International Journal of Innovative Computing, Information and Control
Volume **12**, Number **2**, April **2016**

ICIC International ©2016 ISSN 1349-4198
pp. **615–634**

BENCHMARKING FRAMEWORK FOR MAINTAINABILITY PREDICTION OF OPEN SOURCE SOFTWARE USING OBJECT ORIENTED METRICS

Anuradha Chug[1] and Ruchika Malhotra[2]

[1]University School of Information and Communication Technology
Guru Gobind Singh Indraprastha University
Sector 16-C, Dwarka, New Delhi 110077, India
anuradha@ipu.ac.in

[2]Department of Software Engineering
Delhi Technological University
Bawana Road, Delhi 110042, India
ruchikamalhotra2004@yahoo.com

Received October 2015; revised January 2016

Abstract. *Software maintainability is measured as the ease with which the existing software could be modified and often predicted during the development stage on the basis of some measurable design characteristics. Controlling the software maintainability and understandability of any open source software (OSS) system is extremely challenging because it is written and constantly modified by the developers located all over the world. The current study analyzes the effectiveness of machine learning (ML) techniques for the maintainability prediction of OSS systems. In this work large-scale empirical comparisons of thirteen classifiers over seven open source datasets were conducted followed by extensive statistical tests and post hoc analysis to establish the confidence on the performance of one ML technique over another. The results show two important findings: firstly, we observed that overall good prediction accuracy is achieved by almost all ML techniques; secondly the prediction models using genetically adaptive learning ML technique and group method of data handling (GMDH) technique perform better than the other ML techniques in the context of OSS systems. The outcome of this investigation would be helpful for developers in order to predict maintenance behavior of the software at the earlier stages of software development lifecycle (SDLC). Accordingly, they can optimize their resource allocations, prioritize maintenance tasks and produce high-quality low maintenance software systems. Additionally, it also has numerous other applications such as schedule planning, cost estimation, quality assurance testing, software debugging, budget preparation, and software performance optimization.*

Keywords: Empirical validation, Software maintainability prediction, Object-oriented metrics, Open source software, Friedman test, Post hoc analysis, Feature subselection

1. Introduction.

Software maintenance is an important phase in software development life cycle (SDLC) as it plays a determinant role in finding the total project cost of any software [1]. Maintainability of the software cannot be measured until the software system reaches to the operational phase; however, by then it would be too late to rectify and optimize the quality. In this regard, it has become increasingly important to develop maintenance prediction models to assess accurate maintainability during the early phases of the SDLC. This can be done with the help of some measurable software design characteristics such as cohesion, coupling, abstraction, complexity and inheritance [2,3]. In open source software (OSS), practitioners across the globe are allowed to change, expand and redistribute the newly created version without any requirement of the license [4]. Changes in OSS are made continuously in order to remove defects, improve functionalities, and increase usefulness [5]. Estimating the maintainability of OSS becomes more challenging due to the lack of technical support and the absence of adequate documentation. Although various maintainability prediction models using statistical and machine learning (ML) techniques have been developed in past [6-25], to the best of authors' knowledge, studies on observing the maintainability of OSS systems are very limited except one conducted by Zhou and Xu [16]. Even though Ramil et al. [17] have compiled many empirical studies on OSS, all the studies focused on intuitively judging the software maintainability instead of creating a mathematical prediction model. Myrtveit et al. [18] have also raised an important issue that more reliable research procedures must be developed before believing on the outcome of any one of the prediction models. In order to address these issues an effort has been made in this study to find the answer of the following three research questions:

- **RQ1**: Does the impact of object-oriented (OO) metrics on maintainability exist in the context of OSS?
- **RQ2**: What is the comparative performance of ML techniques for maintainability prediction using OSS?
- **RQ3**: Which pairs of ML techniques perform significantly different from each other in terms of prediction accuracy measures?

We extensively compare the experimental results of thirteen ML classifiers over seven OSS using statistical test followed by post hoc analysis to scrutinize if there exists a significant difference among the performance of any particular ML technique. The thirteen selected classifiers include Linear Regression (LR), M5Rules, Decision Tree (DT), Support Vector Machine (SVM), KStar, Bagging, Jordan Elman Recurrent Network (JERN), Back Propagation Network (BPN), Kohonen Network (KN), Probabilistic Neural Network (PNN), Group Method of Data Handling (GMDH), General Regression Neural Networks (GRNN), and GRNN with Genetic Adaptive Learning (GGAL). The source code of seven OSS Drumkit, OpenCV, Abdera, Ivy, Log4j, JEdit and JUnit is obtained from http://sourceforge.net and https://apache.org to carry out this widespread investigation.

The rest of the paper is organized as follows. Section 2 summarizes the related work and Section 3 describes the independent variables, dependent variables and the process of empirical data collection. Section 4 presents research methodology adopted in the current study. Section 5 describes the results and analysis. Threats to validity are discussed in Section 6 and finally Section 7 concludes the paper with future directions.

2. Related Work.

Predicting the maintainability during early phases of SDLC helps in better planning and appropriate allocation of resources for reducing maintenance effort of the software and therefore remained the prime subject of research. Extensive research has been carried out in the past to empirically prove that software metrics can be used to determine maintainability of a software at early stages of the SDLC by various statistical and ML techniques [6-15]. Various information about the studies is compiled in Table 1, such as the type of the dataset used, validation method, ML technique, statistical test, and performance measures.

Li and Henry [8] attempted as early as in the year 1993 to validate linear regression model using two proprietary datasets, User Interface Management System (UIMS) and Quality Evaluation System (QUES) for evaluating the relationship between Chidamber and Kemerer (C&K) [26] metrics suite and maintainability. The results indicated that a total of 90% and 87% total variance in maintenance effort is accounted by C&K metrics for UIMS and QUES systems respectively. Koten and Gray [12] further validated Bayesian Belief Network using 10-cross validation on the same dataset and found it to

TABLE 1. Salient details of prediction models used by researchers in software maintainability

Study	Dataset	Metric suite	Prediction Model	Validation Method	Prediction Accuracy Measure
Li and Henry [8]	UIMS and QUES dataset	C&K metric suite	Multiple linear regression (MLR)		
Koten and Gray [12]	UIMS and QUES dataset [8]	DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM, Size1, Size2	Linear Rgression, Bayesian Belief Network (BBN)	10-cross Validation	Absolute RES, MRE, MMRE, Pred(q)
Elish and Elish [13]	UIMS and QUES dataset [8]	C&K metric suite	Tree Nets classifier	Leave-one-out cross Validation	MRE, MMRE, Pred(q), Overestimate, Underestimate
Jin and Liu [14]	Programs developed by the students in C++	LCOM, NOC, DIT, WMC, RFC, DAC, MPC, NOM	Support Vector Machine (SVM)	none	MARE, MRE, p-value, r
Kaur et al. [15]	UIMS and QUES dataset [8]	LCOM, DIT, WMC, NOC, RFC, DAC, MPC, NOM	ANN, Fuzzy Inference Systems (FIS) and Adaptive Neuro FIS (ANFIS)	Hold-out	MARE, MRE, p-value, r
Zhou and Xu [16]	Open Source Software	NPAVGC, OSAVG, CSAO, CSA, SDIT, SLCOM, SRFC, SWMC, SNOC, MHF, POF, NCLASS, MNETH, PDIT	Univariate and Multivariate Regression Analysis	Leave-one-out cross Validation	R2, p-value, Std. Error
Zhou and Leung [19]	UIMS and QUES dataset [8]	WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, Size1, Size2	MLR, ANN, RT, SVM, MARS	Leave-one-out cross Validation	RES, ARE, MRE, MMRE
Malhotra and Chug [21]	Proprietary systems namely FLM, and EASY	WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, Size1, Size2	GRNN, FF3LBPNN, and GMDH	Hold-out	MRE, MMRE, Pred(q), Overestimate, Underestimate
Malhotra and Chug [23]	Proprietary systems namely FLM, EASY, SMS, IMS and ABP System	WMC, DIT, NOC, CBO, RFC, LCOM, SCCR, NODBC, MI, Cyclomatic Complexity	GMDH	10 fold cross validation	MRE, MMRE, Pred(q)
Malhotra and Chug [24]	UIMS and QUES dataset [8]	WMC, DIT, NOC, RFC, LCOM, MPC, DAC, NOM, Size1, Size2	GMDH, GA, PNN	Hold-out	MRE, MMRE, Pred(q), R-Square, p-value

be significantly better model in terms of precise prediction accuracy. Elish and Elish [13] corroborate relatively new Technique, TreeNets for software maintainability Predictions. They compared their results with other prevalent models and found them to be least cost effective with more prediction accuracy on UIMS and QUES datasets. The results were analyzed using various prediction accuracy measures such as Magnitude of Relative Error (MRE), Mean Magnitude of Relative Error (MMRE) and Prediction accuracy with less than 25% error (Pred 0.25).

