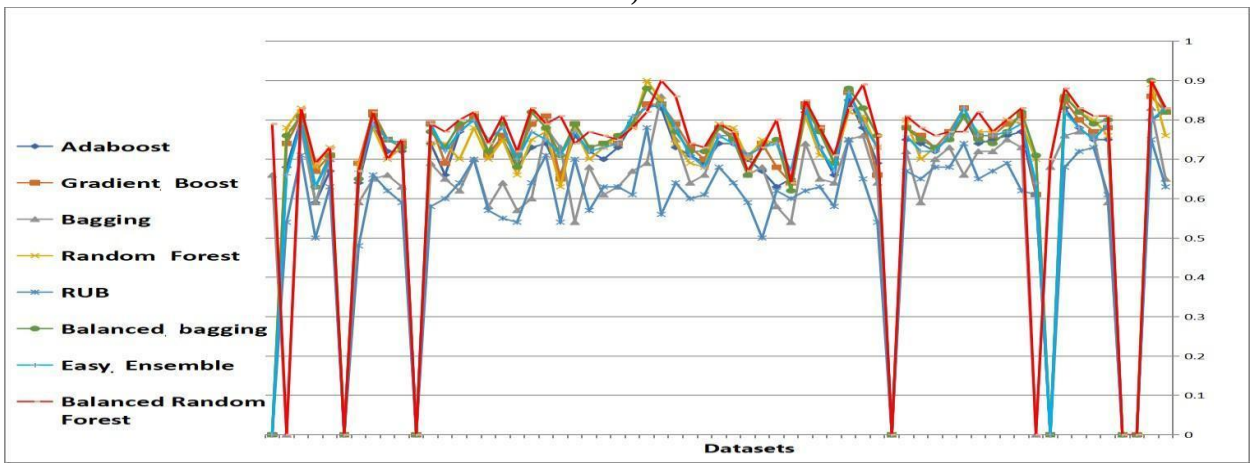


Figure. 9: Model evaluation metrics of applying ensemble learning algorithms t to ELFF datasets
a).Precision , b).Recall , c) Roc_Auc

2. As for the second contribution (investigating the impact of eight ensemble learning algorithms on the imbalance of the datasets):
   Ensemble learning algorithms (Ada-Boost, Gradient-Boost, Bagging, and Random Forest) failed to achieve reliable results for all model metrics, implying that ensemble learning algorithms were not effective in handling highly imbalanced datasets.
   On the other hand, the balanced versions of the algorithms (Random Under-Sampling Boost, Easy Ensemble, Balanced Bagging, and Balanced Random Forest) enhanced the recall and roc_auc values due to their built-in re-sampling strategies but caused the precision value to severely drop.

8. Future work

In this paper, a defect prediction model on the ELFF datasets was investigated using 10 folds of cross validation to predict defects in the same project (within project defect prediction).
Cross-project defect prediction (the prediction of defects in a project based on data from other projects) is an important field of research in the area of SDP.
Hence, one of the future plans is to evaluate cross-project defect prediction (CPDP) on the ELFF datasets using ensemble learning techniques, which means instead of performing training and testing on the same project data, the prediction model based on one dataset will be tested on another project dataset. As there are in the ELFF dataset 69 projects of different versions, the eldest project version can be used as a training set, with the newer versions as testing set.

9. Availability of data and materials

Artifacts for this paper can be downloaded at :
https://github.com/challengerofsmile/Elff.git .
ELFF datasets can be downloaded at :
https://github.com/tjshippey/ESEM2016

References

[1] Pataricza, A., G¨onczy, L., Brancati, F., Moreira, F., Silva, N., Esposito, R., Bondavalli, A., Esper, A.: Cost estimation for independent systems verification and validation. Certifications of Critical Systems-The CECRIS Experience. River Publishers, 117 (2017) .
[2] Hata, H., Mizuno, O., Kikuno, T.: Bug prediction based on fine-grained module histories. In: 2012 34th International Conference on Software Engineering (ICSE), (2012), pp. 200–210 , IEEE
[3] Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering 38(6), (2011), 1276–1304
[4] Pascarella, L., Palomba, F., Bacchelli, A.: Re-evaluating method-level bug prediction. In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), (2018) , pp. 592–601 , IEEE
[5] Shippey, T., Hall, T., Counsell, S., Bowes, D.: So you need more method level datasets for your

