An Extensive Analysis of Machine Learning Based Boosting Algorithms for Software Maintainability Prediction

Shikha Gupta*, Anuradha Chug

University School of Information, Communication & Technology, Guru Gobind Singh Indraprastha University, Sector - 16C, Dwarka, New Delhi - 110078 (India)

Received 12 February 2020 | Accepted 16 February 2021 | Published 8 October 2021

Abstract

Software Maintainability is an indispensable factor to acclaim for the quality of particular software. It describes the ease to perform several maintenance activities to make a software adaptable to the modified environment. The availability & growing popularity of a wide range of Machine Learning (ML) algorithms for data analysis further provides the motivation for predicting this maintainability. However, an extensive analysis & comparison of various ML based Boosting Algorithms (BAs) for Software Maintainability Prediction (SMP) has not been made yet. Therefore, the current study analyzes and compares five different BAs, i.e., AdaBoost, GBM, XGB, LightGBM, and CatBoost, for SMP using open-source datasets. Performance of the propounded prediction models has been evaluated using Root Mean Square Error (RMSE), Mean Magnitude of Relative Error (MMRE), Pred(0.25), Pred(0.30), & Pred(0.75) as prediction accuracy measures followed by a non-parametric statistical test and a post hoc analysis to account for the differences in the performances of various BAs. Based on the residual errors obtained, it was observed that GBM is the best performer, followed by LightGBM for RMSE, whereas, in the case of MMRE, XGB performed the best for six out of the seven datasets, i.e., for 85.71% of the total datasets by providing minimum values for MMRE, ranging from 0.90 to 3.82. Further, on applying the statistical test and on performing the post hoc analysis, it was found that significant differences exist in the performance of different BAs and, XGB and CatBoost outperformed all other BAs for MMRE. Lastly, a comparison of BAs with four other ML algorithms has also been made to bring out BAs superiority over other algorithms. This study would open new doors for the software developers for carrying out comparatively more precise predictions well in time and hence reduce the overall maintenance costs.

Keywords

Boosting Algorithms, Feature Selection, Machine Learning, Software Maintainability Prediction, Software Metrics.

DOI: 10.9781/ijimai.2021.10.002

I. Introduction

SOFTWARE Maintenance, as described in the IEEE Standard for Software Maintenance [1], refers to any modification in a software product after its delivery for improving the performance or any other attribute, for correcting the faults or for adapting the product according to the modified environment. However, software maintenance is not an easy activity because of the complexity that exists in the maintenance behavior of various software systems. Also, a handsome amount of cost is incurred while maintaining software since software maintenance is a high-priced affair. A significant proportion of the comprehensive cost of software during the Software Development Life Cycle (SDLC) is spent in the maintenance phase alone since the cost of maintenance keeps on accumulating with each phase of SDLC. It has been observed that only 30-40% of the resources, including money, time, and effort, are utilized in the development phase, whereas the remaining 60-70% is used for the maintenance activities [1].

* Corresponding author.
E-mail address: shikha.usict.140164@ipu.ac.in

There exists a detailed standard for software quality known as ISO/IEC 25010:2011. It describes eight product quality characteristics, where each characteristic further comprises various sub-related characteristics [2]. Fig. 1 depicts these eight quality characteristics, along with the sub characteristics. Of all the quality characteristics, maintainability is considered for evaluation in the current study since it is one of the most significant characteristics.

In recent times, any software's quality has come out as an essential parameter to account for the software's success. In turn, software quality depends on two main types of attributes: categorized into internal and external categories. Internal attributes like coupling, cohesion, abstraction, inheritance, etc. are directly-measured from the source code during the initial stages of SDLC at the developer level and are hidden from the users. However, external attributes such as durability, understandability, robustness, modifiability, analyzability, etc. are visible to the users and are, in turn, measured indirectly with the help of different internal attributes [3]. The external attributes may also be measured through developers' opinions who write the source code for the open-source software by organizing surveys. However, such surveys involve high costs and are also very time consuming and

may produce biased opinions due to the subjectiveness involved in the external quality attributes. Contrarily, measurement of internal quality attributes using Object-Oriented (OO) metric suites has been validated by many researchers for predicting maintainability keeping in view the relationship that exists between the OO metrics & maintainability [4]–[9]. Hence, the current study also uses these OO metrics for Software Maintainability Prediction (SMP).

Software maintainability these days has become one of the essential external attributes of software, which further forms a basis of research for many researchers working in the fields related to software engineering. Software maintainability can be described as the extent to which a particular software system can be changed concerning the number of Lines of Code (LOC). The researchers' fundamental goal is to develop such models for the prediction that are proficient in predicting any software's maintainability accurately and well in advance. This further ensures optimum utilization of resources, including not only money but also the effort and time put in by the development team. Further, the prediction is not the only goal here, but the predictions made should also have high prediction accuracies with the least possible precision errors. Usually, predictions are made with historical data of particular software for which the prediction model is being developed, including both internal and external attributes. A qualitative description of correlation among the internal & external attributes is also found for SMP [3].

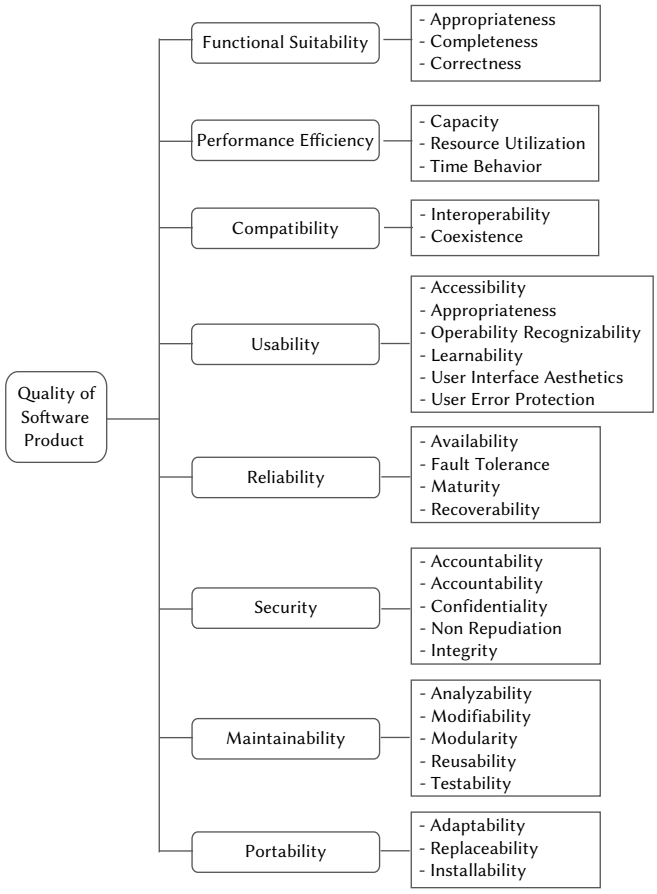

Fig. 1. Model for quality of software product.

In order to estimate the internal quality attributes of software, different metrics have been used such as Afferent Couplings (Ca), Coupling between Object classes (CBO), Efferent Couplings (Ce), & Inheritance Coupling (IC) for coupling; Cohesion among the Methods of a class (CAM), Lack of Cohesion in Methods (LCOM), & Lack of Cohesion among the Methods of a particular Class (LCOM3) for cohesion; Depth of Inheritance Tree (DIT), Number of Children (NOC), & Measure of Functional Abstraction (MFA) for inheritance; Weighted Methods per Class (WMC), & Average Method Complexity (AMC) for complexity; Lines of Code (LOC), & Number of Public Methods (NPM) for size; Response for a Class (RFC) for cardinality; Data Access Metric (DAM) for encapsulation; Measure of Aggregation (MOA) for composition, etc. as compiled in [10], [11]. Software metrics are used to monitor and improve various processes & products in software engineering. These metrics measure various facets of software, such as the design documents or the source code. However, there exist different software metrics based on whether the paradigm is procedural or OO. As per the existing literature, software systems have been analyzed from three perspectives, i.e., the architecture of the system, its design, and the code for SMP [4]. However, out of these, code-level analysis for SMP is the most widely used perspective.

A significant breakthrough for the software industry comes with the advent of Machine Learning (ML). ML [12] is a discipline of artificial intelligence that pertains to the automatic learning capability of different systems and the improvisation of the efficiency based on their past experiences without any explicit programming or learning. The primary focus of ML is developing such programs capable of accessing the data and utilizing it to learn for themselves. The ML process initiates with some data or observations to identify certain patterns in that data, which can further be utilized to make efficient decisions in the future based on the initial data provided. The most important goal of ML is to make the computers capable of automatically learning without human beings' intervention or any kind of external assistance and act accordingly. A wide variety of ML algorithms are currently available for use, broadly classified into two major categories, i.e., supervised (classification & regression) and unsupervised (clustering and association) ML algorithms. Other categories of ML algorithms include semi-supervised and reinforcement learning. Nowadays, ML finds its applications in almost every sphere of life, including web search, computational biology, finance, e-commerce, software engineering, robotics, social networks, debugging, disease diagnosis, stock analysis, marketing analysis, and prediction, etc. Several ML algorithms have been implemented in the fields mentioned above.

As an example, considering the stock analysis field, Sharma et al. [13] in 2018 analyzed ten different supervised classifiers, including logistic regression, C4.5, random forest, etc. for mining of stock data using ICICI bank's data with logistic regression outperforming the other classifiers; Zhong and Enke [14], in 2019, presented a hybrid of deep neural networks with traditional Artificial Neural Networks (ANN) for predicting the return direction for the stock market on a daily basis considering 60 economic & financial features. Rasekhschaffe and Jones [15] in the same year described the primary concepts and the use of ML algorithms in stock selection along with the use of few ensemble models for forecasting stock returns while minimizing the risk of over-fitting. Further, considering the disease diagnosis field, Kaur and Sharma [16], in 2019, extensively reviewed different supervised & nature-inspired ML techniques for mining and analyzing the diagnosis of various psychological disorders using a systematic 3-D search space methodology covering the diagnosis, the disorder & the classification algorithms; and Reddy et al. [17], in 2020, proposed an effective hybrid of adaptive Genetic Algorithm (GA) & fuzzy logic approach (AGAFL) to help the doctors in the early and timely diagnosis of the heart diseases. Again in 2020, Sharma and Kaur [18] conducted a detailed review of the role of several meta-heuristic algorithms based on nature-inspired rules in solving the problem of selecting relevant features for a better classification in different fields; disease diagnosis being the most assessed area. Afterward, in the field of finance, Xiaomeng and Shuliang [19] in 2019, came up with an

improved & efficient ML algorithm, i.e., MLIA for the prediction of credit risk in the market of internet finance. Ghoddusi et al. [20] in the same year performed a comprehensive review of various applications of ML in the area of finance or energy economics, including areas like demand forecasting, data processing, risk management, etc., where support vector machines, ANN, and GA stood out to be the most widely used techniques in the concerned field.

Coming back to the field of software maintainability, over the past few years, researchers have developed different ML, evolutionary, & statistical ensemble models for SMP based on the code level metrics intended to predict software maintainability in the most accurate possible manner. Some of the individual ML models developed in the past include General Regression Neural Network (GRNN), Multilayer Perceptron (MLP) [21], Feed Forward 3-Layer Back Propagation Network (FF3LBPN) & Group Method of Data Handling (GMDH) [7]. Models based on nature-inspired algorithms include Evolutionary Algorithms (EA) [22], GA, Functional Link ANN (FLANN), Clonal Selection Algorithm (CSA) & Particle Swarm Optimization (PSO) [23]. Further, the ensemble models developed to date include bagging [8], [24] & boosting, particularly Adaptive Boosting (AdaBoost) [24]. Detailed information about existing models for SMP is provided in the related work section.

Further, ML based boosting techniques have already been explored & implemented successfully in various fields of software engineering comprising software defect prediction [25], software reliability modeling [26] & software fault proneness [27]. However, it is evident from the past studies that none of the researchers has made extensive use of existing Boosting Algorithms (BAs) for SMP apart from one particular BA, i.e., AdaBoost, which still finds a mention in one or two studies [24] for predicting maintainability. Therefore, this study endeavors to predict software maintainability using various BAs and perform an extensive analysis of these BAs for SMP. Boosting is a homogeneous ML ensemble technique proposed by Freund in 1995 [28]. In boosting, an ensemble of classifiers is formed incrementally on adding one classifier at a point in time utilizing the weighted averages so that the base estimators or the weak learners can be converted into strong learners before generating the final output. Unlike other ML algorithms, BAs aim to improvise the prediction capability by training a series of weak learners, each of which compensates for the weaknesses in its preceding learner. BAs are usually intended to reduce the variance and single estimators' bias resulting in a much more stable model. Various BAs are available today that can be used for solving complex and data-driven real-world problems. A significant advantage of using BAs is that these algorithms provide immense powers to the basic ML algorithms, such as Decision Tree (DT), Random Forest (RF), regression, MLP, etc. to improve the prediction accuracy outperforming the basic models. This happens as BAs combine several weak hypotheses of the base estimator that are moderately correct with an objective to derive a notably accurate hypothesis. BAs take several rounds of operation, after which a noteworthy improvement in the accuracy of the training data is achieved. In every round of BAs, samples in the training set are re-weighed, & the base estimator is run on the re-weighted training samples. The BAs main motive is to drive the focus of the weak base estimator towards the error-prone samples. This ultimately leads to the final hypothesis, which is the weak hypotheses' weighted vote. Subsequently, the BAs major strengths include their easy interpretability, availability of feature selection implicitly, resilience towards over-fitting, and strong predictive capability.

Not only this, another significant motivation behind choosing boosting algorithms to conduct this study comes from the effectiveness of the tree boosting that fits the additive tree models having a high ability of representation [29]. This is possible due to the adaptive neighborhood's property, which enables tree boosting use varying degrees for flexibility of different regions in the input space. Hence, it is robust to the dimensionality problem as it performs the feature selection automatically by capturing interactions that are high in order without getting broken. Further, suppose we talk in particular about eXtreme Gradient Boosting (XGB) [29], [30]. In that case, it will learn better structures of trees since these structures determine the neighborhoods & are highly adaptive to the data. XGB uses smart penalization for an individual tree, which can then have a different number of terminal nodes apart from shrinkage. The benefit of using penalization lies in the fact that all the leaves' weight is not shrunk by a common factor. Instead, the weights estimated through fewer pieces of evidence in the data are shrunk even more heavily. Additionally, XGB uses Newton boosting, unlike gradient boosting, & also includes a parameter for randomization to further de-correlate individual trees, which results in the reduction of the overall variance. Also, XGB has a better learning capability. It uses high order approximation at each iteration of the optimization problem & considers the tradeoff between the bias & the variance while fitting the model.