Further, in the year 2010, Kaur et al. [15] conducted the study and analyzed the prediction capability of the Adaptive Neuro Fuzzy Inference System (ANFIS) technique on UIMS and QUES dataset using Hold-out validation and the outcome showed better performance as compared to previous studies. In another study conducted by Malhotra and Chug [21], the predicted capability of the Group Method of Data Handling (GMDH) technique was analyzed using the UIMS and QUES dataset using MRE, MMRE, Pred(0.25) and Mean Absolute Relative Error (MARE) measures. The results indicated that the hybrid models such as GMDH has enhanced capabilities to capture the design characteristics using C&K metrics and hence results were found to be more precise. Dagpinar and Jhanke [10] suggested that instead of designing level metrics of structure languages, OO metrics should be used for precise capturing while making any prediction model. Further, they also recorded the significant impact of direct coupling metric and size metric on software maintainability instead of other metrics such as cohesion, inheritance and indirect coupling. Aggarwal et al. [25] in his study suggested that since maintainability is very subjective in nature, the use of the fuzzy model for its measurement would be more appropriate.

Overall, the studies reported on the subject reveal that C&K metric suite [26] is quite popular and used extensively in most studies [10-13,19,21] because of its research focus and capability of capturing the OO metrics more accurately. The researchers are always constrained against non-availability of genuine datasets to conduct their validation studies and test newer prediction models of determining maintainability. However, the datasets of two proprietary software systems comprising of UIMS and QUES made public by Li and Henry [8] opened the doors for additional research studies to validate the maintainability prediction models and therefore majority of researchers used the datasets in their experiments [10-13,19,21].

We also found that in two research studies [14,20], datasets developed from the students' software was used to validate their prediction models. In the first study, Jun and Liu [14] validated their prediction model using the datasets collected from the software systems developed by graduate students. Their results show that when Support Vector Machine (SVM) is combined with clustering for the purpose of maintenance effort predictions, correlation between C&K metric suite and maintainability was found to be as high as 0.769 which is statistically quite significant. In the second study, Misra [20] used the datasets collected from a pool of 50 C++ programmes and deployed them in LR classifier as well as intuitive analysis based model using more than twenty design and code measures.

Although many maintainability prediction models have been proposed in the literature, almost all of them have used either the traditional dataset given by Li and Henry [8] in the year 1993 or used the students' programmes to compare the performance of ML techniques. The only exception reported in literature on OSS was by Zhou and Xu [16] but they analyze only statistical regression based model for the correct estimation of maintainability. As the interest for OSS has been rising across the globe, a powerful customized ML technique based prediction model for OSS was seen as the potential scope of research. In order to get unbiased, accurate and repeatable maintainability prediction model for OSS, the current study attempts to create an empirical framework using 13

ML techniques. Outcome of the current research has been further assessed over various releases of seven OSS comprising of Drumkit, OpenCV, Abdera, Ivy, Log4j, JEdit and JUnit available at http://www.github.com and http://www.sourceforge.net repositories.

3. Research Background.

This section presents the selection of dependent and independent variables and subsequently the process of collecting the empirical data for the validation of ML techniques. Our goal was to capture the various attributes of OO paradigm such as size, coupling, cohesion, abstraction, complexity and inheritance. In the current study, we have used C&K metric suite which is very common and used by various researchers [10-13,19,21]. In one of our previous studies [23] we realized few shortcomings in C&K metric suite such as it does not take into account the structural complexity of the software and any metric on account of the amount of database handling. In order to overcome such shortcomings, in addition to C&K metric suite, we have also included the metric suites proposed by Henderson-Sellers [27] and Bansiya and Davis [28].

The dependent variable is 'Change' defined as the number of changes in source code in terms of addition, deletion or modification in the lines of code. In Subsection 3.1, independent variables are explained, in Subsection 3.2, we have discussed the dependent variable and Subsection 3.3 give details of the process of empirical data collection.

3.1. Independent variables.

Various OO metrics were carefully selected from the metric suites proposed by researchers [26-28] to capture all the design attributes such as coupling, cohesion, inheritance, abstraction and complexity of OO paradigm as summarized in Table 2.

3.2. Dependent variables.

The dependent variable in the current study is maintenance effort measured by observing the number of changes made between two consecutive versions and it is counted in terms of a number of lines of source code added, deleted or modified in the newer version with respect to the older version for each class. We identify the common classes between two consecutive versions and compute the exact number of lines where change is performed. Any addition or deletion of a line in the current version with respect to an older version is counted as one change whereas any modification of the line is counted as two changes. The value of each OO metric for each class is calculated and combined with the respective changes made into that class to generate the data points. Same approach is adopted by many researchers [6-15,19,21-24].

3.3. Empirical data collection.

We explore open source repositories for collecting the empirical data keeping in mind two important characteristics which include that it should follow OO paradigm and it should have a high number of downloads in recent times (last 12 months) as it is a clear indication that there are active users contributing constantly.

The details of the selected OSS systems in terms of versions, release date, size, number of classes, etc. are summarized in Table 3 and their functioning in brief are explained as below:

- Drumkit is a Java Mobile based game on JAVA-JME platform. (https://github.com/nokia-developer/drumkit-jme)
- OpenCV stands for Open Source Computer Library designed for providing computational efficiency with a main focus on real-time applications. (http://sourceforge.net/projects/opencvlibrary)
- Abdera is an open source Atom parser generator used for client and server to build high-performance functionality of Internet by producing high quality designed documents. (https://git.apache.org/abdera)

- Ivy is a set of open source libraries and programs that allow applications to broadcast information through text messages, with a subscription mechanism based on regular expressions. (https://git.apache.org/ivy)
- Log4j allows the developer to control which log statements are output with arbitrary granularity. It is fully configurable at runtime using external configuration files. (https://git.apache.org/Log4j)
- JEdit is a highly customizable text editor written in Java and runs on any operating system. It can be extended with macros written in scripting languages. (https://jedit.svn.sourceforge.net/svnroot/jedit)

Table 2. Independent variables

Metrics	Ref	Definition
WMC (Weighted Methods per Class)	[26]	It counts the sum of McCabe's Cyclomatic complexities of all local methods in a class.
DIT (Depth of Inheritance Tree)	[26]	It calculates the depth of the said class in inheritance from the root class.
NOC (Number of Children)	[26]	It counts the number of immediate subclasses', i.e., number of children of the said class in the inheritance hierarchy.
RFC (Response For a Class)	[26]	It counts the number of local methods and the number of nonlocal methods which are called by the local methods.
DAM (Data Access Metric)	[28]	It is calculated as the ratio of private + protected attributes of the said class to the total number of attributes defined in that class.
MOA (Measure Of Aggression)	[28]	It counts the percentage of user defined data declared in the said class.
MFA (Method of Functional Abstraction)	[28]	It is counted as the ratio between the inherited methods and the total number of methods in the said class.
CAM (Cohesion Among the Methods of a Class)	[28]	Based on the signatures of the methods, this metric computes the similarity among various methods of the said class.
AMC (Average Method Complexity)	[27]	It is computed as the average of McCabe's Cyclomatic Complexity of all method.
CBO (Coupling Between Object)	[26]	It is computed by counting the number of other classes to which said the class is coupled.
LCOM (Lack of Cohesion of Methods)	[26]	It is calculated by counting the number of disjoint sets of local methods in a class.
LCOM3 (Lack of Cohesion Among Methods of a Class)	[27]	Proposed by Henderson-sellers to remove some of the disadvantages of LCOM.
NPM (Number of Public Methods)	[28]	It is computed by counting the number of public methods in a given class.
Ca (Afferent Couplings)	[27]	It is counted as the number of classes calling the said class.
Ce (Efferent Couplings)	[27]	It is counted as the number of other classes called by said class.
IC (Inheritance Coupling)	[27]	It is counted as the number of parent classes to which a class is coupled.
LOC (Lines of code)	[27]	It counts the number of lines of code. Comments lines and blank lines are simply ignored.