software defect prediction? voil`a! In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, (2016) , pp. 1–6
[6] Provost, F.: Machine learning from imbalanced data sets 101. In: Proceedings of the AAAI'2000 Workshop on Imbalanced Data Sets, vol. 68, (2000) , pp. 1–3 . AAAI Press
[7] Spelmen, V.S., Porkodi, R.: A review on handling imbalanced data. In: 2018 International Conference on Current Trends Towards Converging Technologies (ICCTCT), (2018) , pp. 1–11 . IEEE
[8] Odejide, B.J., Bajeh, A.O., Balogun, A.O., Alanamu, Z.O., Adewole, K.S., Akintola, A.G., Salihu, S.A., Usman-Hamza, F.E., Mojeed, H.A.: An empirical study on data sampling methods in addressing class imbalance problem in software defect prediction. In: Computer Science On- line Conference, (2022), pp. 594–610 . Springer
[9] Bennin, K.E., Tahir, A., MacDonell, S.G., B¨orstler, J.: An empirical study on the effectiveness of data resampling approaches for cross-project software defect prediction. IET Software 16(2), (2022) , 185–199
[10] Iqbal, A., Aftab, S., Matloob, F.: Performance analysis of resampling techniques on class imbalance issue in software defect prediction. Int. J. Inf. Technol. Comput. Sci 11(11), (2019) , 44–53
[11] Polikar, R.: Ensemble based systems in decision making. IEEE Circuits and systems magazine 6(3), (2006) , 21–45
[12] Giger, E., D'Ambros, M., Pinzger, M., Gall, H.C.: Method-level bug prediction. In: Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, (2012) , pp. 171–180 . IEEE
[13] Niedermayr, R., R¨ohm, T., Wagner, S.: Too trivial to test? an inverse view on defect prediction to identify methods with low fault risk. PeerJ Computer Science 5, (2019) , 187
[14] Moustafa, S., ElNainay, M.Y., El Makky, N., Abougabal, M.S.: Software bug prediction using weighted majority voting techniques. Alexandria engineering journal 57(4), (2018) , 2763–2774 .
[15] Yucalar, F., Ozcift, A., Borandag, E., Kilinc, D.: Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability. Engineering Science and Technology, an International Journal 23(4),(2020), 938–950 .
[16] Mehta, S., Patnaik, K.S.: Improved prediction of software defects using ensemble machine learning techniques. Neural Computing and Applications 33(16), (2021) , 10551–10562
[17] Saheed, Y.K., Longe, O., Baba, U.A., Rakshit, S., Vajjhala, N.R.: An ensemble learning approach for software defect prediction in developing quality software product. In: International Conference on Advances in Computing and Data Sciences, (2021) , pp. 317–326 . Springer
[18] Balogun, A.O., Lafenwa-Balogun, F.B., Mojeed, H.A., Adeyemo, V.E., Akande, O.N., Akintola, A.G., Bajeh, A.O., Usman-Hamza, F.E.: Smotebased homogeneous ensemble methods for software defect prediction. In: International Conference on Computational Science and Its Applications, (2020), pp. 615–631 . Springer
[19] Santos, M.S., Soares, J.P., Abreu, P.H., Araujo, H., Santos, J.: Crossvalidation for imbalanced datasets: avoiding overoptimistic and overfitting approaches [research frontier]. ieee ComputatioNal iNtelligeNCe magaziNe 13(4), (2018) , 59–76
[20] Abuqaddom, I., Hudaib, A.: Cost-sensitive learner on hybrid smoteensemble approach to predict software defects. In: Proceedings of the Computational Methods in Systems and Software,

(2018) , pp. 12–21 . Springer
[21] Aljamaan, H., Alazba, A.: Software defect prediction using tree-based ensembles. In: Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, (2020) , pp. 1–10
[22] Saifan, A.A., Abu-wardih, L.: Software defect prediction based on feature subset selection and ensemble classification. ECTI Transactions on Computer and Information Technology (ECTI-CIT) 14(2), (2020) , 213–228
[23] Khan, M.Z.: Hybrid ensemble learning technique for software defect prediction. International Journal of Modern Education & Computer Science (2020), (1)12
[24] Malhotra, R., Jain, J.: Predicting defects in object-oriented software using cost-sensitive classification. In: IOP Conference Series: Materials Science and Engineering, vol. 1022, (2021), p. 012112 . IOP Publishing
[25] Bennin, K.E., Keung, J.W., Monden, A.: On the relative value of data resampling approaches for software defect prediction. Empirical Software Engineering 24(2), (2019), 602–636
[26] Alsaeedi, A., Khan, M.Z.: Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. Journal of Software Engineering and Applications 12(5), (2019), 85–100
[27] Shatnawi, R., Al-Sharif, Z.: A guided oversampling technique to improve the prediction of software fault-proneness for imbalanced data. International Journal of Knowledge Engineering and Data Mining 2(2-3)),2012), 200-214
[28] Khuat, T.T., Le, M.H.: Ensemble learning for software fault prediction problem with imbalanced data. International Journal of Electrical and Computer Engineering 9(4),(2019), 3241
[29] Alsawalqah, H., Faris, H., Aljarah, I., Alnemer, L., Alhindawi, N.: Hybrid smote-ensemble approach for software defect prediction. In: Computer Science On-line Conference,(2017), pp. 355–366 . Springer
[30] Yang, H., Li, M.: Software defect prediction based on smote-tomek and xgBoost. In: International Conference on Bio-Inspired Computing: Theories and Applications,(2021), pp. 12–31 . Springer
[31] Bowes, D., Counsell, S., Hall, T., Petric, J., Shippey, T.: Getting defect prediction into industrial practice: the elff tool. In: 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),(2017), pp. 44–47 . IEEE
[32] Felix, E.A., Lee, S.P.: Predicting the number of defects in a new software version. PloS one 15(3), 0229131 (2020).
[33] Felix, E.A., Lee, S.P.: Impact of defect velocity at class level. In: 2017 International Conference on Robotics and Automation Sciences (ICRAS), (2017), pp. 182–188 . IEEE
[34] Yang, F., Huang, Y., Xu, H., Xiao, P., Zheng, W.: Fine-grained software defect prediction based on the method-call sequence. Computational Intelligence and Neuroscience 2022 (2022).
[35] Zheng, X., Li, Y.-F., Gao, H., Hua, Y., Qi, G.: Towards balanced defect prediction with better information propagation. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, (2021), pp. 759–767.
[36] Al Qutaish, R.E., Abran, A.: An analysis of the design and definitions of halstead's metrics. In: 15th Int. Workshop on Software Measurement (IWSM'2005). Shaker-Verlag, (2005), pp. 337–352