The current study utilizes open-source datasets to analyze various BAs for SMP. The underlying idea behind the selection of open-source datasets for this study was the easy availability of these datasets through various online platforms such as SourceForge & GitHub and the need for generalization and validation of the proposed models for other software in the industry. The study is conducted on a set of seven empirically collected open-source datasets, namely, Abdera, Ivy, jEdit, jTDS, Log4j, Poi, & Rave. A collection of seventeen OO software metrics has been used as the independent variables. In contrast, the maintenance effort (dependent variable) used here is 'Change', which is equal to the number of lines that have been changed per class in the maintenance history. Original datasets are pre-processed to remove those rows where maintenance effort is equal to zero. Feature scaling using MinMaxScaler and feature selection using the Recursive Feature Elimination (RFE) technique is also performed for improving the quality of data. Five different BAs, i.e., AdaBoost, Gradient Boosting (GBM), XGB, LightGBM, and Categorical Boosting (CatBoost), are selected for developing various prediction models for each dataset.

Models are validated using the ten-fold cross-validation technique, and the capability of these models is assessed using Root Mean Square Error (RMSE), Mean Magnitude of Relative Error (MMRE), Pred(0.25), Pred(0.30) & Pred(0.75) taken as the prediction accuracy measures. Friedman test to rank the performance of different BAs used in the study and the Nemenyi test for conducting the post hoc analysis are also performed based on the MMRE. Lastly, a comparison of results achieved on applying the BAs with the results obtained on applying four other ML algorithms (apart from the BAs), viz., DT, MLP, bagging, and Elastic - Net (EN) is also made. Results show that BAs can effectively be applied for SMP, which opens new ways for the researchers to explore these algorithms further. This study's worth lies in the fact that predicting maintainability has become a crucial point of consideration for the software developers throughout the SDLC while developing any software. Also, the software has become a necessity these days since many tasks are becoming automated each day, and this conversion requires some software to be developed. Therefore, the software industry's importance and, in turn, the software is growing leaps and bounds with each passing day. Since the software is being developed, it needs to be maintained also, but as discussed earlier, a handsome amount of cost is required to be spent in the maintenance phase. Thus, some techniques or models are required for predicting software maintainability sufficiently in advance. The current study fulfills this requirement by providing several such models using ML based BAs for a precise prediction of maintainability in good time to help the software developers utilize the resources, such as the money, time & effort judiciously. This would further bring down the

maintenance costs associated with any software development process to a considerable extent.

The primary objectives of the current study can be summarized as Research Questions (RQs).

RQ1: Whether BAs can be applied for SMP?

RQ2: Which BA performs the best amongst different BAs based on various prediction accuracy measures for different open-source datasets?

RQ3: What is the comparative performance of various BAs during post hoc analysis when MMRE is taken as an accuracy measure?

RQ4: What is the comparison between the results obtained on applying various ML algorithms (other than the BAs) and the results obtained on implementing BAs?

The remaining paper that follows is organized as - Section II describes the *related work* carried out in the current study field. Section III discusses the datasets, independent variables, & the dependent variables that have been used in the current study. Section IV describes the complete *research methodology*, including pre-processing of datasets, feature scaling & feature selection techniques, description of BAs used, cross-validation technique, prediction accuracy measures, statistical test, and post hoc analysis. *Results and discussions* are described in Section V, whereas Section VI highlights the *threats to validity*. Lastly, Section VII closes the paper with *conclusions & future directions*.

II. Related Work

SMP has become a principal aspect to ascertain any software's quality in the industry over the last few years. Predicting this maintainability in initial stages of development is the need of the hour for efficient and optimum development of any software system. Over the years, substantial research is already being done in the field of SMP by various researchers. They have developed several prediction models using different ML, hybrid, nature-inspired, & other suitable techniques. A summary of these prediction models' details, including the information regarding the types of datasets, metrics suites, ML techniques, validation methods, and the prediction accuracy measures used to compare the performances of the developed prediction models, is provided in Table I. The observed accuracy measures prove that a strong relationship exists between the OO metrics & the maintainability.

Li and Henry [31], in 1993, studied the validation of various OO metrics with the maintenance for the first time using Quality Evaluation System (QUES) & User Interface Management System (UIMS) to prove the existence of a powerful relation among OO metrics and the maintainability. Since then, many researchers have been working in the area of SMP for QUES and UIMS datasets using OO metrics [32]–[38]. Later, Malhotra and Chug [39], in 2012, proposed three ML algorithms, i.e., GMDH, Probabilistic NN (PNN), & GAs, using the Gaussian activation function to predict maintainability & compared their performance with other existing models such as ANN. Results showed that the GMDH model is comparatively more precise & more accurate than the existing models. Again in 2012, Dubey et al. [21] proposed using a robust & adaptive MLP NN model to predict maintainability. MLP, when compared with other models, i.e., WNN & GRNN, was found to be more superior. In another study conducted by Ahmed and Al-Jamini in 2013 [3], fuzzy logic based prediction models, i.e., Mamdani Fuzzy Inference Engine & T-S, were developed & compared for SMP. In comparison, the Mamdani-based prediction model gave the most accurate results of all. In 2014, Malhotra and Chug [7] evaluated the GMDH technique's effectiveness for predicting maintainability by comparing it with the other two techniques, i.e., FF3LBPN & GRNN. It was observed that the GMDH technique performed the best with minimum error & high precision.

In another study, Malhotra and Chug [22] suggested using EAs for SMP using ten-fold cross-validation. The model's performance was analyzed with the help of MRE, MMRE & Pred(q), and later compared with other statistical and ML algorithms. It was found that EAs can effectively predict maintainability with more accuracy and precision as compared to other traditional methods. In 2015, Elish et al. [24] presented three empiric studies for SMP using different homogeneous & heterogeneous ensemble methods. They evaluated and compared three of the heterogeneous ensemble methods to predict maintenance effort, i.e., Weighted-based (WT), Average-based (AVG), and Best in Training-based (BT) ensemble methods. Resultantly, ensemble models came out to be the best when compared to other individual models. All the ensemble and individual models were outperformed by the BT ensemble method. In 2015 only, Kumar et al. [6] suggested using class-level OO software metrics in predicting maintainability with the help of a Neuro-GA for developing the prediction model for QUES and UIMS datasets. Results indicated a successful implementation of Neuro-GA for SMP by generating promising results.

Kumar and Rath [23], in 2016, suggested the use of three Artificial Intelligence (AI) techniques, i.e., FLANN-GA, FLANN-PSO, and FLANN-CSA, to develop models for predicting maintainability along with a few feature reduction techniques. Best & improved results are obtained using feature reduction with FLANN-Genetic. In 2016 again, Chug and Malhotra [9] studied the effect of several ML techniques like GRNN, GMDH, Support Vector Machines (SVM), M5Rules, etc., while predicting the maintainability of seven different open-source software. Results were analyzed for Mean Absolute Error (MAE), RMSE & Pred(q) as the prediction accuracy measures, and it was found that the proposed ML techniques successfully predicted the maintainability for open source software & GMDH and GRNN with Genetic Adaptive Learning (GGAL) performed better than other techniques. In another study conducted by Kumar and Rath [40] in 2017, a Neuro-Fuzzy approach - a hybrid of NN & fuzzy logic was proposed for SMP with Principal Component Analysis (PCA) & Rough Set Analysis (RSA) for selecting suitable features. Results showed that the Neuro-Fuzzy model successfully predicts the software maintainability of OO systems with a further improvement in accuracy using feature selection techniques and parallel computing concepts. In 2018, Baskar and Chandrasekar [41] proved the superiority of the Neuro-PSO (NPSO) model over three other models, namely GMDH, GRNN, & PNN for SMP using MRE, MMRE, & Pred(q) as the accuracy measures. Again in 2018, Alsolai et al. [8] tried to assess the effectiveness of bagging models, i.e., the ensemble models for SMP & proved that there was a noteworthy enhancement in the performance using the bagging models. Further, if combined with *k*-Nearest Neighbour (*k*-NN) as the base model, the bagging model outperformed all the other models resulting in high accuracy.

In 2019, Jha et al. [42] put forth a deep learning approach (LSTM) for SMP using large datasets and 29 OO metrics. They compared the proposed approach with the results of five other ML algorithms, viz. ridge regression, DT, quantile regression forest, SVM, & PCA, to further affirm the LSTM approach's superiority to other models. In the same year, Wang et al. [43] introduced a fuzzy network-based approach for SMP using UIMS & QUES datasets, resulting in an improvement in transparency equal to 71.3% and an improvement in accuracy beyond 11.0%. Recently, in 2020, Gupta and Chug [44] described the cross-project technique for predicting maintainability based on the RMSE values leading to an improvement equal to 13.09% in the overall performance of the predictive models. Again, Gupta and Chug [45] in the same year presented an enhanced RF approach to predict

TABLE I. SUMMARIZED DETAILS OF DIFFERENT PREDICTION MODELS DEVELOPED BY RESEARCHERS FOR SMP

Study	Dataset	Metric Suite	Prediction Model	Validation Method	Prediction Accuracy Measure/s (PAMs)	Values for PAMs
Li & Henry [31]	UIMS & QUES	C&K metric suite	MLR	-	-	
Dagnipar & Jahnke [32]	Fujaba-UML (FUML) & Dynamic Object Browser (dobs)	Size-NIM & TNOS, Inheritance-NOC & AID, Cohesion-LCC	Regression Model	LOO	R-square adjusted (between 61.60% & 99.70%)	Between 61.60 & 99.70 %
Thwin & Quah [33]	QUES	DIT, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE1 & SIZE2.	WNN, GRNN	10-cross-validation	R squared, Correlation coefficient r	R^2=WNN-0.56067 & GRNN-0.71139, r= WNN-0.7609805 & GRNN-0.8580623
Koten & Gray [34]	UIMS & QUES	DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE1, SIZE2	Linear Regression (LR), BNM	10-cross validation	Absolute Residual (Ab.Res.), MRE, MMRE, Pred(q)	Using BNM, for UIMS, MMRE = 0.972, pred(0.25) = 0.446, pred(0.30) = 0.469 For QUES, MMRE = 0.452, pred(0.25) = 0.391, pred(0.30) = 0.430
Aggarwal et al. [35]	UIMS & QUES	LCOM, NOC, DIT, WMC, RFC, DAC, MPC, NOM	ANN	-	MARE, MRE	MARE=0.265, MRE=0.09
Zhou & Leung [36]	UIMS & QUES	WMC, DIT, RFC, NOC, LCOM, MPC, DAC, NOM & SIZE2 & SIZE1	MLR, ANN, RT, SVR, MARS	LOO cross-validation	Residual (Res.), Absolute Residual Error (ARE), MRE, MMRE, Pred(q)	Using MARS, for UIMS, MMRE=1.86, pred(0.25)=0.28, pred(0.30)=0.28, For QUES, MMRE=0.32, pred(0.25)=0.48, pred(0.30)=0.59
Elish & Elish [37]	UIMS & QUES	C&K - WMC, DIT, NOC, RFC, & LCOM; Li & Henry - MPC, DAC, NOM, & SIZE2; & SIZE1	TreeNet classifier	LOO cross-validation	MMRE, MRE, Pred(q), underestimation, overestimation	For UIMS, MMRE=1.57, pred(0.25)=0.31, pred(0.30)=0.41, For QUES, MMRE=0.42, pred(0.25)=0.58, pred(0.30)=0.65
Kaur et al. [38]	UIMS & QUES	LCOM, DIT, WMC, NOC, RFC, DAC, MPC, NOM	ANN, FIS, ANFIS	-	MARE, MRE, R-value, p-value	MARE=36.8% (feed forward ANN), 25.5% (GRNN), 30.8% (FIS), 24.2% (ANFIS)
Malhotra & Chug [39]	UIMS & QUES	WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, Size1, Size2	GMDH, GA, PNN	Hold-out	MRE, MMRE, Pred(q), R-Square, p-value	For GMDH, MMRE=0.210, pred(0.25)=0.69, pred(0.30)=0.722, pred(0.75)=0.944, For GA, MMRE=0.220, pred(0.25)=0.66, pred(0.30)=0.722, pred(0.75)=0.972, For PNN, MMRE=0.230, pred(0.25)=0.68, pred(0.30)=0.75, pred(0.75)=0.944,
Dubey et al. [21]	UIMS & QUES	DIT, NOC, RFC, WMC, LCOM, MPC, DAC, NOM, Size1, Size2	MLP NN	-	R-square, r, MAE, min Absolute Error (AE), max AE	Using MLP. for UIMS, R^2=0.8274, r=0.946, MAE=17.86, for QUES, R^2=0.988, r=0.976, MAE=5.264
Ahmed & Al-Jamini [3]	UIMS & QUES	DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE1, SIZE2	Fuzzy logic-based models - Mamdani Fuzzy Inference Engine & T-S	-	MRE, Normalized Root Mean square Error (NRMSE), MMRE, Pred(q)	For UIMS, MMRE=0.53, NRMSE=0.21, pred(0.25)=0.30, pred(0.30)=0.35, for QUES, MMRE=0.27, NRMSE=0.16, pred(0.25)=0.52, pred(0.30)=0.62
Malhotra & Chug [7]	FLMS & EASY	WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, SIZE1, SIZE2	GRNN, FF3LBPNN & GMDH	Hold-out	MRE, MMRE, Pred(q), Overestimate, Underestimate	For GRNN, MARE=0.5476, pred(0.25)=0.44, pred(0.30)=0.47, for FF3LBPNN, MARE=0.4578, pred(0.25)=0.51, pred(0.30)=0.59, for GMDH, MARE=0.3566, pred(0.25)=0.61, pred(0.30)=0.71
Malhotra & Chug [22]	Apache Poi & Rave	WMC, DIT, NOC, CBO, RFC, LCOM, LOC	A set of 14 statistical regression, traditional ML & hybrid algorithms	10-fold cross-validation	MRE, MMRE, Pred(0.25), Pred(0.30)	EAs achieved accuracy in the range of 22-25%