TABLE 3. Characteristics of open source software

Software	Version	Release Date	No. of Classes	% change
Drumkit	1-0.5.0 to 1-0.6.0	25 Apr 2014	101 classes	20.34
OpenCV	2.4.10 to 3.0	12 Sept 2014	143 Classes	16.29
Abdera	1.1.2 to 1.1.3	08 Jan 2014	686 Classes	61.88
Ivy	2.2.0 to 2.3.0	21 June 2013	614 Classes	74.76
Log4j	1.2.16 to 1.2.17	31 Mar 2010	351 classes	34.50
JEdit	5.1 to 5.2	28 July 2013	417 classes	24.96
JUnit	4.10 to 4.11	29 Sept 2011	251 classes	14.644

- JUnit is an open source framework for writing and executing unit tests and defining test suite. They provide compatibility with almost all IDEs and inbuilt test drivers; hence only tests case needs to be written. (http://sourceforge.net/projects/junit/)

4. **Research Methodology.** Basically, there are two broad approaches in which the maintainability of a software can be measured, firstly through the measurement of external quality factors such as understandability, analyzability, modifiability and secondly through the measurement of internal quality metrics and use them for making software maintainability prediction model. In the first approach, external factors can only be measured by collecting the opinion from the developers who participate in writing the source code of the open source software. Conducting such surveys is not only time-consuming and involves high cost but also produces different opinions due to the subjective nature of external quality factors. The second approach of measuring the internal quality attributes through OO metrics suite has been used in many empirical studies [3,6-16]. Almost all of the studies showed the existence of the relationship between OO metrics suite and maintainability. In the current empirical investigation, the second approach is used and the adopted research methodology is shown in Figure 1.

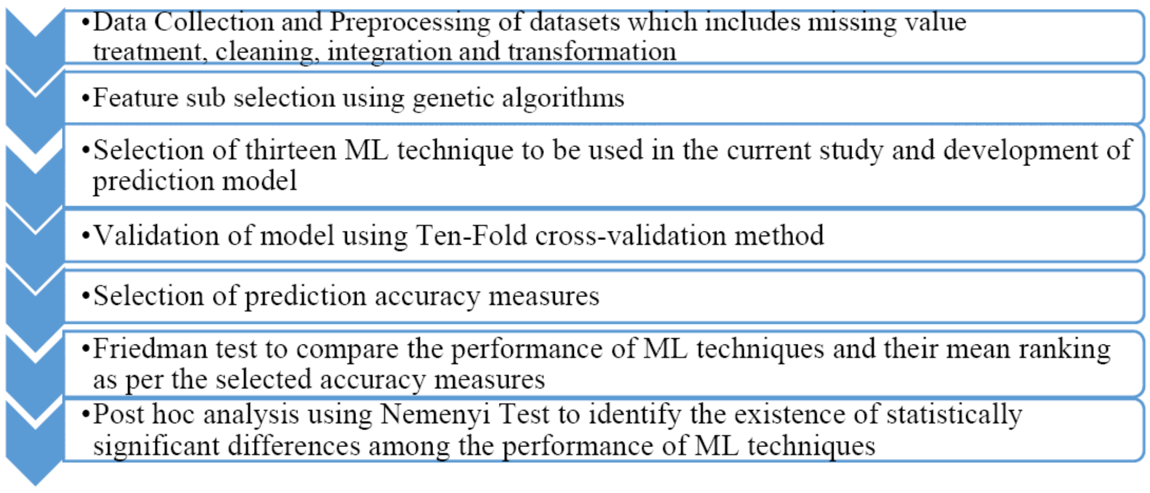

•Data Collection and Preprocessing of datasets which includes missing value treatment, cleaning, integration and transformation
•Feature sub selection using genetic algorithms
•Selection of thirteen ML technique to be used in the current study and development of prediction model
•Validation of model using Ten-Fold cross-validation method
•Selection of prediction accuracy measures
•Friedman test to compare the performance of ML techniques and their mean ranking as per the selected accuracy measures
•Post hoc analysis using Nemenyi Test to identify the existence of statistically significant differences among the performance of ML techniques

FIGURE 1. Proposed research methodology

4.1. **Data pre-processing.** Git repository mining tool 'Defect Collection and Reporting System' (DCRS) developed in the Java language at Delhi Technological University by Malhotra et al. [29] has been used for the purpose of empirical data collection from two repositories http://www.github.com and http://www.sourceforge.net. It processes the repositories and reads the change descriptions such as timestamp of committing the incurred change, unique change identifier, type of change (defective, perfective or corrective), change descriptions, list of changes and counting the line of code where changes took place and generates the reports containing detailed information for each class. It also calculates a total number of changes, values of the OO metrics for each class and provides insight such as cloning of Git Repositories and Self Loggings. For example, if Abdera 1.1.2 (older version) and Abdera 1.1.3 (newer version) are being analyzed in DCRS, processing the change logs generates the number of changes performed from older version to newer version per class. Similarly, processing the classes in older version generates the class wise values of each OO metric. Finally, both are combined for each class to generate the data points. Figure 2 graphically summarizes the changes made into each system selected in the current study.

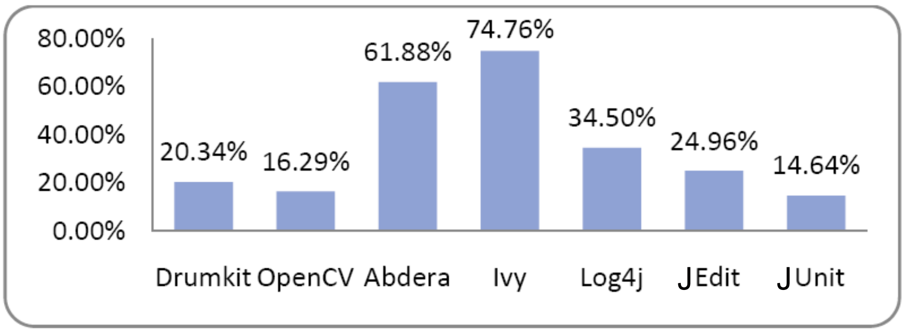


Figure 2. Summary of changes made into each dataset

During the preprocessing we extract those classes which are common in current as well as the previous version for each software system. Classes either added in latest version or deleted from the older version are simply discarded. Library classes, as well as interface classes, are also excluded from the list.

4.2. **Feature sub-selection.** The aim for carrying out the feature sub-selection process is to remove irrelevant and redundant independent variables from the dataset before it can be used further by the seventeen classifiers selected in the current study for training purpose as suggested by Donell [30]. This dimensionality reduction process not only reduces the unnecessary attributes and irrelevant noisy data, but it also enhances the execution time, improves the quality of datasets and thereof amplifies the accuracy of the prediction process. Kohavi and John [31] explained the feature selection algorithms and categorized them in two types: Wrappers and Filters. While evaluating the most valued subset from a given set of attributes, filter method does not require the classification algorithms whereas wrappers method needs them. In this empirical investigation, genetic algorithms (GA) are used for feature sub-selection process as suggested by Yang and Honavar [32]. GA is deployed using three operators: selection, crossover and mutation. In selection operation, every solution is evaluated on the basis of predefined fitness function and good solutions are picked up to breed new generations. In crossover operations, these good solutions are combined to generate better off springs. In mutation operation, solution string is mutated to maintain genetic diversity from one generation to the next. In each generation, the population is evaluated against the predefined fitness function to test if the termination criterion is met or not. This genetic cycle of applying all three operators on the population and re-evaluation continues until a specific termination criterion is achieved.

Table 4. Description of the machine learning techniques

Name of the ML Technique	Reference	Description
Linear regression (LR)	Hoffmann and Shafer [38]	LR first determines the percent of variance occurring in the dependent variable due to each independent variable separately and further used this knowledge to predict the dependent variable.
M5Rules	Kohavi and Sommerfield [39]	Accurate and compact rule sets are generated using separate-and-conquer paradigm in this technique. Initially, the rules are induced and later on revised using complex global optimization procedure to build a model tree and further used for predictions.
Decision Tree (DT)	Kohavi and Sommerfield [39]	It creates a decision tree based on the concept of entropy and information gain. During the construction as per the pre-specified splitting criterion, the most qualified independent variable is selected at the node.
Support Vector Machine (SVM)	Cortes and Vapnik [40]	Although SVM was originally developed for solving the binary classification problems, later on it was also extended to solve the regression problems. Hyperplane is created in this method to separate the data into the nonlinear region and finally with the help of kernel function data points are mapped into different dimensional space.
K-Star	Lee and Song [41]	It is an instance-based classifier that uses similarity function from the training set to classify the test set. This method uses entropy based distance function and missing values are averaged by column entropy curves.
Bagging	Brieman [42]	Developed by Leo Breiman to increase the accuracy of regression models, it reduces the variance and helps to avoid the problems associated with over-fitting. The idea is to build various similar training sets and train a new function for each of them.
Jordan Elman Recurrent Network (JERN)	Lee and Song [41]	It is special kind of ANN with the recurrent network in which hidden layers are fed directly into the input layers. Although slower but the recurrent network has the ability to learn sequences. It is a very powerful learning as the hidden layer is fed back into the input layer, so features detected in all previous patterns are fed into the network with each new pattern.
Back Propagation Network (BPN)	Specht and Shapiro [43]	It is a supervised learning process in which a new pattern is presented to the network through a forward activation flow of outputs in each cycle and the backward error propagation of weight adjustments is performed.
Kohonen Network (KN)	Kohonen [44]	It is an unsupervised learning network which neither uses any sort of activation function nor bias. It does not use any hidden layer. When a pattern is presented to a Kohonen network, only one of the output neurons is selected as the winner.
Probabilistic Network (PNN)	Specht [45]	PNN consists of several sub-networks in which the input nodes are the set of measurements, second layer consists of the Gaussian functions formed using the given set of data points as centers, the third layer performs an average operation of the outputs from the second layer for each class and the fourth layer performs a vote, selecting the largest value to determine the associated class label.
Group Method of Data Handling (GMDH)	Ivakhnenko and Koppa [46]	It is ideal for complex, unstructured system where the investigator is only interested in obtaining a high order input-output relationship. It can build a multinomial of a degree in hundreds whereas standard multiple regression exhausted in computation.
General Regression Neural Networks (GRNN)	Specht [47]	It is a memory-based network with one-pass learning technique with a highly parallel structure which provides smooth transitions from one observed value to another even with sparse data in a multidimensional measurement space.
GRNN Genetic Adaptive Learning (GGAL)	Specht [47]	By simulating the biological evolution, this genetic inspired neural network method has the ability to search large and complex spaces to determine near optimal solutions in time and space efficient manner.

4.3. **Machine learning techniques.** This section presents the definition of all thirteen ML techniques used in the current study as compiled in Table 4 for making prediction models. All the ML techniques are divided into two sets, and first set contains all those ML techniques which are basic in nature and non-bioinspired such as LR, M5Rules, DT, SVM, KStar, and Bagging. Second set consists of classifiers based on the bio-inspired nonlinear statistical data modeling technique such as ANN. In this set all ML techniques explored the complex relationships between inputs and outputs to identify the patterns during the training process and further used the same knowledge for predictions process such as JERN, BPN, KN, PNN, GMDH, GRNN, and GGAL.

4.4. **Ten-fold cross-validation.** Ten-fold cross-validation scheme is used to train and validate the software maintainability prediction model. In this technique, the data is divided into ten random partitions. Each classifier trains itself from 9 partitions of the datasets and validates itself on the remaining 10th partition. The process is repeated 10 times and the results of all folds are combined to produce the final result. It ensures that the training of the model from the dataset remains unbiased and accurate.

4.5. **Prediction accuracy measures.** Estimating the accuracy of the predicted model is a very important step in any empirical study. Each time the predicted value of the dependent variable is generated from the model, which is compared with the respective actual value to find the errors. Although many types of residual-based measures have been recommended in literature to evaluate the prediction accuracy of any given model, however, following three prevalent measures suggested by Kitchenham et al. [33] and Conte et al. [34] are used to compare the performance of various ML techniques used in the current study.

4.5.1. *Mean absolute error.* Mean absolute error (MAE) is a normalized measure to detect the discrepancy between actual and predicted value of dependent variable (maintenance effort in this case). In MAE first the difference between the actual and predicted value is calculated and the result is divided by the actual value. Then, the absolute value for each data point is summed and is divided by the total number of data points. MAE is defined as follows:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\frac{|\text{Actual Value} - \text{Predicted Value}|}{(\text{Actual Value})} \tag{1}$$

where N is the number of classes.

4.5.2. *Root mean squared error.* Another measure used to compare the ML techniques is the root mean squared error (RMSE) defined as the square root of the variance of the residual value. In RMSE, the difference between the predicted values with actual values for each class is squared then averaged and finally the square root of this average value is taken. The RMSE measure is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\text{Actual Value} - \text{Predicted Value})^2} \tag{2}$$

This method gives comparatively high weightage to large errors as the

differences are squared before they are averaged. It is chosen when large errors are most undesirable. Lower values of RMSE indicate better fitness of the model used for predictions.

4.5.3. Pred(q). Another prediction accuracy measure, selected to compare the performance of ML technique is Pred(0.25), i.e., at 25%. It calculates the proportions of the result that has MAE less than 0.25 to the total number of observations made. It is given as:

$$\text{Pred}(q) = \frac{K}{N} \tag{3}$$

where q is the specified value, K is a number of observations whose MAE is less than or equal to q and N is the total number of classes.

4.6. **Friedman test to rank the performance.** The Friedman's test [35] is a kind of non-parametric statistical test, which is used to find if there exists significant difference among the performance of ML techniques and rank them accordingly. The following hypothesis is formed before conducting the Friedman test on results.

Null Hypothesis (H0): There is no significant difference among the performance of participant ML techniques.

Alternate Hypothesis (H1): There exists significant difference among the performance of participant ML Techniques.

As suggested by Demsar [36], when performances of multiple classifiers needed to be compared on multiple datasets, it is one of the best techniques. The Friedman measure is defined as follows:

$$\chi_r^2 = \left(\frac{12}{Nk(k+1)}\sum_{i=1}^{k} R^2\right) - 3N(k+1) \tag{4}$$

where R is the average rank of the individual method, N is the number of datasets; k is number of ML techniques considered for ranking. The value of $\chi_{\text{calculated}}$ is calculated from the given Equation (4) and compared with $\chi_{\text{tabulated}}$ using chi-square distribution table. If the value of Friedman Measure, i.e., $\chi_{\text{calculated}}$ lies in the critical region, Null hypothesis is rejected, alternate hypothesis is accepted and it is concluded that there exists significant difference among the performance of participant ML techniques. Otherwise Null hypothesis is accepted, alternate hypothesis is rejected and concluded that there does not exist significant difference between the performance of participant ML techniques.

Further, individually every technique is also ranked using Friedman's Individual Rank (FIR) based on undermentioned Equation (5).

$$\text{Friedman's Individual Rank (FIR)} = \frac{C}{N} \tag{5}$$

where C is the cumulative rank and N is number of datasets. FIR for each technique is calculated and the technique which scores lowest value of FIR is considered as the best performer and the technique achieving the highest rank is termed as worst performer. If the results based on the mean rank achieved using FIR for both the performance measure MAE or RMSE is found to be significant, it is advisable to check whether the difference in mean rank is statistically significant or not by means of post hoc analysis using Nemenyi test.