Study	Dataset	Metric Suite	Prediction Model	Validation Method	Prediction Accuracy Measure/s (PAMs)	Values for PAMs
Elish et al. [24]	UIMS & QUES for Regression Problem	WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, SIZE1, SIZE2	Different homogeneous & heterogeneous ensemble methods (AVG, WT & BT ensemble)	Ten-fold cross-validation	MMRE, Standard Deviation Magnitude of Relative Error (StdMRE), Pred(q)	Using BT, for UIMS, MMRE=0.97, StdMRE=1.61, pred(0.3)=25, for QUES, MMRE=0.41, StdMRE=0.32, pred(0.3)=60
Kumar et al. [6]	QUES & UIMS	WMC, NOC, DIT, RFC, LCOM, MPC, DAC, NOM, SIZE1, SIZE2	Neuro-GA	10-fold (QUES) and 5-fold (UIMS) cross-validation	MAE, MARE, RMSE, Standard Error of the Mean (SEM)	MMRE=0.3155 (UIMS), 0.3775 (QUES)
Kumar & Rath [23]	QUES & UIMS	WMC, DIT, NOC, LCOM, RFC, MPC, DAC, NOM, SIZE1, SIZE2	FLANN-GA, FLANN-PSO, FLANN-CSA	QUES-10-fold cross-validation, UIMS-5-fold cross-validation	MAE, MMRE, SEM, True Error (e), Estimate of True Error ($\hat{e}$)	Using FGA, MMRE=0.2881 (UIMS), 0.3889 (QUES), using FPSO, MMRE=0.3238 (UIMS), 0.3650 (QUES), using FCSA, MMRE=0.2843 (UIMS), 0.4469 (QUES)
Chug & Malhotra [9]	7 Open Source Software (Drumkit, OpenCV, Abdera, Ivy, Log4j, jEdit, JUnit)	WMC, DIT, NOC, RFC, DAM, MOA, MFA, CAM, AMC, CBO, LCOM, LCOM3, NPM, Ca, Ce, IC, LOC	Thirteen different ML classifiers like LR, M5Rules, GMDH, GRNN, SVM, PNN, etc.	Ten-fold cross-validation	MAE, RMSE, Pred(q)	Pred(0.25) > 60% in all cases using different ML techniques, GGAL & GMDH superior of all techniques
Kumar & Rath [40]	UIMS & QUES	DIT, WMC,RFC, DAC,LCOM, NOC,MPC, NOM,SIZE1, SIZE2	Neuro-Fuzzy Approach & Parallel Computing concept	Five-fold cross validation	MAE, MARE, MMRE, SEM, True Error (e), Estimate of True Error ($\hat{e}$)	MMRE=0.2826 (UIMS), 0.3375 (QUES)
Baskar & Chandrasekar [41]	QUES & UIMS	DIT, WMC, NOC, CBO, LCOM, MPC, RFC, DAC, NOM, Size1, Size2	NPSO	-	MRE, MMRE, Prediction (Pred(q))	MaxMRE=2.02547, MMRE=0.2931, pred(0.25)=0.2998, pred(0.75)=0.5612
Alsolai [8]	QUES	WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, SIZE2, SIZE1	Individual Models (RT, MLP, *k*-NN, M5Rules) & a bagging ensemble model	10-fold cross-validation	MRE, MMRE, Pred(0.25), Pred(0.30), Standard Deviation of Absolute Residuals (SD. Ab.Res.)	Using bagging ensemble models, for RT, MMRE = 0.3, pred(0.25)=0.6, pred(0.30)=0.7, for MLP, MMRE=0.2, pred(0.25)=0.7, pred(0.30)=0.8, for *k*-NN, MMRE=0.1, pred(0.25)=0.9, pred(0.30)=0.9, for M5Rules, MMRE= 0.3, pred(0.25)=0.5, pred(0.30)=0.6
Wang et al. [43]	UIMS & QUES	DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE2, SIZE1	Fuzzy network	-	MMRE, Transparency (TI)	Best MMRE=0.443, best TI=1
Gupta & Chug [44]	QUES & UIMS	DAC, DIT, LCOM, MPC, NOC, NOM, RFC, SIZE1, SIZE2, WMC	The Cross-Project technique using 19 different regression models	10-fold cross-validation	RMSE	Without CPSMP, Average RMSE=82.31, with CPSMP, Average RMSE=71.53
Gupta & Chug [45]	QUES & UIMS	DAC, DIT, LCOM, MPC, NOC, NOM, RFC, SIZE1, SIZE2, WMC	RF with three different feature selection techniques	10-fold cross-validation	R^2	For QUES, R^2=0.9207; for UIMS, R^2=0.9907

maintainability by combining RF algorithm with three different feature selection methods, i.e., chi-squared, RF, & linear correlation using R^2 as the accuracy estimator. Results show a remarkable improvement in the R^2 values using the enhanced RF approach compared to the basic existent RF approach. Further, Gupta and Chug [46] also propounded an effective utilization of Least Squares SVM (LS-SVM) in predicting maintainability by deriving notable values of MAE, RMSE, & MMRE on using LS-SVM.

It is observable from Table I and from the discussion of various studies conducted in the field of SMP that many researchers have already proposed a vast number of prediction models for SMP to date. However, most of them have used only the publicly available traditional Li and Henry datasets [31] for conducting their research rather than use open-source datasets. It was found that only two of the studies have used open-source datasets [9], [22]. Also, none of the researchers has made extensive use of ensemble methods, particularly the boosting techniques with any kind of dataset, to predict software maintainability apart from a few who considered ensemble models for SMP in their study [8], [24]. Hence, to overcome the above-identified gaps of the existing studies and due to the motivation gained through the availability and effectiveness of various BAs as discussed in the Introduction, the current study attempts to conduct an extensive analysis of BAs for SMP using open-source datasets.

III. Research Methodology

This section presents a detailed elucidation of the seven empirically collected open-source datasets used in this study and the process of collecting them. The independent and dependent variables chosen for the current study are also described in this section. A careful attempt is made while selecting the independent variables. All the possible and relevant design-related attributes of the OO paradigm, like abstraction, inheritance, complexity, coupling, and cohesion, are covered to sincerely analyze BAs capabilities for SMP. A collection of metrics picked up from different suites proposed by various researchers is selected, including the famous Chidamber & Kemerer (C&K) metrics suite [47]. However, due to some shortcomings encountered in the C&K metrics suite as identified by Malhotra and Chug [48], such as it does not contain any metric to measure the extent of database handling and also its inability to account for the structural complexity that exists in any software; two more metric suites are also considered, Henderson-Sellers [49] and Bansiya & Davis [50]. In totality, a set of seventeen OO metrics covering all the three metric suites has been used while conducting this study, as compiled in Table II.

The dependent variable used here is 'Change,' defined in respect to the number of lines in the source code that were added, deleted, or modified after delivering the final product to the customer. Further, as stated in Section I, there are two types of software attributes, i.e., internal and external. Internal attributes like coupling, cohesion, etc. can directly be measured by the developers during different SDLC stages. In contrast, external attributes like maintainability need to be measured indirectly using the metrics calculated for the internal attributes. In this study, an attempt has been made to measure an external attribute, i.e., maintainability (measured through the dependent variable 'Change') based on internal attributes by finding a correlation between different OO metrics & the dependent variable, developing various SMP models using several different BAs.

The overall implementation for the current study has been performed in Python 3 using Jupyter Notebook 5.7.8 platform. An overview of the research methodology being adopted for this study is depicted in Fig. 2. This section is further subdivided into different sub-sections.

A. Datasets and Data Collection

In this study, seven empirically collected datasets, i.e., Abdera, Ivy, jEdit, jTDS, Log4j, Apache Poi, and Apache Rave from various open-

TABLE II. Independent Variables Used in the Study

Metrics	Definition
WMC (Weighted Methods per Class)	WMC measures the static complexity of all the methods, which is the summation of McCabe's cyclomatic complexity of those methods.
DIT (Depth of Inheritance Tree)	DIT measures a class's position in the inheritance hierarchy, root class having this value equal to zero.
NOC (Number of Children)	The number of direct subclasses of a class is measured using the NOC metric.
CBO (Coupling between Object classes)	The number of classes coupled to a particular class is measured through CBO.
RFC (Response for a Class)	The cardinality of the response set of a class is measured through RFC, which is nothing but the sum of the number of local methods & number of methods called by these methods.
LCOM (Lack of Cohesion in Methods)	The number of disjoint sets of local methods is measured through the LCOM metric.
Ca (Afferent Couplings)	The number of classes that call a particular class is counted by the Ca metric.
Ce (Efferent Couplings)	The number of other classes that are called by a particular class is counted by the Ce metric.
NPM (Number of Public Methods)	The number of public methods of a class is counted by the NPM metric.
LCOM3 (Lack of Cohesion among the Methods of a particular Class)	LCOM3 metric is used to overcome specific disadvantages of the LCOM metric.
LOC (Lines of Code)	The number of code lines, excluding comments & blank lines, is measured using the LOC metric.
DAM (Data Access Metrics)	The ratio of the sum of private & protected methods of a particular class to the total number of attributes defined for that class is calculated by the DAM metric.
MOA (Measure of Aggregation)	The percentage of user-defined data in a particular class is calculated by the MOA metric.
MFA (Measure of Functional Abstraction)	The ratio of inherited methods to total methods in a class is calculated by the MFA metric.
CAM (Cohesion among the Methods of a class)	The similarity between different methods of a particular class is computed by the CAM metric.
IC (Inheritance Coupling)	The number of parent classes to which a particular class is coupled is counted by the IC metric.
AMC (Average Method Complexity)	The average value for McCabe's cyclomatic complexity of all the methods is calculated by the AMC metric.

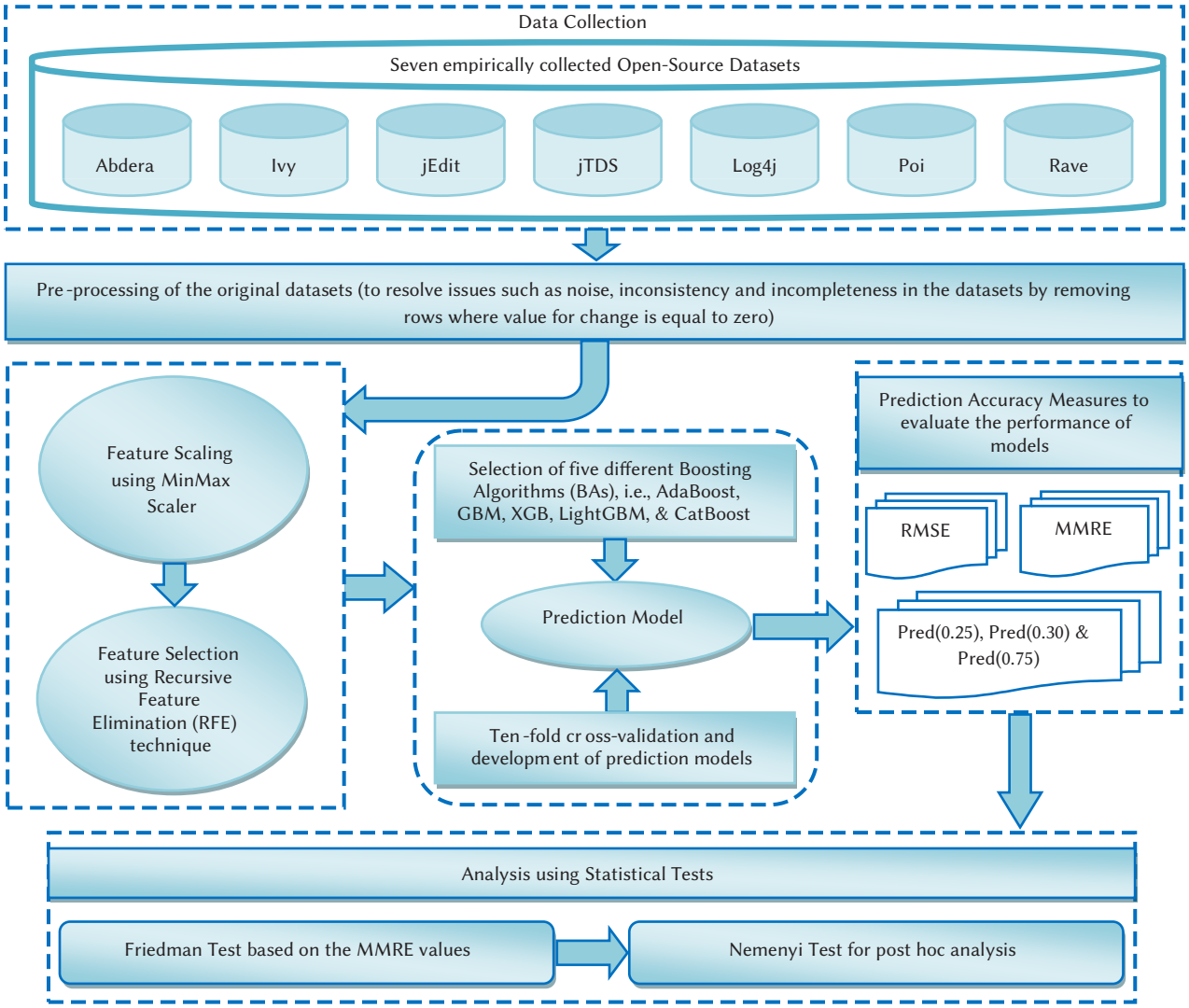

Fig. 2. Research methodology.

source repositories such as SourceForge and GitHub are analyzed using BAs for SMP. The details and description of various datasets follow:

- Abdera (685 classes) - Abdera is an atom parser generator that is used to build functionally a high in performance Internet for both the ends, i.e., the client and the server, by producing such designed documents which are high in quality. (https://github.com/apache/abdera)
- Ivy (613 classes) - Ivy is an assembly of various programs and open source libraries, which allow the broadcast of information using text messages along with a mechanism of subscription, which is usually based on the regular expressions. (https://github.com/apache/ant-ivy)
- jEdit (416 classes) - jEdit is one of the text editors written using Java. It can run on any of the operating systems and is customizable to a great extent. It is also extendable with the help of macros that are written in different scripting languages. (https://sourceforge.net/projects/jedit/)
- jTDS (64 classes) - jTDS is a free and open-source JDBC driver for Sybase ASE & Microsoft SQL Server written purely in Java, which is based on FreeTDS. Also, it is the fastest production-ready JDBC driver that exists currently. (http://jtds.sourceforge.net/)
- Log4j (350 classes) - Log4j is a software that allows control over log statements by the developer to decide which statements can be output having arbitrary granularity. It can entirely be configured at runtime with the help of externally configurable files. (https://github.com/apache/log4j)
- Poi (939 classes) - Poi stands for "Poor Obfuscation Implementation". It is a free open source library written in Java which is used to read and write confusing and hard to interpret document formats of Microsoft Office such as Word, Excel, PowerPoint, etc. (http://poi.apache.org/)
- Rave (671 classes) - Rave is a kind of mash-up supporting different platforms since it is highly customizable. It is a light-weighted and web-based data integration software written in Java that manages various social gadgets by hosting different widgets. It works by combining the data and functionality of two or even more than two sources for creating some new services. (https://rave.apache.org/)

B. Independent Variables

A set of seventeen different OO design metrics taken from different metrics suites proposed by several researchers in their studies [47], [49], [50] has been selected as independent variables of the current study to analyze different BAs for SMP. This set is chosen, keeping in mind that all the essential design-related facets of an OO paradigm

like complexity, abstraction, inheritance, coupling, and cohesion are covered. A glimpse of all the selected OO metrics, along with the description, can be viewed in Table II.

C. Dependent Variable

The dependent variable used here is the maintenance effort, which is the 'Change' measured as the number of LOCs for each of the classes that were added, deleted, or modified in the new version compared with the older version of particular software. This comparison is made between two successive versions of the same software where the new version is always the next version, by finding out the common classes of both the versions & subsequently finding the exact count of the lines that have been changed for each class. Each addition or deletion concerning a line is counted as a single change. In contrast, any modification is considered as two changes since, in modification, every deletion is followed by a corresponding addition. Different data points are generated for each class by calculating each of the OO metrics' values and then combining them with the corresponding values of Change made in a particular class.

Further, the details of different open source systems used here, including the version, size (number of classes), and the date of release, are provided in Table III.