4.7. **Post hoc analysis using Nemenyi test.** When the sample size is equal and the data is not normalized, Nemenyi test is a very powerful test for post hoc analysis [37]. It is used to compare the performance of ML models for finding the existence of statistically significant difference among themselves. First of all critical difference (CD) is calculated using Equation (6) which depends upon the number of techniques, number of datasets and the level of significance.

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

where k means the number of techniques, N is the number of the data sample. The value of q_α is based on Studentized range statistics for a given level of significance as defined by Demsar [36]. While comparing the performance of two ML techniques during post hoc analysis, the difference between their respective FIR values is calculated. If this difference is greater than or equal to the value of CD, it is concluded that the performance of two ML techniques is statistically significant at the selected significance level α. If the difference is less than the value of CD, it is concluded that the difference between their performances is not statistically significant.

5. Result Analysis.

This section presents the prediction results of various classifiers based on ML techniques for maintainability prediction using OO metrics.

5.1. Feature sub selection (FSS).

The first step in the empirical study was FSS; in which irrelevant and unimportant features were removed using GA. Table 5 summarizes the relevant metrics found after applying FSS using GA over all the releases of seven datasets selected in the current study. In total 17%, 17%, 41%, 35%, 23%, 41% and 35% reductions were observed for Drumkit, OpenCV, Abdera, Ivy, Log4j, JEdit and JUnit datasets respectively. On an average 24% saving is observed for all datasets.

Out of the seventeen independent variables, we found that LCOM3, LOC and DIT are the most commonly selected OO metrics in the current study. Efferent coupling Ce is also found to be significant in Abdera and Log4j Systems. We also found that the results obtained using reduced set of independent variables after applying FSS were slightly better as compared to the results obtained using all independent variables in prediction models. Similar observations were made by Kohavi and John [31] as well as Yang and Honavar [32] that not only the impact of FSS on the accuracy is minimal but they are also capable of capturing all the characteristics irrespective of the size of the extracted subset. Moreover, the time consumed by prediction model on newly reduced dataset using FSS is comparatively lesser than the time consumed on actual dataset.

Table 5. Metrics obtained using feature sub-selection using GA algorithm

Software Name	Selected Relevant OO Attributes
Drumkit	WMC, RFC, DIT, LCOM3
OpenCV	CBO, DIT, LCOM3, LOC
Abdera	Ce, NPM, LOC, LCOM3, DAM, CAM
Ivy	LCOM3, LOC, DAM, MOA, CAM, AMC
Log4j	NPM, Ce, LOC, LCOM3, DIT, MOA, CAM
JEdit	WMC, LOC, DIT, DAM, CAM, AMC
JUnit	RFC, CBO, LCOM, LCOM3, NPM, IC

RQ1: Does the impact of OO metrics on maintainability exist in the context of OSS?

To assess the outcome of ML techniques based prediction models, their prediction accuracy was measured through MAE, RMSE, Pred(0.25) and Pred(0.75). Their values were evaluated as per criterion set by previous researchers [33,34] that any prediction model is considered accurate if its MAE values are less than 0.40 also the value of Pred(0.25) should always be greater than Pred(0.75).

Table 6. MAE values over seven datasets

Dataset	Drumkit	OpenCV	Abdera	Ivy	Log4j	JEdit	JUnit
LR	0.57	0.48	0.41	0.29	0.36	0.38	0.54
M5Rule	0.49	0.52	0.43	0.37	0.38	0.41	0.52
DT	0.58	0.57	0.44	0.39	0.42	0.45	0.49
SVM	0.43	0.64	0.46	0.49	0.46	0.61	0.52
KStar	0.45	0.62	0.51	0.33	0.48	0.42	0.43
Bagging	0.51	0.45	0.43	0.41	0.47	0.52	0.44
JERN	0.53	0.54	0.41	0.59	0.51	0.52	0.44
BPN	0.42	0.58	0.61	0.37	0.35	0.42	0.47
KN	0.56	0.49	0.37	0.41	0.43	0.52	0.50
PNN	0.38	0.42	0.38	0.53	0.37	0.41	0.47
GMDH	0.49	0.37	0.32	0.34	0.32	0.30	0.35
GRNN	0.38	0.42	0.45	0.41	0.32	0.49	0.48
GGAL	0.33	0.31	0.29	0.36	0.28	0.34	0.36

Firstly, we present the results of all 13 ML techniques for maintainability prediction models validated using 10-fold cross validation on seven OSS. Difference between the predicted value and actual value is compared and analyzed using various accuracy measures such as MAE, RMSE, Pred(25%) and Pred(75%) using Equations (1), (2) and (3) respectively.

The MAE values of each ML technique for all seven datasets used in the study are summarized in Table 6. For example, it is observed that when LR technique was applied on JEdit datasets, it gave an accuracy of 62% (since the error is 0.38, accuracy is 100-38). Similarly, we found 67% accuracy when GGAL technique applied on Drumkit datasets respectively. The accuracy of all the ML techniques w.r.t. MAE on all seven selected datasets lies between the ranges of 39-77% which is quite encouraging. Thus, it highlights the capability of ML technique for effective maintainability predictions of OSS.

The value of RMSE is obtained using Equation (2) after performing ten runs of ten-fold cross-validation for each ML technique on each dataset tabulated in Table 7. Each row represents the RMSE value of a particular technique on specific datasets. For example, first row compiles the value of RMSE when LR technique is used with all seven datasets and generates values as 0.66, 0.56, 0.61, 0.58, 0.49, 0.42 and 0.47 respectively.

Prediction accuracy of each classifier on each dataset is calculated at 25% as well as at 75% and the results are compiled in Table 8. While compiling the results for each ML technique, each dataset is divided in two sub-columns; first column contains the values of Pred(0.25), i.e., 25% and second column contains the values of Pred(0.75), i.e., 75%. It also helps in determining whether the results are as per the criterion set by [33,34] that any prediction model is considered accurate if the value of (0.25) is less than Pred(0.75). As per the results presented in Tables 6, 7, and 8 for the respective values of MAE, RMSE, Pred(25%) and Pred(75%), we found that even though the prediction models for software maintainability usually attain less accuracy [10-13,19,21], in the current study we found their reasonable values.

When we analyzed the values of pred(0.25) and pred(0.75) from Table 8, it is recorded in the range of 72-78% for Pred(25%) and 66-89% for Pred(75%) which is quite reassuring that ML techniques are very effective. With respect to Pred(0.25), GMDH is found to be most accurate with Log4j dataset and JUnit dataset, i.e., 79% accuracy (entries are highlighted in Table 8). We also observe that GMDH method achieved more than 70% accuracy with four out of seven datasets. Similarly, if Pred(0.75) is taken as accuracy

Table 7. RMSE values over seven datasets

Dataset	Drumkit	OpenCV	Abdera	Ivy	Log4j	JEdit	JUnit
LR	0.66	0.56	0.61	0.58	0.49	0.42	0.47
M5Rule	0.72	0.38	0.30	0.37	0.41	0.56	0.63
DT	0.86	0.74	0.41	0.34	0.43	0.47	0.58
SVM	0.64	0.67	0.47	0.42	0.39	0.48	0.54
KStar	0.67	0.77	0.55	0.38	0.39	0.56	0.61
Bagging	1.23	1.52	0.59	0.39	0.47	0.59	0.43
JERN	0.53	0.54	0.41	0.59	0.47	0.39	0.48
BPN	0.42	0.58	0.61	0.33	0.41	0.42	0.51
KN	0.56	0.48	0.37	0.40	0.35	0.43	0.52
PNN	0.38	0.42	0.33	0.53	0.43	0.53	0.41
GMDH	0.23	0.37	0.32	0.34	0.37	0.38	0.42
GRNN	0.38	0.42	0.45	0.41	0.32	0.49	0.47
GGAL	0.33	0.41	0.49	0.36	0.28	0.37	0.32

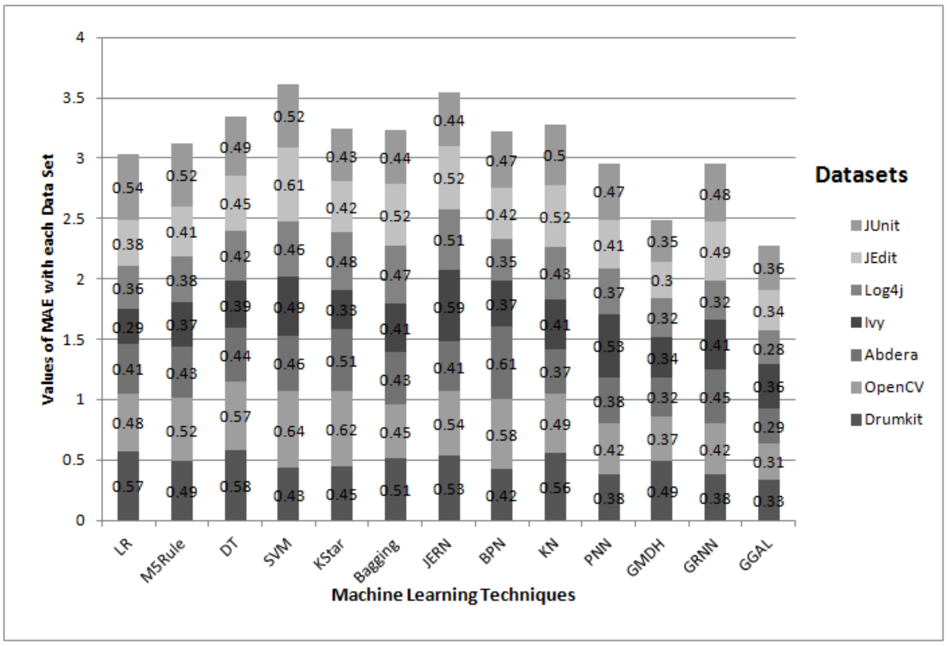
Table 8. Results of Pred(25%) and Pred(75%) over seven datasets

Dataset	Drumkit		OpenCV		Abdera		Ivy		Log4j		JEdit		JUnit	
	25%	75%	25%	75%	25%	75%	25%	75%	25%	75%	25%	75%	25%	75%
LR	62	67	58	71	56	66	61	71	63	73	59	70	54	64
M5Rule	63	69	49	69	57	76	59	74	65	75	51	73	57	55
DT	70	78	53	72	49	63	57	87	68	78	61	71	63	69
SVM	65	76	47	67	51	72	52	82	73	81	42	64	49	66
KStar	68	88	55	75	52	69	58	78	72	88	57	72	61	69
Bagging	75	79	51	69	59	72	65	77	69	**89**	43	61	38	52
JERN	68	88	69	73	72	74	62	71	74	81	61	69	52	64
BPN	62	86	57	77	62	81	73	79	77	87	49	57	60	77
KN	67	87	63	79	53	83	48	69	73	81	58	65	52	72
PNN	51	83	61	73	59	79	64	71	73	85	59	66	61	75
GMDH	69	77	73	78	65	74	78	88	**79**	86	73	84	68	**79**
GRNN	65	81	59	83	65	75	68	73	71	84	75	80	62	73
GGAL	72	85	67	82	73	72	75	69	69	88	70	83	74	81

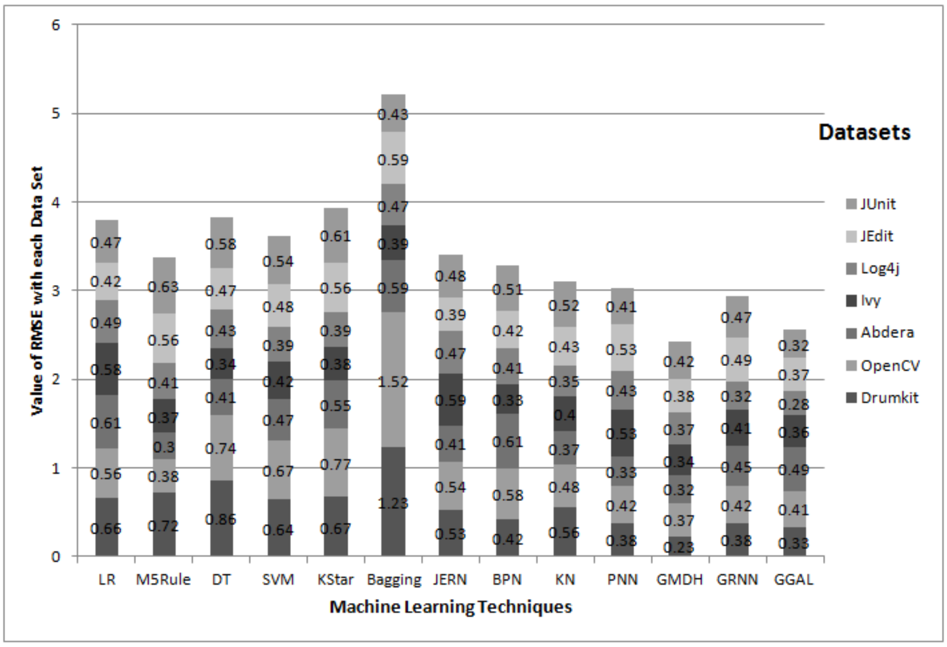
measure, Bagging is found to be most accurate with Log4j dataset, i.e., 89% accuracy. GGAL is also found to be the best ML technique because more than 80% accuracy is achieved with five out of seven datasets at Pred(0.75). When we closely observe the range of accuracies, GGAL has performed outstandingly. Results are in the range of 67-75% across all datasets which are quite close to the criterion set by Kitchenham et al. [33] and Conte et al. [34].

Further, Figures 3(a) and 3(b) depict the MAE and RMSE values obtained by each ML technique on all the seven datasets correspondingly. Figure 3(a) clearly shows that GMDH and GGAL ML techniques have performed highest over all seven datasets. It is also evident from Figure 3(a) that minimum values recorded for MAE on all seven datasets were within the range of 0.28-0.36.

Additionally, with respect to RMSE as depicted in Figure 3(b), we found that mean values of RMSE range from 0.23-0.63. It is also observed that four ML techniques PNN, GMDH, GRNN and GGAL have achieved less than 30% error which is considered to be excellent.



(a)



(b)

FIGURE 3. (a) MAE values of each ML technique on corresponding datasets using 10-fold cross validation, (b) RMSE values of each ML technique on corresponding datasets using 10-fold cross validation

On judging the overall performance of all ML techniques using four measures in the current study, it clearly satisfies the criteria laid down by [33,34], and hence we conclude that the impact of OO metrics on maintainability indeed exists in the perspective of OSS; *thus ML techniques can be successfully applied for their maintainability prediction using OO metrics suite.*

RQ2: What is the comparative performance of ML techniques for maintainability prediction of OSS?

We applied extensive statistical tests in order to check whether the performances of proposed ML techniques are significantly different or not. As per Demsar [36], non-parametric tests are safer as they do not assume normal distribution or homogeneity of variance in the data. In the current investigation, Friedman test was used to compare the performance of thirteen ML techniques on seven datasets. We calculate the value of critical region at 5% significance level and degree of freedom 12 (i.e., for 13 ML techniques-1). The value of $\chi_{\text{tabulated}}$ is obtained from Chi-square table where the degree of freedom is twelve (for thirteen ML techniques) and level of significance was 95%.