TABLE III. Details of Different Open Source Systems Used

Software	Version	Size	Date of Release
Abdera	1.1.2 - 1.1.3	685 classes	15th January 2011 - 21st December 2012
Ivy	2.2.0 - 2.3.0	613 classes	13th June 2012 - 19th August 2015
jEdit	5.1.0 - 5.2.0	416 classes	28th July 2013 - 05th February 2015
jTDS	1.2.8 - 1.3.1	64 classes	08th June 2013
Log4j	1.2.16 - 1.2.17	350 classes	06th April 2010 - 06th May 2012
Poi	3.9 -3.10	939 classes	03rd December 2012 - 08th February 2014
Rave	0.21.1 - 0.22	671 classes	03rd May 2013 - 10th July 2013

D. Pre-processing of the Datasets

Pre-processing is one of the data mining techniques used for transforming raw real-world data into an easy to understand format resolving various issues such as noise (presence of outliers), inconsistency (discrepancy of codes or names), incompleteness (missing attribute values), lack of particular trend and error-proneness in the original datasets. While calculating the dependent variable during this study, a comparison between the old and new versions of all the datasets was made. A Java-based data mining tool for calculating the C&K Java metrics and several other metrics, namely, CKJM extended (http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/), has been used for empirical data collection. This tool processes the compiled Java files through their byte code & then calculates 19 different size & structure metrics for software. The results in the form of the metrics calculated for each class are displayed on the standard output or are saved in a particular file. The class-wise OO metrics (independent variables) for the older version, for example, for version 2.2.0 of the Ivy dataset, were collected on processing the jar file of that version through CKJM extended tool.

Further, classes common to both the versions, i.e., old and new, were extracted. Those classes added in the new version or deleted from the old version were plainly discarded. Both the library and interface classes were not included in this study. Further, those classes where the value of Change was zero were again excluded while considering the study's datasets. A graphical representation for the percentage reduction achieved for all the seven datasets is provided in Fig. 3.

After pre-processing, comparable classes for both versions were received. Afterward, a line by line comparison of these classes was made with the help of the Beyond Compare tool (https://www.scootersoftware.com/index.php), which provides a quick & easy comparison of files & folders at high speed. It verifies and compares the designated files or folders thoroughly in a byte-by-byte manner and further highlights the specified differences in a different color (generally red). This is required to compute the dependent variable's value, i.e., Change for each class through a line by line comparison. Each addition or deletion of a particular line in a class accounts for a single change, whereas any modification in a particular line of code accounts for two changes, .i.e., a single deletion followed by a single addition.

As shown in Fig. 3, some datasets have even more than 70% of their classes being discarded after pre-processing. However, such systems have been included in the original datasets for this study aiming to include diversified datasets where some datasets have a higher number of classes that get changed between different versions in contrast to the datasets where only a few of the classes get changed in going from one version to another.

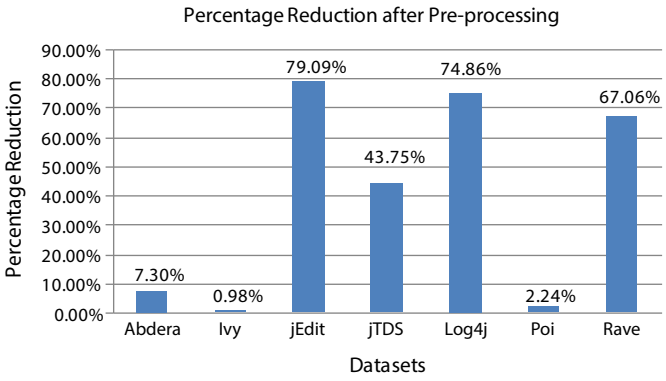

Fig. 3. Percentage reduction after pre-processing.

Further, the descriptive statistics for all the seven datasets have been provided in Table IV. Values highlighted in bold depict the minimum standard deviation for a particular OO metric, whereas the values highlighted in bold with an underline depict the maximum standard deviation for a particular metric.

E. Feature Scaling

Feature Scaling is a technique performed during data pre-processing to standardize a dataset's independent variables in a fixed range. It is also known as data normalization. Feature scaling is done since some algorithms cannot perform appropriately without normalization due to the original datasets' varying range of values. Various feature scaling methods are available for pre-processing data such as MinMaxScaler, RobustScaler, StandardScaler, etc. Of these, MinMaxScaler works by subtracting the minimum value from each of the values in a feature and then dividing it by range where the range is the difference of minimum & maximum values of a feature. In this study also, MinMaxScaler [51] in Python has been applied to normalize all the datasets used here, which transforms all the features by rescaling them to a given range (here, this range is [0, 1]).

F. Feature Selection

Feature selection is a method for choosing a subset of variables or features in ML to develop various models, ensuring the removal of redundant & irrelevant features without incurring the loss of information. It also enhances the prediction models' prediction accuracy since the quality of datasets due to the removal of inconsistent and noisy data and the model's execution time improves. Feature selection algorithms can broadly be classified under two

categories: the wrapper methods & the filter methods, as suggested by Kohavi and John [52]. However, a third category is also known as the embedded methods that combine the quality of wrappers and filters and simultaneously perform model fitting and feature selection such as lasso, ridge regression, etc. Wrappers usually evaluate a feature subset's performance based on the learning algorithm's resulting performance, such as forward selection, backward elimination, & RFE, etc. On the other hand, filter methods such as Pearson's correlation, Linear Discriminant Analysis (LDA), etc. generally use some proxy measure for evaluating the importance of the features based on some inherent characteristics without incorporating any learning algorithm. This is contrary to wrapper methods that use error rate for scoring a subset of features. Also, filters are computationally less intensive and faster than wrappers and produce a subset such that it is not tuned to some specific prediction model, making it a more general subset than the one derived from the wrappers.

In this study, one of the wrapper methods called RFE [53] has been used for selecting a subset of independent variables from all the initially selected seventeen independent variables. The improved results obtained in one of the studies conducted for the Intrusion Detection System [54] using the RFE algorithm for ranking the features provided the motivation for using the RFE algorithm in the current study as well for the selection of features. According to the study mentioned above, RFE improves accuracy by counting only the essential features while training, which reduces the learning time. An overall improvement of 0.4% in precision, between 16.2% and 26.8% improvement in false-negative rate, and a one-third reduction in time is achieved. RFE, in general, fits a model by recursively removing the weakest features by taking into account smaller & smaller groups of features, based on the significance of each of the features till a desired count for the features is subsequently reached. This importance is adjudged based on an external estimator, here, therespective boosting algorithms, that assign some weights to the features. Initially, the estimator training is done using a complete set of features, and the importance is obtained for each of the features. After this, the features with the least importance are removed from the initial set (here, one at each iteration since the value for parameter '*step,*' i.e., the count for features to be removed at each iteration is set equal to 1). This procedure is repeated for the reduced set in a recursive manner until a desired set of features that should be selected is eventually obtained. However, in the current study, the default value for the parameter '*n_features_to_select,*' i.e., the count of the features to be selected has been used. This default value is '*None*' and selects half of the total features leaving eight out of seventeen variables in this study, almost equal to half. This way, the RFE algorithm reduces the initial feature set by 52.94%.

Here, only the default values of different parameters for RFE have been used without any changes since the primary focus of the current study is to explore boosting algorithms. However, analyzing the role of tuning of different RFE algorithm parameters for feature selection can form a good base for future studies.

Features selected for all the seven datasets obtained by applying the RFE feature selection algorithm are presented in Table V. It is evident that out of a total of seventeen independent variables, LCOM, NPM, and LOC are found to be the most commonly selected variables. Following them, WMC, RFC, and Ce are the second most commonly selected independent variables. However, DIT, NOC, Ca, and IC have not been selected for any dataset. Also, MOA and MFA came out to be the least significant variables based on the RFE algorithm.

Further, the results in Table V show that each dataset has a different set of features obtained from the RFE algorithm. This difference can be explained through the descriptive statistics presented in Table IV, where values highlighted in bold depict the minimum values, whereas those highlighted in bold with an underline depict the maximum values for a particular metric. The values for standard deviation shows the extent of variation in different OO metrics' values for each of the datasets. The difference in the standard deviations of each of the OO metrics for all the seven datasets accounts for the difference in selecting various features for every dataset using the RFE method since different metrics affect distinct datasets differently while predicting maintainability. Also, due to a comparatively large difference in the variation of values for some OO metrics that have been calculated by finding the range of standard deviation for each metric from Table IV, only a particular set of metrics are selected by the RFE algorithm. For example, the most commonly selected metrics, i.e., LCOM, LOC, RFC, NPM, WMC, and Ce, have considerable variation in their values for almost all the datasets, which have a significant impact in predicting maintainability. Hence, these metrics have been selected by RFE for almost all the datasets.

G. Boosting Algorithms

This sub-section provides an overview of different BAs, i.e., the ensemble of ML algorithms used in the current study to develop various SMP prediction models. A set of five most commonly used BAs, namely AdaBoost, GBM, XGB, LightGBM, and CatBoost, has been applied to identify specific patterns while training each of the seven datasets. These algorithms explore the complex relationship or the correlation among various independent variables & the dependent variable, using the knowledge derived during the training process for making predictions.

1. AdaBoost

AdaBoost is one of the first ensemble boosting techniques proposed by Freund and Schapire [55], [56] to be adapted in practice to solve both regression & classification problems. It works by creating multiple sequential models from poorly performing models, each correcting the previous model's errors to increase the accuracy to build a reliable model ultimately. It is an iterative ensemble technique that generally uses DTs for modeling. However, any ML technique can be used as a base classifier, provided it accepts the weights on the training set. In the current study, the DT regressor has been used as the base estimator while implementing AdaBoost. The basic idea behind AdaBoost is to ensure that the unusual observations are predicted accurately by setting up the weights of classifiers and training data samples in every iteration. AdaBoost is expected to fulfill two main conditions; first, the classifier's interactive training on several weighted training examples should be done, and second, it should try to provide an accurate fit for the above examples in each iteration by minimizing the error in training.

2. GBM

GBM is another ensemble ML algorithm used for classification & regression problems by combining multiple weak learners to develop a strong learner. Friedman described GBM in two of his popular studies in 1999 and 2001 [57], [58]. Generally, Regression Trees (RTs) are used as base learners, and each tree is built subsequently in a series based on the errors measured by the previous tree, and the foremost goal is to overcome these errors. The difference here is that the weights are not incremented for the misclassified values; instead, an attempt is made to optimize and reduce the loss function that adds several weak learners by adding some new model. Broadly, GBM comprises three main components, i.e., the loss function that should be optimized, an additive model for minimizing the loss function & a weak learner for making the predictions.

3. XGB

XGB is a highly effective, novel, and advanced implementation for the GBM ensemble ML algorithm, particularly RTs and K classification.

TABLE IV. Descriptive Statistics for Open Source Datasets (SD = Standard Deviation)

Metric	Abdera			Ivy			jEdit			jTDS		
	Min	Max	SD	Min	Max	SD	Min	Max	SD	Min	Max	SD
WMC	0	255	21.21	1	243	21.64	1	275	34.34	0	211	**42.16**
DIT	0	4	0.63	0	4	0.60	0	7	**1.72**	0	3	0.62
NOC	0	17	1.02	0	17	1.22	0	20	2.22	0	2	0.44
CBO	0	17	1.94	0	17	1.96	1	396	**44.57**	0	34	6.84
RFC	0	256	21.23	2	244	21.64	1	570	**89.92**	0	293	70.72
LCOM	0	32385	1724.93	0	29403	2158.27	0	21943	2605.30	0	21831	**3432.78**
Ca	0	14	1.64	0	17	1.74	0	327	**37.20**	0	30	4.95
Ce	0	5	0.93	0	9	1.17	0	116	**14.86**	0	30	5.07
NPM	0	254	20.65	0	215	18.96	0	228	27.81	0	191	**40.07**
LCOM3	1.0039	2	0.43	1.0041	2	0.42	0	2	0.57	0	2	0.47
LOC	0	1531	123.34	6	1461	132.98	1	10007	**1471.109**	4	8251	1448.33
DAM	0	1	**0.49**	0	1	0.47	0	1	0.44	0	1	**0.39**
MOA	0	327	**14.81**	0	7	0.69	0	13	2.59	0	14	2.40
MFA	0	1	0.17	0	1	0.12	0	0.9987	0.37	0	1	0.23
CAM	0	1	0.30	0.0556	1	0.28	0.0455	1	0.22	0	1	**0.19**
IC	0	3	0.28	0	2	0.21	0	3	**0.58**	0	2	0.41
AMC	0	5	2.24	0	5	2.39	0	139.451	32.32	0	255.11	**46.30**
Change	2	14667	1172.30	2	17586	**1434.72**	1	249	**43.54**	0	355	59.86

Metric	Log4j			Poi			Rave		
	Min	Max	SD	Min	Max	SD	Min	Max	SD
WMC	1	104	13.35	0	165	14.19	0	62	**9.69**
DIT	0	6	1.48	0	5	0.70	0	4	**0.25**
NOC	0	4	0.64	0	151	**5.13**	0	2	**0.21**
CBO	0	76	10.98	0	228	17.25	0	4	**0.69**
RFC	1	130	25.05	0	426	33.42	0	63	**9.71**
LCOM	0	5356	575.44	0	5908	475.71	0	1891	**209.75**
Ca	0	65	9.16	0	228	14.52	0	3	**0.41**
Ce	0	29	4.81	0	167	9.60	0	2	**0.43**
NPM	0	31	**7.50**	0	140	12.51	0	57	9.24
LCOM3	0	2	0.48	0	2	**0.60**	0	2	**0.39**
LOC	3	1864	283.14	0	4455	370.97	0	405	**59.97**
DAM	0	1	0.45	0	1	0.43	0	1	0.43
MOA	0	14	2.13	0	49	3.54	0	3	**0.29**
MFA	0	1	**0.38**	0	1	0.18	0	1	**0.09**
CAM	0.0726	1	0.24	0	1	0.24	0	1	**0.31**
IC	0	3	0.51	0	3	0.27	0	1	**0.20**
AMC	0	205	25.83	0	392.2222	23.40	0	5	**1.69**
Change	1	1612	194.36	2	17956	1331.08	1	470	55.08

It was proposed by Chen and Guestrin in 2016 [30]. It prevents over-fitting and intends towards the optimization of computational resources. It is possible through the simplification of objective functions by allowing the combinations of regularization, provided an optimum computational speed is also maintained alongside. During the training phase, automatic parallel calculations are performed for various functions in XGB. XGB is approximately ten times faster than any other BA and is also known as a "regularized boosting technique." In the current study, 'gbtree' booster, i.e., tree-based models, have been used to run at every iteration.

4. LightGBM

LightGBM was proposed by Ke et al. [59] as a newer implementation of Gradient Boosting DT (GBDT). It is a fast and distributed framework based on DT algorithms and is used for different ML tasks such as ranking & classification. It makes use of a leaf-wise strategy while splitting the trees with the best fit, unlike other algorithms that use . a level-wise or depth-wise approach. Also, LightGBM being leaf-wise is more accurate than other BAs since it can reduce more loss while growing on the same leaf, and it also ensures reduced memory consumption. In the current study, while implementing LightGBM, RF has been used as the basic boosting type.