The null hypothesis of the Friedman test states that there is no significant difference between the performance of ML techniques. We found that at significant level 0.05 calculated value of Friedman measure, i.e., $\chi_{\text{calculated}}$ lies in the critical range for MAE as well as RMSE; hence, the *Null hypothesis is rejected and alternative hypothesis is accepted and it is concluded that significant difference exists among the performance of participant ML techniques.*

Further, in order to rank the performance of each of the ML technique, their FIR is calculated using Equation (5) and compiled in Table 9 and Table 10 for MAE and RMSE respectively. As discussed earlier, lower the mean rank means better the performance. The outcome of the Friedman test using FIR for ranking as compiled in Table 9 with respect to MAE measure indicates that the performance of *GGAL technique is the best and GMDH is the second best technique.*

Table 9. Mean ranking of ML techniques by applying Friedman test on MAE

ML Tech	GGAL	GMDH	PNN	GRNN	LR	BPN	M5Rule	KStar	Bagging	KN	DT	JERN	SVM
Mean Rank	1.79	2.71	3.57	4.36	5.36	6.64	7.14	8.57	9.1	9.43	10.6	11.28	11.57

Table 10. Mean ranking of ML techniques by applying Friedman test on RMSE

ML Tech	GMDH	GGAL	GRNN	KN	PNN	BPN	M5Rule	JERN	DT	SVM	LR	KStar	Bagging
Mean Rank	1.21	2.08	3.79	4.14	5.79	7.28	7.87	8.32	8.5	9.21	9.57	10.36	11.58

With respect to RMSE from Table 10, we observe that GMDH technique is the second best and GGAL as the best technique for the maintainability prediction of OSS on the basis of their mean rank.

In order to ascertain whether the performance differences which exist between FIR values of various ML techniques are statistically significant or not, we proceed towards post hoc analysis in RQ3.

RQ3: In terms of performance, which ML technique is significantly better than the other techniques?

In RQ2, with the help of Friedman test, we concluded that there exists a significant difference among the performance of ML technique; hence we proceed towards post hoc analysis using Nemenyi test to determine whether the difference is actually statistically significant between the performances of ML techniques or not.

The value of CD is calculated as 6.8 after putting the values of n as 13 (number of ML techniques) and value of k as 7 (number of datasets) into Equation (6). Next, we make a pair for each ML technique with every other to calculate their rank differences (FIR) and compiled in Tables 11 and 12 for MAE and RMSE, respectively. In total, 78 such pairs were formed as we have used 13 ML techniques in our study.

In Table 11, we have highlighted those entries which have values greater than CD. It is quite evident that out of 78 pairs of ML techniques, 13 pairs (bold entries) were found to have significant differences among their performances. One pair between GGAL and KStar has attained the difference (6.78) almost touching the critical difference (6.8). Hence, 14 pairs out of 78 means, *the performance of 17.9% of pairs was found to be significantly different using statistical test and not coincidental.*

Results shown in Table 11 also depict that GGAL performed better than KStar, Bagging, KN, DT, JERN and SVM whereas GMDH performed better than DT, JERN and SVM. *Hence, on the basis of post hoc analysis of MAE, we conclude that GMDH and GGAL outperformed other ML techniques.* The difference between the performance of all other ML techniques was not found to be significant.

We performed the same procedure for RMSE and the rank difference of each pair was calculated and compiled in Table 12. Highlighted entries in Table 12 indicate that the difference of FIR between that pair of ML technique is greater than CD.

It is observed that out of 78 pairs of ML techniques, 12 pairs (bold entries) were found to have significant differences among their performances. *So, with the help of Nemenyi*

Table 11. Computation of pairwise rank difference amongst all ML techniques in terms of MAE

ML Tech	GGAL	GMDH	PNN	GRNN	LR	BPN	M5Rule	KStar	Bagging	KN	DT	JERN	SVM
GGAL	–	0.92	1.78	2.57	3.57	4.85	5.35	**6.78**	**7.31**	**7.64**	**8.81**	**9.49**	**9.78**
GMDH		–	0.86	1.65	2.65	3.93	4.43	5.86	6.39	6.72	**7.89**	**8.57**	**8.86**
PNN			–	0.79	1.79	3.07	3.57	5.00	5.53	5.86	**7.03**	**7.71**	**8.00**
GRNN				–	1.00	2.28	2.78	4.21	4.74	5.07	6.24	**6.92**	**7.21**
LR					–	1.28	1.78	3.21	3.74	4.07	5.24	5.92	6.21
BPN						–	0.5	1.93	2.46	2.79	3.96	4.64	4.93
M5Rule							–	1.43	1.96	2.29	3.46	4.14	4.43
KStar								–	0.53	0.86	2.03	2.71	3.00
Bagging									–	0.33	1.5	2.18	2.47
KN										–	1.17	1.85	2.14
DT											–	0.68	0.97
JERN												–	0.29
SVM													–

Table 12. Computation of pairwise rank difference amongst all ML techniques in terms of RMSE

ML Tech	GMDH	GGAL	GRNN	KN	PNN	BPN	M5Rule	JERN	DT	SVM	LR	KStar	Bagging
GMDH	–	0.87	2.58	2.93	4.58	6.07	6.66	**7.11**	**7.29**	**8.00**	**8.36**	**9.15**	**10.37**
GGAL		–	1.71	2.06	3.71	5.2	5.79	6.24	6.42	**7.13**	**7.49**	**8.28**	**9.5**
GRNN			–	0.35	2	3.49	4.08	4.53	4.71	5.42	5.78	6.57	**7.79**
KN				–	1.65	3.14	3.73	4.18	4.36	5.07	5.43	6.22	**7.44**
PNN					–	1.49	2.08	2.53	2.71	3.42	3.78	4.57	5.79
BPN						–	0.59	1.04	1.22	1.93	2.29	3.08	4.3
M5Rule							–	0.45	0.63	1.34	1.7	2.49	3.71
JERN								–	0.18	0.89	1.25	2.04	3.26
DT									–	0.71	1.07	1.86	3.08
SVM										–	0.36	1.15	2.37
LR											–	0.79	2.01
KStar												–	1.22
Bagging													–

test conducted on RMSE measure, 12 pairs were found to be significantly different out of 78 pairs which is almost 15.3% of the total pairs.

It is also quite apparent that GMDH-JERN, GMDH-DT, GMDH-SVM, GMDH-LR, GMDH-KStar and the GMDH-Bagging pair were found to be significant as they have a value greater than CD. GGAL was found to be performing significantly superior to SVM, LR, KStar and Bagging. GRNN and KNN also performed better than Bagging technique.

Hence, we conclude that the difference in the performances of GGAL and GMDH were statistically different significantly as well as better than other ML techniques and the difference among the performance of all other ML techniques is not found to be significant.

6. Threats to Validity.

This empirical investigation is carried out on OSS which may not be the true representative of all those softwares used in industry but with the help of ten-fold cross-validation and repetition of same ML technique for training over seven datasets with different individuality, we have tried our best to obtain as much unbiased and generalized results as possible. The results of this study would be incomplete without the discussion of all three types of threats to validity present in every empirical investigation: External, Internal and Construct Validity.

External validity means the degree with which the results of this empirical study can be generalized. By taking seven OSS with different sizes, different characteristics, different maintenance requirements, we have reduced this threat to its minimum.

Internal validity is defined as the degree with which conclusions can be drawn about the consequence of independent variables on dependent variables. We have minimized this threat by successfully applying FSS using GA and further exploring the cumulative effects of all the selected independent variables on maintainability.

Construct validity represents the extent to which the OO characteristics are accurately captured through the independent and dependent variables used in any study. We have not stuck to only one metric suite in our study, instead seventeen different metrics were selected to capture the OO characteristics proposed by well-tested metrics suite from Chidamber and Kemerer [26], Bansiya and Davis [27], and Henderson-Sellers [28]; hence, it is reasonable to claim that this threat is also minimized.

7. Conclusion and Future Direction.

The objective of the study was to analyze the effectiveness of ML techniques for predicting software maintainability and the results are validated using dataset collected from open source software. An extensive empirical comparison of thirteen ML techniques on seven datasets obtained from open source code repositories is conducted. Prediction models were developed using seventeen most commonly used OO metrics. We further compared the performance of ML techniques using four prediction accuracy measures MAE, RMSE, Pred(0.25) and Pred(0.75). The variations amongst the performance of various ML techniques were further evaluated for significance using Friedman test. Post hoc analysis using Nemenyi test was also conducted to identify whether there exists the statistical difference of performance between the pair of different ML techniques. The main findings of the work are summarized below.