TABLE V. Feature Selection Through RFE Algorithm

Datasets	Features selected through RFE Algorithm
Abdera	WMC, CBO, RFC, LCOM, NPM, LCOM3, LOC, CAM
Ivy	WMC, RFC, LCOM, Ce, NPM, LCOM3, LOC, CAM
jEdit	RFC, LCOM, Ce, NPM, LCOM3, LOC, MFA, AMC
jTDS	WMC, CBO, RFC, LCOM, Ce, NPM, LOC, AMC
Log4j	WMC, LCOM, Ce, NPM, LOC, DAM, MOA, CAM
Poi	WMC, CBO, RFC, LCOM, Ce, NPM, LOC, CAM
Rave	WMC, RFC, LCOM, Ce, NPM, LOC, DAM, CAM

5. CatBoost

The fundamental algorithmic approaches behind CatBoost were explained by Prokhorenkova et al. [60] in 2018 in one of their studies. CatBoost is also a GBDT for handling categorical features well. It allows one to use the complete dataset for training, and an extensive pre-processing of data is not required here. Rather than the pre-processing time, CatBoost deals with the categorical features while training. As per authors, target statistics can efficiently handle categorical features ensuring minimum loss of information. However, in regression, the initial value is calculated using a standard technique where the average value for a label in the dataset is considered. Overall, CatBoost is a robust, easy-to-use, and high in performance BA in the family of ML algorithms.

H. Cross-Validation Technique

Cross-validation is a model validation technique to account for the accuracy of a prediction model on new data & to check if this model can be generalized for real-world datasets. It is used in a scenario where the ultimate goal is prediction. For prediction, the model is trained using a known dataset (training set), whereas testing of this model is done on a new dataset (test set) after training. Several types of cross-validation approaches exist, such as LOO, *k*-fold, hold-out, etc. However, the *k*-fold technique is one of the most basic cross-validation forms with *k* equal to 10 [61]. Its significance lies in the fact that it can use the dataset for dual purpose, i.e., training and testing. As per literature, 5-fold and 10-fold approaches are the most commonly used cross-validation approaches to design a model. According to [61], *k*-fold validation using moderate values of k, i.e., between 10-20, helps minimize the variance with an increase in the bias; *k* equal to 10 being the most preferred, most frequently used, and the most recommended one.

Further, if the value of *k* decreases say between 2-5, along with smaller sample size, then variance seeks in because of the instability in the training set, which further increases the variance. Hence, the 10-fold cross-validation technique has been selected for the current study, keeping the above points in mind. In this technique, the complete dataset is sub-divided into 10 equal partitions, of which one partition is considered validation data for testing the model, whereas the rest of the partitions are utilized to train the prediction model. The same process is iteratively repeated 10 times for each of the 10 partitions, each partition being used as the validation set exactly once. Lastly, a single final estimation is reached by calculating an average of the 10 results obtained above.

I. Prediction Accuracy Measures

This section presents various prediction accuracy measures for the current study to assess various BAs performance for SMP for all the seven datasets. Estimating and assessing the accuracy of a prediction model is an essential part of any study. This is done by comparing the dependent variable's predicted value with its actual value and finding the corresponding value of the error. In literature, different residual-based prediction accuracy measures have been suggested by various researchers. However, in this study, the following three measures of accuracy have been selected for estimating the accuracy of the proposed prediction models, as suggested by Conte et al. [62] and Kitchenham et al. [63], [64] in their studies.

1. Root Mean Square Error (RMSE)

RMSE is the measure of standard deviation in the prediction errors, i.e., the residuals. It is calculated by taking the square root of Mean Square Error (MSE) using the formula defined by Conte et al. (1986) [62].

$$MSE = \frac{1}{n}\sum_{i=1}^{i=n}(y_i - \hat{y}_i)^2 \tag{1}$$

where y_i is the i^{th} value being observed & $\hat{y}_i$ is the i^{th} value, which is predicted by the prediction model. Further, from this formula, the formula for RMSE can also be derived.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{i=n}(y_i - \hat{y}_i)^2} \tag{2}$$

Residuals measure the data points' remoteness from the standard regression line, and RMSE measures these residuals' spread. Alternately, RMSE describes the concentration of the data around the best line of fit. RMSE is one of the most widely and frequently used indicators to account for the goodness of fit in regression models. The significance of using RMSE may be attributed to the risk-averse predictors, where large deviations are penalized more in comparison to small deviations by RMSE. This happens as RMSE is based on the mean or average value obtained by summing the residuals' squared values. Another implication of squaring the errors in determining the importance of RMSE lies in the fact that RMSE would be even more useful in cases where large errors may particularly be highly undesirable. RMSE is highly useful to compare several prediction models developed using different techniques. The magnitudes of the prediction errors for several times are accumulated into a compound indicator of the predictive power with the help of RMSE. Further, RMSE values may range from 0 to ∞ and are inconsiderate towards the errors' direction. RMSE is a negatively-oriented indicator in which lower values are considered to be the better values. Also, RMSE does not essentially increase with a rise in the error variance but increases with a rise in the variance related to the frequency distribution of the magnitude of the errors.

2. Mean Magnitude of Relative Error (MMRE)

MMRE is the most frequently used quality indicator in software engineering while accounting for the performance of various software estimation models defined by Conte et al. (1986) [62].

$$MMRE = \frac{1}{n}\sum_{i=1}^{i=n}\left(\frac{|y_i - \hat{y}_i|}{y_i}\right) \tag{3}$$

MMRE is different from relative error in the sense that unlike relative error, the absolute value of the differences between the actual & predicted values is used while calculating MMRE. The use of absolute value prevents both underestimation and overestimation by canceling each other out. MMRE measures the variance or the spread of accuracy (predicted / actual). To better understand what MMRE measures, 'y' is considered a normally distributed random variable having μ and σ^2 as the mean and variance, respectively. Iglewicz [65] has already illustrated the following for a sample of 'n' observations, where $\bar{y}$ is the mean of those observations:

$$d_n = \frac{1}{n}\sum|y_i - \bar{y}| \to \sigma\sqrt{\frac{\pi}{2}} \quad \text{as } n \to \infty \tag{4}$$

On re-writing MMRE as below:

$$\frac{1}{n}\sum_{i=1}^{i=n}\left|\frac{\hat{y}_i}{y_i} - 1\right| = \frac{1}{n}\sum_{i=n}^{i=1}|z_i - 1| \tag{5}$$

it is evident that if $\hat{y}_i$ is an unbiased estimation of y_i, then the value of $z_i = \frac{\hat{y}_i}{y_i}$ as expected is equal to 1. If z_i is normally distributed having mean and variance equal to 1 and σ_z, respectively, then MMRE tends towards the value of $\sigma_z\frac{\pi}{2}$. This illustrates that MMRE estimates the spread or the variance of the 'z' variable, which is not that susceptible to the large outliers as the RMS estimate is. As MMRE measures the spread, it would be wrong to call it a prediction accuracy metric. However, 'z' has an optimal defined value equal to 1, indicating whether or not the prediction system's estimation is under or overestimated and hence a better criterion to indicate the prediction accuracy. This further shows that any prediction model's quality can be described in respect of the average value of the 'z' variable, and MMRE is used for assessing the variability of this variable 'z.'

3. Pred(m)

It measures the proportion of the values predicted by the prediction models with a magnitude of MRE lower than or equal to a specific value.

$$Pred(m) = \frac{k}{n} \quad (6)$$

where 'm' represents the particular specified value, 'k' refers to the number of predictions in the dataset whose MRE is lower than or equal to 'm' & 'n' is the total number of observations in the dataset. Pred(m) measures the kurtosis or the shape of the accuracy (predicted / actual). Pred(m) is the percentage of predictions within m% of the initial or the actual values. 'm' is usually set to 25, 30, & 75 such that Pred(m) shows how much proportion of the predictions lie within the tolerance of 25%, 30%, & 75% respectively. Further, Pred(m) is inconsiderate to the extent of the inaccuracy of the predictions that lie beyond a particular level of tolerance. For example, for two different prediction models whose predictions deviate by 27% and 270%, respectively; a Pred(25) indicator will not differentiate between the two models. Like MMRE, Pred(m) should preferably be formulated for prediction by considering the actuals' percentage lying within m% of the prediction. As mentioned earlier, Pred(m) is a measure for kurtosis that provides the degree to which a particular distribution has been peaked surrounding the central value. To better understand Pred(m), consider a case where certain distribution is more peaked than a normal distribution. As a result, if a sample is selected from a distribution having more peak, then it would have comparatively more values within 25% (in case of Pred(25)) of the mean value than normal. On the other hand, if a sample is selected from a distribution having a flatter peak, it would have comparatively fewer values within 25% than the normal scenario.

J. Friedman Test for Ranking the Performance

Friedman test [66] is a statistical test, which is non-parametric in nature, to rank the performance of various algorithms used in a study by finding any significant difference between those algorithms' performance. Here, this test has been used to rank the performance of different BAs used in this study. Before the test, a hypothesis is formulated.

Null Hypothesis (H_0) - No significant difference exists between the performances of various BAs used in this study.

Alternate Hypothesis (H_1) - A significant difference exists between the performances of various BAs used in this study.

Further, the Friedman measure is calculated using the given formula.

$$\chi_r^2 = \left(\frac{12}{Nk(k+1)}\sum_{i=1}^{k} R^2\right) - 3N(k+1) \quad (7)$$

where R represents the average rank for each BA, N stands for the number of datasets used in the current study, and k represents the number of BAs considered for the ranking. The value for $\chi_{calculated}$ is calculated using (7) and further compared with $\chi_{tabulated}$ given in the distribution table for chi-square. If $\chi_{calculated}$, which is the Friedman measure, falls in the critical region, it is concluded that a significant difference exists between the performance of various BAs, thereby rejecting the null hypothesis & accepting the alternate hypothesis. However, if $\chi_{calculated}$ does not fall in the critical region, it is then concluded that no significant difference exists between the performance of various BAs, thereby rejecting the alternate hypothesis and accepting the null hypothesis.

Each BA is ranked individually with the help of Friedman's Individual Rank (FIR) using (8).

$$FIR = \frac{C}{N} \quad (8)$$

where C represents the cumulative rank & N represents the total number of datasets. Based on the FIR values calculated for each BA, one having the lowest FIR value is declared to be the best performer whereas, on the other hand, BA having the highest FIR value is declared to be the worst performer. Further, suppose the values of FIR obtained for various prediction accuracy measures are found significant. In that case, post hoc analysis should be done using the Nemenyi test to check if the difference between various mean ranks obtained by the Friedman test is statistically significant or not. However, in this study, both Friedman and Nemenyi tests are performed only for MMRE.

K. Nemenyi Test for Post Hoc Analysis

Nemenyi test [67] is a test in statistics for post hoc analysis that intends to find groups of data that differ when statistical tests such as the Friedman test for multiple comparisons rejects the null hypothesis stating that no significant difference exists between the performance of various groups of data. This test is used to make pair-wise tests of performance for comparing the performance of various BAs used in this study to find if any statistical difference exists among them. The first step for conducting the Nemenyi test is to calculate the Critical Difference (CD), which depends on the total number of BAs & the number of datasets used, along with the level of significance, using (9).

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \quad (9)$$

where k represents the total number of BAs, N represents the number of data samples & q_α is the critical value as suggested by Demsar [68] in his study; based on the Studentized range statistics for a particular significance level. After calculating CD, the individual differences between the FIR values of different pairs of BAs are calculated to compare each possible pair of BAs' performance during the post hoc analysis. If the difference calculated for each possible pair of BAs comes out to be either more than or equal to CD, then the performance of that particular pair is considered statistically significant for the selected level of significance. On the other hand, if this difference is less than CD, then that particular pair's performance is statistically not significant.

IV. Results & Discussions

This section presents the results of the current study & a detailed discussion and analysis of these results to analyze different BAs for SMP using open-source datasets. A few of the selected plots for all the seven datasets showing true versus predicted values for the best performing BA based on MMRE values are presented in Fig. 4.

It is noted that based on the MMRE values, XGB performed the best for six out of the seven datasets except for jEdit, for which CatBoost performed the best. Subsequently, various RQs framed for the current study in the introduction section are answered in this section.

RQ1: Whether BAs can be applied for SMP?

Various prediction accuracy measures, i.e., RMSE, MMRE, and Pred(0.25), Pred(0.30) & Pred(0.75), have been used to analyze the performance of different BAs used in this study for all the seven datasets using (2), (3) and (6), respectively. The results obtained for all the five BAs validated using ten-fold cross-validation are presented, compared, and analyzed in this section.

Table VI provides the RMSE values for each of the BAs for all the seven datasets. The best value of RMSE for each dataset is marked in bold. It is clear from Table VI that based on the RMSE, GBM performed the best, resulting in the lowest RMSE values for three of the seven datasets, namely jEdit, jTDS, and Log4j, i.e., for 42.86% of the total datasets. Similarly, LightGBM performed the second-best in terms of RMSE for Ivy and Poi, i.e., for 28.57% of the datasets, whereas AdaBoost and CatBoost performed well for Abdera and Rave, respectively.

However, XGB came out to be the worst performer since it did not perform well for any of the datasets when RMSE is considered the accuracy measure. Overall, if we look at Table VI, it is concluded that the best RMSE value equal to 43.42 is obtained for the jEdit dataset using GBM.

TABLE VI. RMSE Values For All the Seven Datasets using BAs

Accuracy Measure	RMSE						
Datasets	Abdera	Ivy	jEdit	jTDS	Log4j	Poi	Rave
Boosting Algorithm							
AdaBoost	**986.25**	1251.82	46.63	70.15	184.08	1241.31	55.81
GBM	1150.49	1389.05	**43.42**	**67.54**	**179.01**	1310.24	55.30
XGB	1164.34	1368.90	45.81	76.14	211.25	1324.29	58.57
LightGBM	1070.95	**1199.34**	54.92	70.21	195.85	**1197.91**	75.98
CatBoost	1077.89	1278.42	44.24	76.03	201.73	1230.27	**54.78**

The MMRE values for each of the BAs validated using ten-fold cross-validation for all the datasets are provided in Table VII. Also, the least obtained values of MMRE for each dataset, which are also the best, are marked in bold since a low value of MMRE indicates less error in prediction and hence better accuracy. Every row shows MMRE values for a particular BA on a specific dataset.

TABLE VII. MMRE Values For All the Seven Datasets using BAs

Accuracy Measure	MMRE						
Datasets	Abdera	Ivy	jEdit	jTDS	Log4j	Poi	Rave
Boosting Algorithm							
AdaBoost	4.36	7.31	2.22	2.04	7.98	6.49	2.81
GBM	4.56	7.87	3.86	2.97	8.75	6.44	3.91
XGB	**1.84**	**2.98**	1.77	**0.90**	**3.82**	**2.85**	**1.43**
LightGBM	4.91	7.45	4.94	3.33	8.90	6.88	5.01
CatBoost	2.93	4.62	**1.71**	2.54	6.26	4.09	1.92

It is concluded from Table VII that based on the MMRE values so obtained, XGB performed the best for six out of seven datasets, i.e., for 85.71% of the total datasets by providing the least values for MMRE. However, in the jEdit dataset, CatBoost performed the best in terms of MMRE with a value equal to 1.71. Overall, XGB performed the best with the jTDS dataset having MMRE value equal to 0.90 when MMRE is considered an accuracy measure to analyze different BAs performance over seven open-source datasets.

Further, the prediction accuracy of all the BAs for each of the datasets has been calculated at 25%, 30% & 75%, and results are summed up in Table VIII where each column for a particular dataset is further subdivided into three columns; one each for Pred(0.25), Pred(0.30) & Pred(0.75). Best obtained values are highlighted in the table for each dataset & each prediction accuracy level, i.e., 25%, 30%, and 75%.

On analyzing the values in Table VIII, it is observed that for Pred(0.25), which ranges up to 31%, CatBoost BA is found to be the most accurate in the case of Abdera. If we consider Pred(0.30) for prediction accuracy, which ranges up to 36%, it is found that CatBoost for Abdera performed the best. Again, for Pred(0.75), which ranges up to 79%, it is observed that XGB performed the best for Abdera by providing the highest prediction accuracy equal to 79%, which further assures the effectiveness of BAs for SMP. Overall, it is concluded that the best prediction accuracies are obtained for Abdera. Also, in the case of Pred(0.30), LightGBM performed the best for three out of the seven datasets, i.e., Ivy, jEdit, and Poi, whereas, in the case of Pred(0.75), XGB gave the best performance for six out of the seven datasets (excluding Log4j) with prediction accuracies ranging from 51% to 79% which are satisfactory and reasonable.

Further, the difference in the performance of different BAs for different datasets can be accounted to the wide range and variations in the values of various OO metrics and the dependent variable 'Change' measured through standard deviation as presented in Table IV, representing the descriptive statistics for each of the seven datasets. Hence, a different range of values is obtained for various prediction accuracy measures used in the current study. For example, if we consider the predictor variable 'Change,' then the difference between the maximum and minimum values for standard deviation so obtained or the range for standard deviation is 1391.18, which is really wide and hence the difference in performance.

Hence, based on each of the BAs' overall performance for all the seven datasets based on the values obtained for the five prediction accuracy measures and from the comparative analysis of these measures, it can be concluded that BAs can effectively be applied for SMP.

RQ2: Which BA performs the best amongst different BAs based on various prediction accuracy measures for different open-source datasets?

A non-parametric statistical test named the Friedman test is applied for an extensive analysis of different BAs used in the current study to determine if a significant difference exists between various BAs. Friedman's test is selected because it is a non-parametric test; it is safe and robust as it does not assume homogeneity of variance or the normal distributions as recommended by Demsar in his work [68]. The Friedman test has been conducted for comparing the performance of five different BAs applied on seven different datasets based on the MMRE values by calculating the value of critical region for the level of significance equal to 5% & degree of freedom equal to 4, i.e., 5 BAs minus 1 (or k-1 where 'k' is the total number of BAs used in this study). Value for $\chi_{tabulated}$ is read from the Chi-square table corresponding to the 95% significance level and degree of freedom equal to 4.

TABLE VIII. Pred(m) Values for All the Seven Datasets Using BAs

Accuracy Measure	Pred (m)																				
Datasets	Abdera			Ivy			jEdit			jTDS			Log4j			Poi			Rave		
Boosting Algorithm	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)
AdaBoost	0.24	0.28	0.62	**0.21**	0.24	0.54	0.20	0.22	0.51	0.14	**0.19**	0.33	0.25	0.28	**0.64**	**0.24**	**0.27**	0.57	0.12	0.14	0.47
GBM	0.15	0.19	0.41	0.12	0.16	0.39	0.09	0.13	0.30	0.14	**0.19**	0.44	**0.27**	**0.30**	0.60	0.13	0.16	0.45	0.13	0.16	0.36
XGB	0.21	0.25	**0.79**	0.17	0.21	**0.66**	0.07	0.14	**0.52**	0.14	0.14	**0.53**	0.02	0.02	0.49	0.13	0.17	**0.66**	**0.14**	**0.19**	**0.51**
LightGBM	0.23	0.27	0.57	**0.21**	**0.26**	0.53	**0.21**	**0.23**	0.47	**0.17**	0.17	0.36	0.24	0.25	0.59	0.23	**0.27**	0.57	0.12	0.14	0.39
CatBoost	**0.31**	**0.36**	0.76	**0.21**	0.24	0.61	0.11	0.14	0.44	0.08	0.11	0.44	0.07	0.11	**0.64**	0.21	0.24	**0.66**	**0.14**	0.17	0.49

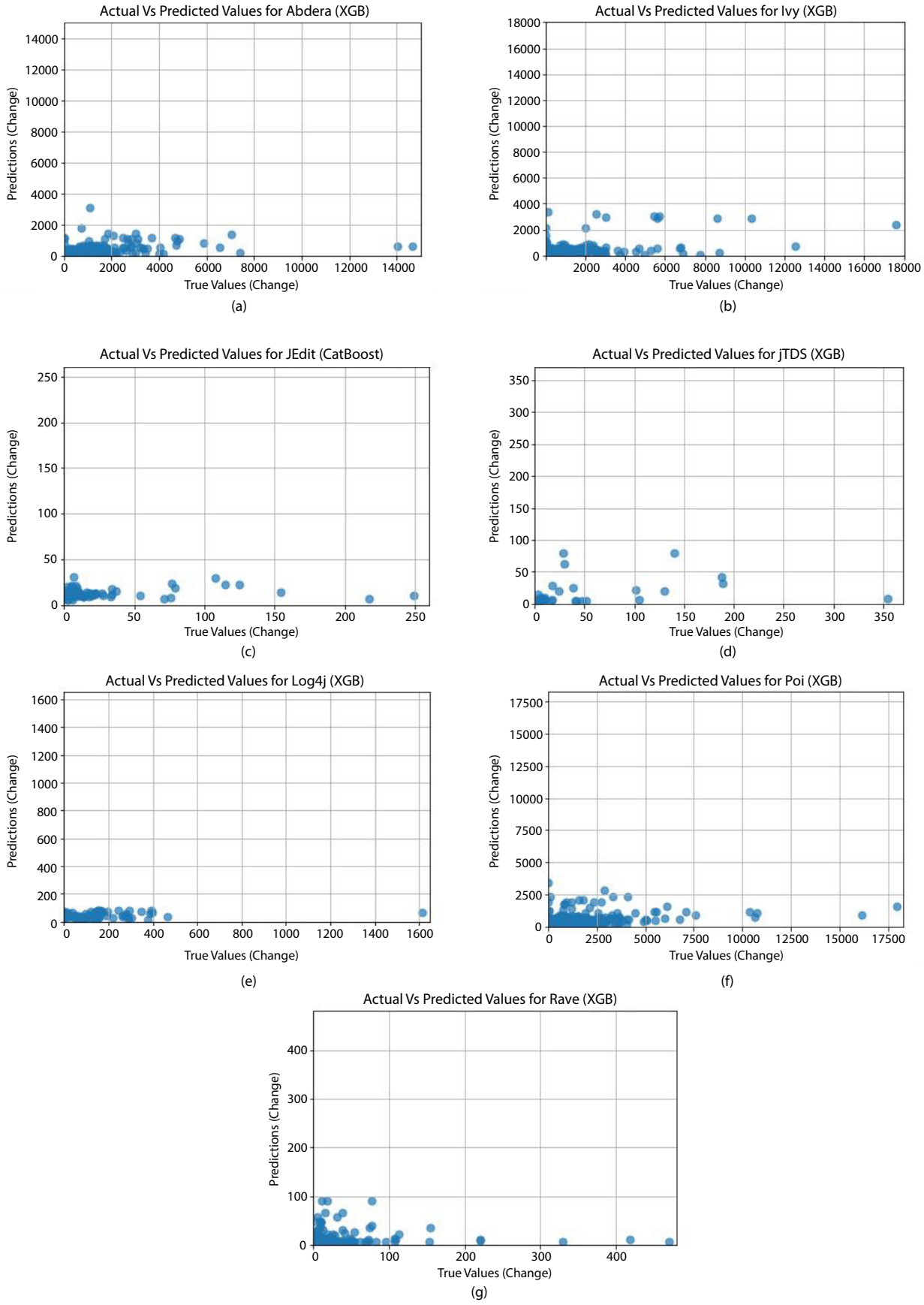

Fig. 4. Plots for (a) Abdera; (b) Ivy; (c) jEdit; (d) jTDS; (e) Log4j; (f) Poi; (g) Rave showing true versus predicted values for the best performing BA based on the MMRE values.

According to the null hypothesis of Friedman's test, which states that no significant difference exists between the performance of various BAs used in this study, it was found that at 0.05 level of significance, $\chi_{calculated}$ which is the Friedman measure lies in the critical region for MMRE. Therefore, it is concluded by accepting the alternative hypothesis and rejecting the null hypothesis that a significant difference exists between the performances of various BAs used in this study. Test statistics for the Friedman test are stated in Table IX.

TABLE IX. Friedman Test - Test Statistics for MMRE

N	7
Chi-Square	24.914
Df	4
Asymp. Sig.	.000

Further, each BA is ranked for its performance by calculating FIR from (8) based on MMRE, and the values obtained for the mean ranks of different BAs for MMRE are compiled in Table X. It is also known that the lowest mean rank indicates the best performance. Hence, it is evident from Table X that based on the MMRE values, XGB performed the best, whereas CatBoost performed the second best. Also, LightGBM is found to be the worst performer.

TABLE X. Mean Ranking of BAs on Applying Friedman Test for MMRE

Boosting Algorithm	XGB	CatBoost	AdaBoost	GBM	LightGBM
Mean Rank	1.14	2.00	3.00	4.00	4.86

On exploring the reason for the difference in the performances of XGB and LightGBM, it was found that due to the use of Newton boosting in XGB, it is likely to learn better structures. Apart from this, XGB consists of an extra parameter for regularization, namely column sub-sampling (including built-in L1 & L2 regularization that prevents the model from being over-fitted) for reducing the correlation between each of the trees further. Also, XGB uses a histogram-based pre-sorted algorithm for computing the best split and achieve faster training. In contrast, LightGBM uses the GOSS technique, i.e., Gradient-based One Side Sampling, for filtering the data samples to find a value for the split. Unlike other algorithms, where trees grow horizontally (level wise), in LightGBM, trees grow vertically (leaf wise) by choosing the leaf having maximum delta loss.

A further implication of the results can be the utilization of boosting algorithms, especially the XGB, for developing different prediction models in a scenario where training data is limited, time for training is less, and the expertise for tuning of parameters also lacks.

RQ3: What is the comparative performance of various BAs during post hoc analysis when MMRE is taken as an accuracy measure?

After the Friedman test, post hoc analysis using the Nemenyi test is performed to check if the differences between the performances of various BAs based on the FIR values, as concluded in RQ2 above, are statistically significant or not.

The value for CD is calculated to be equal to 2.31 using (9) where k is taken to be 5 (number of BAs), and N is taken to be 7 (number of datasets). After this, all the possible pairs of BAs are formed with every other BA for calculating the rank differences between them, i.e., between the FIR values so obtained. Here, ten such combinations are formed for five different BAs for MMRE, and the same results are compiled in Table XI.

Values for differences in ranks greater than or equal to CD, i.e., 2.31, are shown in bold in Table XI. It is observed that 3 out of 10, i.e., 30% of the total pairs of BAs have been highlighted, which means 30% of the pairs have the difference above or equal to CD, showing that the performance of these pairs is found to be significantly different using Nemenyi test.

Differences calculated in Table XI also show that XGB performed better than GBM, and LightGBM, whereas CatBoost performed better than LightGBM only. Therefore, from this post hoc analysis of MMRE values, it is concluded that XGB and CatBoost significantly outperformed the rest of the BAs. However, the differences between the performances of all other pairs of BAs have not been found significant.

TABLE XI. Pair-Wise Rank Differences Between Different BAs in Terms of MMRE

Boosting Algorithm	XGB	CatBoost	AdaBoost	GBM	LightGBM
XGB	-	0.86	1.86	**2.86**	**3.72**
CatBoost		-	1.00	2.00	**2.86**
AdaBoost			-	1.00	1.86
GBM				-	0.86
LightGBM					-

RQ4: What is the comparison between the results obtained on applying various ML algorithms (other than the BAs) and the results obtained on implementing BAs?

To show why BAs are so good compared to other ML algorithms, a comparison of results between different ML algorithms and the BAs has been made through the RQ mentioned above based on the RMSE, MMRE, and different Pred(m) values. Four different ML algorithms belonging to four different categories, i.e., tree-based models (DTs), neural network-based models (MLP), ensemble models (Bagging), and linear models (Elastic - Net) have been selected for carrying out this comparison. All the four models, along with a brief description, have been presented as follows:

- Decision Trees (DTs): DTs are a supervised and non-parametric ML algorithm for solving classification & regression problems. DTs' primary goal is to develop such predictive models where the response variable is predicted using the knowledge learned from various decision rules that have been inferred through the data attributes. Here, the rules are generated by breaking down the complex process of decision making into several simple decision rules which often provide us with easily interpretable solutions resembling the desired set of solutions [69]. DTs have several advantages, including DTs are easy to understand & interpret, require little or no data preparation, the computational cost is logarithmic to the number of training data points used in the tree, etc.
- Multi-layer Perceptron (MLP): MLP is again a supervised and neural network-based ML algorithm which learns the following function through training on the dataset,

$$f(\bullet): R^{in} \rightarrow R^{out} \tag{10}$$

where '*in*' corresponds to the number of input dimensions and '*out*' represents the number of output dimensions. For a given set of attributes, say, $X = x_1, x_2, \cdots, x_l$ and a response variable y, MLP can provide a non-linear approximation of the function for regression or a classification problem. MLP consists of one or more hidden non-linear layers between the two layers, i.e., input & output layer. MLPs are capable of learning in real-time, and they can learn non-linear models also. Particularly, in the case of regression, backpropagation has been used for implementing MLP with identity function being the activation function or having no activation function at all for the output layer [70]. Also, the square error is used as the loss function having the response variable as a collection of several continuous values.

- Bagging: Bagging is one of the ensemble ML methods that works by combining the predictions obtained from various base estimators built using a particular ML algorithm to improve the generalizability or robustness of the single estimator. Bagging belongs to the family of averaging methods out of the two prominent families of ensemble methods, i.e., the averaging methods and the boosting methods. The basic idea behind bagging [71] is to implement several independent base estimators (e.g., DTs, MLPs, etc.) over random subdivisions of the initial training set in the first instance and then taking out the average of each of the predictions to obtain a final prediction. Overall, the combined or aggregated bagging estimator is supposed to be better than the single estimators since the variance has been reduced.
- Elastic - Net (EN): EN [72] is a regularized linear ML algorithm for regression, which combines the penalties of two other linear models, i.e., the lasso & the ridge models, in a linear manner having *L1* & *L2* regularization, respectively. This aggregation encourages an efficient learning procedure, especially for the models having few non-zero weights like the lasso, with simultaneous maintenance of the properties of regularization for the ridge method. EN is advantageous in the case of multiple attributes being correlated to each other. However, the lasso is expected to select only one of them, that too randomly, whereas EN is expected to select both of them.

Further, all the ML models mentioned above have been implemented using similar procedures while implementing different BAs. All the algorithms have been implemented for seven open-source datasets (Abdera, Ivy, jEdit, jTDS, Log4j, Poi, & Rave) after pre-processing. Further, feature selection using the RFE algorithm and ten-fold cross-validation has also been performed. The performance of these models has been evaluated using the same performance measures, viz, RMSE, MMRE, Pred(0.25), Pred(0.30), & Pred(0.75) for comparison of the results so obtained with various prediction models that have been built using BAs. The results obtained on applying these four algorithms, i.e., DT, MLP, Bagging, and EN for all the datasets based on RMSE, MMRE, and Pred(m), have been provided in Tables XII, XIII, & XIV, respectively.

TABLE XII. RMSE Values for All the Datasets Using ML Algorithms

Accuracy Measure	**RMSE**						
Datasets	Abdera	Ivy	jEdit	jTDS	Log4j	Poi	Rave
ML Algorithm							
DT	1312.59	1719.58	71.55	**71.60**	198.66	1525.33	79.56
MLP	1203.93	1467.22	48.91	80.64	198.09	**1334.38**	61.76
Bagging	1215.03	1477.21	49.75	79.32	199.28	1452.23	61.72
Elastic - Net	**1174.68**	**1431.94**	**43.27**	73.84	**197.56**	1339.19	**55.29**

TABLE XIII. MMRE Values for All the Datasets Using ML Algorithms

Accuracy Measure	**MMRE**						
Datasets	Abdera	Ivy	jEdit	jTDS	Log4j	Poi	Rave
ML Algorithm							
DT	6.80	15.27	6.19	**0.98**	**9.00**	11.35	**3.15**
MLP	6.14	11.25	7.82	11.19	9.77	6.59	8.33
Bagging	6.58	11.75	8.12	10.43	10.25	11.03	8.20
Elastic - Net	**4.53**	**7.94**	**3.71**	6.33	11.36	**6.37**	3.92

On comparing Table VI and Table XII showing the RMSE values for seven different datasets using BAs and other ML algorithms, respectively, it is observed that BAs have performed better than the other ML algorithms. Precisely, the RMSE values obtained for Abdera, Ivy, and Poi datasets are lower and better using any of the five BAs compared to all the four other ML algorithms, i.e., DT, MLP, bagging, and EN. In the jEdit dataset, four out of the five BAs, i.e., AdaBoost, GBM, XGB, & CatBoost, performed better than three out of the four other ML algorithms, i.e., DT, MLP, & bagging in terms of RMSE. Further, for jTDS and Log4j datasets, three out of the five BAs, viz., AdaBoost, GBM, & LightGBM (i.e., 60% of the total BAs) show comparatively lower values of RMSE than all other ML algorithms. Lastly, in the Rave dataset, CatBoost BA outperformed all the other ML models with a lower value of RMSE, whereas AdaBoost, GBM, & XGB BAs outperformed DT, MLP, & bagging models. Overall, based on the RMSE values provided in Table VI and Table XII and on comparing the lowest RMSE values (values marked in bold) computed for each dataset in both the tables, BAs show a better performance since the lowest, and hence the best RMSE values have been obtained using BAs as compared to other ML algorithms for six (Abdera, Ivy, jTDS, Log4j, Poi, & Rave) out of the seven datasets, i.e., for 85.71% of the datasets. As an example, the least RMSE value equal to 71.60 obtained for the jTDS dataset on applying the DT algorithm reduces to 67.54 on applying GBM BA, leading to an improvement of 5.67%. Subsequently, on analyzing the mean RMSE values obtained for all the BAs taken together and also for all the other ML algorithms as shown in Fig. 5, it is concluded that the performance of BAs (having comparatively lower RMSE values) is better than other ML algorithms for all the seven datasets. An overall improvement in the mean RMSE values equal to 11.14%, 14.86%, 11.94%, 5.68%, 2.03%, 10.76%, and 6.95% for Abdera, Ivy, jEdit, jTDS, Log4j, Poi, and Rave datasets, respectively, has been achieved on applying BAs when compared to other ML algorithms.

Further, based on the MMRE values obtained for BAs and other ML algorithms presented in Table VII and Table XIII, it is evident that BAs performance is undoubtedly better than the other ML algorithms. Specifically, the lower MMRE values for three (AdaBoost, XGB, & CatBoost) out of the five BAs are better than all the four other ML algorithms, i.e., DT, MLP, bagging, & EN for both Abdera and jEdit

TABLE XIV. Pred(*m*) Values for All the Datasets Using ML Algorithms

Accuracy Measure	**Pred (m)**																				
Datasets	Abdera			Ivy			jEdit			jTDS			Log4j			Poi			Rave		
ML Algorithm	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)	(0.25)	(0.30)	(0.75)
DT	**0.20**	**0.25**	**0.62**	**0.17**	**0.20**	**0.54**	**0.20**	**0.21**	**0.52**	**0.31**	**0.31**	**0.61**	**0.24**	**0.26**	0.48	**0.25**	**0.28**	**0.59**	**0.20**	**0.23**	**0.46**
MLP	0.11	0.14	0.33	0.09	0.11	0.28	0.02	0.02	0.17	0.00	0.06	0.22	0.23	**0.26**	**0.61**	0.15	0.19	0.49	0.07	0.09	0.20
Bagging	0.10	0.12	0.30	0.09	0.11	0.26	0.01	0.01	0.15	0.00	0.06	0.25	0.22	0.24	0.59	0.10	0.12	0.28	0.07	0.09	0.20
Elastic - Net	0.14	0.17	0.39	0.11	0.14	0.38	0.10	0.14	0.30	0.14	0.14	0.28	0.19	0.25	0.57	0.13	0.16	0.43	0.11	0.15	0.36

datasets. Also, GBM and LightGBM BAs performed better than three (DT, MLP, & bagging) out of the four other ML algorithms for Abdera and jEdit datasets. Next, considering the Ivy and Log4j datasets, it is observed that lower and better MMRE values have been achieved for all the five BAs as compared to any of the other ML algorithms. In the jTDS dataset, XGB outperformed all other ML algorithms, whereas, rest of the four BAs outperformed three (MLP, bagging, & EN) out of the four other ML algorithms. Proceeding to the Poi dataset, XGB and CatBoost BAs provide lower MMRE values than any other ML algorithms. At the same time, AdaBoost and GBM BAs performed better than three (DT, MLP, & bagging) of the other ML algorithms. Lastly, on considering the Rave dataset, it is found that three out of the five BAs, viz. AdaBoost, XGB, & CatBoost BAs show better MMRE values than all other ML algorithms. However, GBM BA shows better performance than MLP and bagging algorithms, and it performs almost as good as the EN algorithm. Overall, on comparing the lowest MMRE values (values marked in bold) provided for each dataset in Table VII and Table XIII using BAs and other ML algorithms, respectively, it is observed that BAs showcase a better performance due to the lowest and the best-obtained MMRE values for all the seven open-source datasets, .i.e. for 100% of the datasets. As an example, the least MMRE value equal to 9.00 obtained for the Log4j dataset on applying the DT algorithm reduces to 3.82 on applying XGB BA, leading to an improvement of 57.56%. Not only this, the mean MMRE values calculated for all the BAs taken together and for all the other ML algorithms have been depicted in Fig. 6. It is evident from Fig. 6 that lower MMRE values have been obtained using BAs for all the seven datasets considered in this study which, further strengthens the conclusion stating that BAs are better than other ML algorithms for SMP. An overall improvement in the mean MMRE values equal to 38.10%, 47.62%, 55.11%, 67.36%, 29.31%, 39.48%, and 48.81% for Abdera, Ivy, jEdit, jTDS, Log4j, Poi, and Rave datasets, respectively, has been achieved on applying BAs when compared to other ML algorithms.

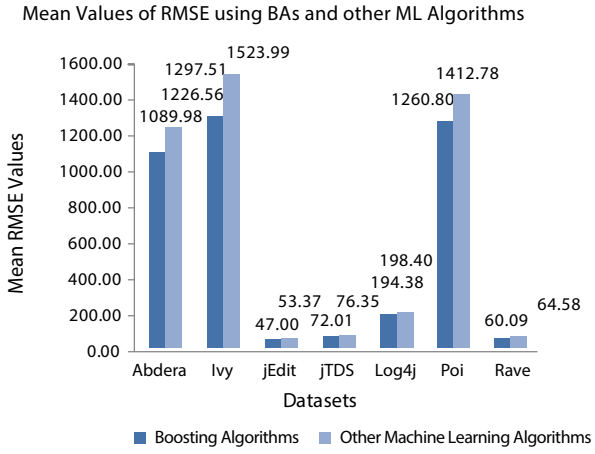

Fig. 5. Mean values of RMSE using BAs and other ML algorithms.

Subsequently, Table VIII and Table XIV present the Pred(m) values at 25%, 30%, and 75% using BAs and other ML algorithms. A comparison between these two tables also indicates BAs supremacy over other ML algorithms while predicting maintainability. Overall, comparing the best. i.e., the highest values (values marked in bold) obtained for Pred(0.25), Pred(0.30), & Pred(0.75) in both the tables, it is observed that these values are better using BAs than other ML algorithms for four out of the seven datasets, i.e., for 57.14% of the datasets (which is more than half). Further, for Poi and Rave datasets, better Pred(0.75) values have been obtained using BAs. Hence, it is clear that BAs perform better than other ML algorithms based on the Pred(m) values.

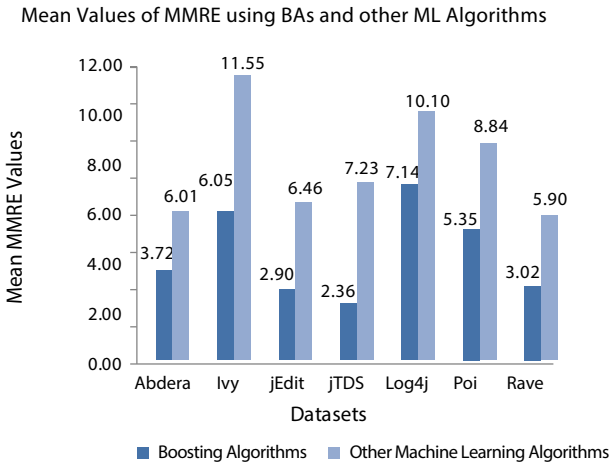

Fig. 6. Mean values of MMRE using BAs and other ML algorithms.

Therefore, on the whole, it is concluded that BAs are good performers and indeed better than other ML algorithms. All the above analysis and comparison made under this RQ, based on the RMSE, MMRE, & Pred(m) values obtained on applying BAs and other ML algorithms to each of the seven open-source datasets, further support the supremacy of BAs over other algorithms.

Conclusively, this research work can further benefit the society, especially the software engineers, in predicting the maintainability of the software being developed well in advance, thereby reducing the overall software development costs. This reduction in overall cost is mainly attributed to reducing the maintenance cost in particular, which gets accumulated with each phase of SDLC if not taken care of. The growing demand for different software in society over the last few years due to the automation of several tasks has led to a surge in the design & development of various software systems in the software industry. However, these systems require to be maintained once they are delivered to the customer involving high costs. Therefore, a great deal of specific techniques or mechanisms is needed to bring down these high costs. This can only be done by estimating the software's maintenance effort in the initial phases of development using some prediction models that can predict the software's maintainability in good time with high precision. The current research would help developers achieve this goal of predicting maintainability by utilizing different SMP models developed using various BAs, as proposed in this study. These models not only help in the task of predicting maintainability but also outperform several other models available for predictive modeling.

V. Threats to Validity

While conducting the current empirical study, certain potential threats to validity were encountered. This study has been performed on various open-source datasets, limiting its use and does not ascertain its applicability for various other types of software available in the industry for its generalization. However, a sincere attempt has been made to overcome this threat by using 10-fold cross-validation & applying all the five BAs over each of the seven datasets with different characteristics. The results obtained are possibly less biased and can further be generalized. Also, while developing prediction models using various BAs, hyper-parameter tuning of function parameters has not been performed. The default settings have been mainly used, which again becomes a limitation of this study since the results so obtained may be correct only to a first approximation. Apart from this, three of the most common threats to validity existent in any empirical study are presented below.

Internal validity refers to an extent to which conclusions of an empirical study can support the claim for cause & effect, i.e., the independent & the dependent variables. An attempt has been made to minimize this effect by applying feature selection using the RFE algorithm and using only the selected variables to study their effect on maintainability.

External validity is the extent of the generalizability of the outcomes or the results of any empirical study. A set of seven open-source datasets with different size, characteristics, and maintenance requirements has been used in this study to minimize this effect.

Construct validity is the quality of choice of various independent and dependent variables of a study, as this choice undoubtedly impacts the results of that study. So, the threat to construct validity arises from the choice of these independent and dependent variables. A set of seventeen OO metrics from different suites proposed by various researchers, namely Chidamber & Kemerer [47], Henderson-Sellers [49], and Bansiya & Davis [50] has been selected to minimize this threat rather than adhering to a particular metric suite.

VI. Conclusion & Future Direction

The current study's main objective was to analyze various ML based BAs for SMP using open-source datasets. An extensive analysis and comparison of five different BAs (AdaBoost, GBM, XGB, LightGBM, and CatBoost) were conducted using each of the seven empirically collected open-source datasets (Abdera, Ivy, jEdit, jTDS, Log4j, Poi, & Rave) to predict maintainability. Seventeen different OO metrics were selected from three different metrics suites to develop the prediction models. Feature selection using the RFE algorithm and cross-validation using the ten-fold cross-validation technique was also performed. Performance of various BAs was evaluated using RMSE, MMRE, Pred(0.25), Pred(0.30) & Pred(0.75) as the prediction accuracy measures. Further, to determine if a significant difference exists between different BAs performances & finding their mean ranks, a non-parametric statistical test named the Friedman test was conducted. Afterward, a post hoc analysis using an advanced statistical test named the Nemenyi test was also performed to identify if the difference in various BAs performance, if it exists, is statistically significant or not. Lastly, a comparison was made between the results obtained for SMP using the BAs and the results obtained on applying four other ML algorithms (DT, MLP, bagging, and EN). The major findings of the current study are as presented below.

- A reduction in features equal to 52.94% is achieved after feature selection using the RFE algorithm.
- While calculating residual errors for all the datasets using RMSE and MMRE as the accuracy measures, it was found that in the case of RMSE, GBM performed the best, followed by LightGBM, whereas, in the case of MMRE, XGB performed the best.
- Prediction accuracies also confirm the use of BAs for SMP, particularly Pred(0.75), where XGB stood out to be the best performer with a fairly reasonable predictive ability for six out of seven datasets, i.e., for 85.71% of the datasets, ranging from 51% to 79%.
- The Friedman test results and post hoc analysis using the Nemenyi test further unfolded the superiority of XGB and CatBoost BAs over other selected BAs in the study for SMP using open-source datasets.
- The comparison between the results obtained for SMP using BAs and other ML algorithms revealed that BAs are indeed the better performers than other algorithms based on all the measures of accuracy considered in this study.

Hence, prediction models developed using various BAs from the family of ML algorithms can indeed be implemented for SMP using open-source datasets. However, this is a limited implementation of the proposed study.

More research and studies can be planned in the future to implement the algorithms used in this study in isolation or in combination with other ML techniques for different types of software systems available in the industry, which are written in different programming languages to generalize the results of this study further. Different paradigms and models, more feature selection, dimensionality reduction, ensemble, and re-sampling techniques, can be considered while conducting future studies. Also, while developing prediction models in the future using the proposed algorithms, hyper-parameter tuning of different function parameters can be done as an extension to the current work.

Acknowledgment

This research work has been supported by the O/o Director (Research & Consultancy), GGSIPU under the FRGS scheme through the project entitled, "Determination of Optimum Refactoring Sequence after Prioritization of Classes on the basis of their bad smell," dt. 03.05.2019, Ref. No. GGSIPU/DRC/FRGS/ 2019/1553/62.

References

[1] "IEEE Standard for Software Maintenance," *IEEE Std 1219-1993*, 1993, doi: 10.1109/IEEESTD.1993.11557.

[2] "ISO/IEC 25010:2011(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," 2011. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en.

[3] M. A. Ahmed and H. A. Al-Jamimi, "Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model," *IET software*, vol. 7, no. 6, pp. 317–326, 2013, doi: 10.1049/iet-sen.2013.0046.

[4] N. Zighed, N. Bounour, and A.-D. Seriai, "Comparative Analysis of Object-Oriented Software Maintainability Prediction Models," *Foundations of Computing and Decision Sciences*, vol. 43, no. 4, pp. 359–374, 2018, doi: 10.1515/fcds-2018-0018.

[5] H. Alsolai and M. Roper, "Application of Ensemble Techniques in Predicting Object-Oriented Software Maintainability," in *Proceedings of the Evaluation and Assessment on Software Engineering*, 2019, pp. 370–373, doi: 10.1145/3319008.3319716.

[6] L. Kumar, D. K. Naik, and S. K. Rath, "Validating the effectiveness of object-oriented metrics for predicting maintainability," *Procedia Computer Science*, vol. 57, pp. 798–806, 2015, doi: 10.1016/j.procs.2015.07.479.

[7] R. Malhotra and A. Chug, "Application of Group Method of Data Handling model for software maintainability prediction using object oriented systems," *International Journal of System Assurance Engineering and Management*, vol. 5, pp. 165–173, 2014, doi: 10.1007/s13198-014-0227-4.

[8] H. Alsolai, "Predicting Software Maintainability in Object-Oriented Systems Using Ensemble Techniques," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 716–721, doi: 10.1109/ICSME.2018.00088.

[9] A. Chug and R. Malhotra, "Benchmarking framework for maintainability prediction of open source software using object oriented metrics," *International Journal of Innovative Computing, Information and Control*, vol. 12, no. 2, pp. 615–634, 2016.

[10] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994, doi: 10.1109/32.295895.

[11] R. Malhotra and A. Chug, "Software Maintainability: Systematic Literature Review and Current Trends," *International Journal of Software Engineering and Knowledge Engineering*, vol. 26, no. 8, pp. 1221–1253, 2016, doi: 10.1142/S0218194016500431.

[12] D. Michie, D. J. Spiegelhalter, C. C. Taylor, and others, "Machine learning," *Neural and Statistical Classification*, vol. 13, no. 1994, pp. 1–298, 1994.

[13] M. Sharma, S. Sharma, and G. Singh, "Performance analysis of statistical

and supervised learning techniques in stock data mining," *Data*, vol. 3, no. 4, p. 54, 2018, doi: 10.3390/data3040054.

[14] X. Zhong and D. Enke, "Predicting the daily return direction of the stock market using hybrid machine learning algorithms," *Financial Innovation*, vol. 5, no. 1, p. 4, 2019, doi: 10.1186/s40854-019-0138-0.

[15] K. C. Rasekhschaffe and R. C. Jones, "Machine learning for stock selection," *Financial Analysts Journal*, vol. 75, no. 3, pp. 70–88, 2019, doi: 10.1080/0015198X.2019.1596678.

[16] P. Kaur and M. Sharma, "Diagnosis of human psychological disorders using supervised learning and nature-inspired computing techniques: a meta-analysis," *Journal of medical systems*, vol. 43, no. 7, p. 204, 2019, doi: 10.1007/s10916-019-1341-2.

[17] G. T. Reddy, M. P. K. Reddy, K. Lakshmanna, D. S. Rajput, R. Kaluri, and G. Srivastava, "Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis," *Evolutionary Intelligence*, vol. 13, no. 2, pp. 185–196, 2020, doi: 10.1007/s12065-019-00327-1.

[18] M. Sharma and P. Kaur, "A Comprehensive Analysis of Nature-Inspired Meta-Heuristic Techniques for Feature Selection Problem," *Archives of Computational Methods in Engineering*, pp. 1–25, 2020, doi: 10.1007/s11831-020-09412-6.

[19] X. Ma and S. Lv, "Financial credit risk prediction in internet finance driven by machine learning," *Neural Computing and Applications*, vol. 31, no. 12, pp. 8359–8367, 2019, doi: 10.1007/s00521-018-3963-6.

[20] H. Ghoddusi, G. G. Creamer, and N. Rafizadeh, "Machine learning in energy economics and finance: A review," *Energy Economics*, vol. 81, pp. 709–727, 2019, doi: 10.1016/j.eneco.2019.05.006.

[21] S. K. Dubey, A. Rana, and Y. Dash, "Maintainability prediction of object-oriented software system by multilayer perceptron model," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 5, pp. 1–4, 2012, doi: 10.1145/2347696.2347703.

[22] R. Malhotra and A. Chug, "Application of evolutionary algorithms for software maintainability prediction using object-oriented metrics," in *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*, 2014, pp. 348–351, doi: 10.4108/icst.bict.2014.258044.

[23] L. Kumar and S. K. Rath, "Hybrid functional link artificial neural network approach for predicting maintainability of object-oriented software," *Journal of Systems and Software*, vol. 121, pp. 170–190, 2016, doi: 10.1016/j.jss.2016.01.003.

[24] M. O. Elish, H. Aljamaan, and I. Ahmad, "Three empirical studies on predicting software maintainability using ensemble methods," *Soft Computing*, vol. 19, no. 9, pp. 2511–2524, 2015, doi: 10.1007/s00500-014-1576-2.

[25] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4537–4543, 2010, doi: 10.1016/j.eswa.2009.12.056.

[26] E. O. Costa, G. A. de Souza, A. T. R. Pozo, and S. R. Vergilio, "Exploring genetic programming and boosting techniques to model software reliability," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 422–434, 2007, doi: 10.1109/TR.2007.903269.

[27] M. Akour, I. Alsmadi, and I. Alazzam, "Software fault proneness prediction: a comparative study between bagging, boosting, and stacking ensemble and base learner methods," *International Journal of Data Analysis Techniques and Strategies*, vol. 9, no. 1, pp. 1–16, 2017, doi: 10.1504/IJDATS.2017.10003991.

[28] Y. Freund, "Boosting a weak learning algorithm by majority," *Information and computation*, vol. 121, no. 2, pp. 256–285, 1995, doi: 10.1006/inco.1995.1136.

[29] D. Nielsen, "Tree boosting with xgboost-why does xgboost win' every' machine learning competition?," NTNU, 2016.

[30] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

[31] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of systems and software*, vol. 23, no. 2, pp. 111–122, 1993, doi: 10.1016/0164-1212(93)90077-B.

[32] M. Dagpinar and J. H. Jahnke, "Predicting maintainability with object-oriented metrics-an empirical comparison," in *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.*, 2003, pp. 155–164, doi: 10.1109/WCRE.2003.1287246.

[33] M. M. T. Thwin and T.-S. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," *Journal of systems and software*, vol. 76, no. 2, pp. 147–156, 2005, doi: 10.1016/j.jss.2004.05.001.

[34] C. Van Koten and A. R. Gray, "An application of Bayesian network for predicting object-oriented software maintainability," *Information and Software Technology*, vol. 48, no. 1, pp. 59–67, 2006, doi: 10.1016/j.infsof.2005.03.002.

[35] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Application of artificial neural network for predicting maintainability using object-oriented metrics," *Transactions on Engineering, Computing and Technology*, vol. 15, pp. 285–289, 2006, doi: 10.5281/zenodo.1058483.

[36] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *Journal of systems and software*, vol. 80, no. 8, pp. 1349–1361, 2007, doi: 10.1016/j.jss.2006.10.049.

[37] M. O. Elish and K. O. Elish, "Application of treenet in predicting object-oriented software maintainability: A comparative study," in *2009 13th European Conference on Software Maintenance and Reengineering*, 2009, pp. 69–78, doi: 10.1109/CSMR.2009.57.

[38] A. Kaur, K. Kaur, and R. Malhotra, "Soft computing approaches for prediction of software maintenance effort," *International Journal of Computer Applications*, vol. 1, no. 16, pp. 69–75, 2010, doi: 10.5120/339-515.

[39] R. Malhotra[1] and A. Chug[2], "Software Maintainability Prediction using Machine Learning Algorithms," *Software engineering: an international Journal (SeiJ)*, vol. 2, no. 2, pp. 19–36, 2012.

[40] L. Kumar and S. K. Rath, "Software maintainability prediction using hybrid neural network and fuzzy logic approach with parallel computing concept," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 2, pp. 1487–1502, 2017, doi: 10.1007/s13198-017-0618-4.

[41] N. Baskar and C. Chandrasekar, "An Evolving Neuro-PSO-based Software Maintainability Prediction," *International Journal of Computer Applications*, 2018, doi: 10.5120/ijca2018916305.

[42] S. Jha *et al.*, "Deep learning approach for software maintainability metrics prediction," *Ieee Access*, vol. 7, pp. 61840–61855, 2019, doi: 10.1109/ACCESS.2019.2913349.

[43] X. Wang, A. Gegov, F. Arabikhan, Y. Chen, and Q. Hu, "Fuzzy network based framework for software maintainability prediction," *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems*, vol. 27, no. 5, pp. 841–862, 2019, doi: 10.1142/S0218488519500375.

[44] S. Gupta and A. Chug, "Assessing Cross-Project Technique for Software Maintainability Prediction," in *Procedia Computer Science*, 2020, vol. 167, pp. 656–665, doi: 10.1016/j.procs.2020.03.332.

[45] S. Gupta and A. Chug, "Software maintainability prediction using an enhanced random forest algorithm," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 23, no. 2, pp. 441–449, 2020, doi: 10.1080/09720529.2020.1728898.

[46] S. Gupta and A. Chug, "Software maintainability prediction of open source datasets using least squares support vector machines," *Journal of Statistics and Management Systems*, vol. 23, no. 6, pp. 1011–1021, 2020, doi: 10.1080/09720510.2020.1799501.

[47] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," 1991, doi: 10.1145/118014.117970.

[48] R. Malhotra and A. Chug, "An empirical study to redefine the relationship between software design metrics and maintainability in high data intensive applications," in *Proceedings of the World Congress on Engineering and Computer Science*, 2013, vol. 1.

[49] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

[50] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on software engineering*, vol. 28, no. 1, pp. 4–17, 2002, doi: 10.1109/32.979986.

[51] "MinMaxScaler Link." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (accessed Dec. 14, 2019).

[52] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1–2, pp. 273–324, 1997, doi: 10.1016/S0004-3702(97)00043-X.

[53] "RFE Documentation." https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html (accessed Dec. 16, 2019).

[54] K. T. Khaing, "Enhanced Features Ranking and Selection using Recursive Feature Elimination (RFE) and k-Nearest Neighbor Algorithms in Support Vector Machine for Intrusion Detection System," *International Journal of Network and Mobile Technologies*, vol. 1, no. 1, pp. 1832–6758, 2010.

[55] Y. Freund, R. E. Schapire, and others, "Experiments with a new boosting algorithm," in *Thirteenth International Conference on International Conference on Machine Learning (ICML'96)*, 1996, vol. 96, pp. 148–156.

[56] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997, doi: 10.1006/jcss.1997.1504.

[57] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002, doi: 10.1016/S0167-9473(01)00065-2.

[58] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001, doi: 10.1214/aos/1013203451.

[59] G. Ke *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146–3154.

[60] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems*, 2018, pp. 6638–6648.

[61] R. Kohavi and others, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995, vol. 14, no. 2, pp. 1137–1145.

[62] S. D. Conte, H. E. Dunsmore, and Y. E. Shen, *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA, 1986.

[63] B. A. Kitchenham, S. MacDonell, L. Pickard, and M. Shepperd, "Assessing prediction systems," *The Information Science Discussion Paper Series, University of Otago*, vol. 99/14, 1999, [Online]. Available: http://hdl.handle.net/10523/1015.

[64] B. A. Kitchenham, L. M. Pickard, S. G. MacDonell, and M. J. Shepperd, "What accuracy statistics really measure," *IEE Proceedings-Software*, vol. 148, no. 3, pp. 81–85, 2001, doi: 10.1049/ip-sen:20010506.

[65] B. Iglewicz, "Robust scale estimators and confidence intervals for location," *Understanding robust and exploratory data analysis*, p. 405431, 1983.

[66] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940, doi: 10.1214/aoms/1177731944.

[67] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008, doi: 10.1109/TSE.2008.35.

[68] [J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[69] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991, doi: 10.1109/21.97458.

[70] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, classification," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992, doi: 10.1109/72.159058.

[71] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996, doi: 10.1023/A:1018054314350.

[72] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005, doi: 10.1111/j.1467-9868.2005.00527.x.

Shikha Gupta

Shikha Gupta received her Master's degree in Computer Applications at Indira Gandhi National Open University (IGNOU), New Delhi, India in December 2017. She has been awarded the University Gold Medal for securing first position in order of merit and CEMCA Award 2019 for being the Best Female Student in the Master Degree Programme. Currently, she is pursuing her PhD from University School of Information, Communication & Technology, Guru Gobind Singh Indraprastha University, New Delhi, India since September 2018. Her research interests include software engineering, data mining and machine learning.

Anuradha Chug

Dr. Anuradha Chug has long teaching experience of almost 30 years to her credit as faculty and in administration at various educational institutions in India. She has worked as guest faculty in Netaji Subhash Institute of Information and Technology, Dwarka, New Delhi and Regular Faculty at Government Engineering College, Bikaner. Before picking the current assignment as Assistant Professor at USICT, GGSIP University, she has also worked as Academic Head, Aptech, Meerut and Program Coordinator at Regional Centre, Indira Gandhi National Open University (IGNOU), Meerut. In academics, she has earned her doctorate degree in Software Engineering from the Delhi Technological University, Delhi, India. Before pursuing PhD, she has achieved top rank in her M.Tech (IT) degree and conferred the University Gold Medal in 2006 from Guru Gobind Singh Indraprastha University. Previously she has acquired her Master's degree in Computer Science from Banasthali Vidyapith, Rajasthan in the year 1993. Her H-index as reported by Google Scholar is 12. She has published more than 50 research papers in international and national journals and conferences. She has also served as reviewer of several national and international journals and conferences in the area of software engineering (ACM transaction, IJKESE, FOCS, IEEE Access, Neurocomputing, Informatica, Inderscience, etc). She is also the recipient of DST funded project entitled "Application of Internet of Things (IoT) in Agriculture Sector" as CO-PI. She is passionate to design, develop and deploy IoT applications in various aspects of human life. She has delivered many talks featuring Data Mining, IoT, Mining Software Repositories at various FDPs, student talks, etc.