- Feature subselection using GA was used and it could successfully reduce the dimensions by almost 26.6%.
- To measure the residual error, MAE and RMSE prediction accuracy was used and we found that GGAL and GMDH techniques perform better than other ML techniques.
- The work also confirms that ML techniques have overall fare predictive ability as Pred(0.25) values are more than 60% in all cases; hence, they can be used for making the prediction model for software maintainability of OSS.

- The superiority of GGAL and GMDH techniques over other ML techniques in the context of maintainability prediction of OSS was further confirmed by the results of Friedman test and post hoc analysis.

Hence, ML techniques based prediction models can be used to decide the ideal time for throwing away the old system and replace it by the whole new system. The results of this study are exploratory and indicative. Moreover, there might be a number of other factors that may possibly affect the results and limit their generalization. We plan more studies in future work validated on industrial software systems.

REFERENCES

[1] IEEE Std 828-1998, *IEEE Standard for Software Configuration Management Plans*, Standard, 1998.

[2] M. Jorgensen, Experience with the accuracy of software maintenance task effort prediction models, *IEEE Trans. Software Engineering*, vol.21, no.8, pp.674-681, 1995.

[3] A. D. Lucia, E. Pompella and S. Stefanucci, Assessing effort estimation models for corrective maintenance through empirical studies, *Journal of Information and Software Technology*, vol.47, no.1, pp.3-15, 2005.

[4] W. Scacchi, Understanding the requirements for developing open source software systems, *IEE Proceedings Software*, vol.149, no.1, pp.24-39, 2002.

[5] I. Samoladas, I. Stamelos, L. Angelis and A. Oikonomou, Open source software development should strive for even greater code maintainability, *Communications of the ACM*, vol.47, no.10, pp.83-87, 2004.

[6] R. K. Bandi and V. K. Vaishnavi, Turk DE: Predicting maintenance performance using object-oriented design complexity metrics, *IEEE Trans. Software Engineering*, vol.29, no.1, pp.77-87, 2003.

[7] V. R. Basili, L. C. Briand and W. L. Melo, A validation of object-oriented design metrics as quality indicators, *IEEE Trans. Software Engineering*, vol.22, no.10, pp.751-761, 1996.

[8] W. Li and S. Henry, Object-oriented metrics which predict maintainability, *Journal of Systems and Software*, vol.23, no.2, pp.111-122, 1993.

[9] F. Fioravanti and P. Nesi, Estimation and prediction metrics for adaptive maintenance effort of an object-oriented system, *IEEE Trans. Software Engineering*, vol.27, no.12, pp.1062-1084, 2001.

[10] M. Dagpinar and J. H. Jhanke, Predicting maintainability with object-oriented metric – An empirical comparison, *Proc. of the 10th Working Conference on Reverse Engineering*, pp.155-164, 2003.

[11] 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.

[12] C. V. 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.

[13] M. O. Elish and K. O. Elish, Application of tree net in predicting object-oriented software maintainability: A comparative study, *Proc. of European Conference on Software Maintenance and Reengineering*, Kaiserslautern, Germany, pp.69-78, 2009.

[14] C. Jin and J. A. Liu, Applications of support vector machine and unsupervised learning for predicting maintainability using object-oriented metrics, *Proc. of the 2nd International Conference on Multi-Media and Information Technology*, pp.24-27, 2010.

[15] 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.339-515, 2010.

[16] Y. Zhou and B. Xu, Predicting the maintainability of open source using design metrics, *Wuhan University Journal of Natural Sciences*, vol.13, no.1, pp.14-20, 2008.

[17] F. J. Ramil, A. Lozano, M. Wermelinger and A. Capiluppi, Empirical studies of open source evolution, *Software Evolution*, pp.263-288, 2008.

[18] I. Myrtveit, E. Stensrud and M. Shepperd, Reliability and validity in comparative studies of software prediction models, *IEEE Trans. Software Engineering*, vol.31, no.5, pp.380-391, 2005.

[19] 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.

[20] S. C. Misra, Modeling design/coding factors that drive maintainability of software systems, *Software Quality Journal*, vol.13, pp.297-320, 2005.

[21] 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, no.2, pp.165-173, 2014.

[22] K. K. Aggarwal, Y. Singh, A. Kaur and R. Malhotra, Application of artificial neural network for predicting maintainability using an object-oriented metric, *Trans. Engineering, Computing and Technology*, vol.15, pp.285-289, 2008.

[23] R. Malhotra and A. Chug, An empirical study to redefine the relationship between software design metrics and maintainability in high data intensive applications, *Lecture Notes in Engineering and Computer Science: Proc. of the World Congress on Engineering and Computer Science (WCECS)*, San Francisco, USA, pp.61-66, 2013.

[24] R. Malhotra and A. Chug, Software maintainability prediction using machine learning algorithm, *Software Engineering: An International Journal*, vol.2, no.2, pp.9-36, 2012.

[25] K. K. Aggarwal, Y. Singh, P. Chandra and M. Puri, Measurement of software maintainability using a fuzzy model, *Journal of Computer Sciences*, vol.1, no.4, pp.538-542, 2005.

[26] S. R. Chidamber and C. F. Kemerer, Towards a metrics suite for object oriented design, *Proc. of the 6th ACM Conference on Object-Oriented Programming, Systems Languages, and Applications*, Phoenix, AZ, 1991.

[27] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*, Englewood Cliff, Upper Saddle River, NJ, USA, Prentice-Hall Inc., 1996.

[28] J. Bansiya and C. G. Davis, A hierarchical model for object-oriented design quality assessment, *IEEE Trans. Software Engineering*, vol.28, no.1, pp.4-17, 2002.

[29] R. Malhotra, N. Pritam, K. Nagpal and P. Upmanyu, Defect collection and reporting system for GIT based open source software, *Proc. of International Conference on Data Mining and Intelligence Computing*, New Delhi, India, pp.1-7, 2014.

[30] G. M. Donell, Establishing relationships between specification size and software process effort in case environment, *Information and Software Technology*, vol.39, no.1, pp.35-45, 1997.

[31] R. Kohavi and G. H. John, Wrappers for feature subset selection, *Journal of Artificial Intelligence*, vol.97, no.2, pp.273-324, 1997.

[32] J. Yang and V. Honavar, Feature subset selection using genetic algorithm, *IEEE Intelligent System and Their Applications*, vol.13, no.2, pp.44-49, 1998.

[33] B. Kitchenham, L. M. Pickard, S. G. MacDonell and M. L. Shepperd, What accuracy statistics really measure, *IEE Proceedings Software*, vol.148, no.3, pp.81-85, 2001.

[34] S. Conte, H. Dunsmore and V. Shen, *Software Engineering Metrics and Models Book*, Benjamin-Cummings Publishing Co., Menlo Park, CA, 1986.

[35] 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.

[36] J. Demsar, Statistical comparison on classifier over multiple datasets, *Journal Machine of Learning Research*, vol.7, pp.1-30, 2006.

[37] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, *IEEE Trans. Software Engineering*, vol.34, no.4, pp.485-496, 2008.

[38] J. P. Hoffmann and K. Shafer, *Linear Regression Analysis, Assumptions, and Applications*, NASW Press, 2015.

[39] R. Kohavi and D. Sommerfield, Targeting business users with decision table classifiers, *Proc. of IEEE Symposium on Information Visualization*, pp.102-105, 1998.

[40] C. Cortes and V. Vapnik, Support-vector networks, *Machine Learning*, vol.20, no.3, pp.273-297, 1995.

[41] S. W. Lee and H. H. Song, A new recurrent neural network architecture for visual pattern recognition, *IEEE Trans. Neural Networks*, vol.8, no.2, pp.331-340, 1997.

[42] L. Brieman, Bagging predictors, *Machine Learning*, vol.24, no.2, pp.123-140, 1996.

[43] D. Specht and P. Shapiro, Generalization accuracy of probabilistic neural networks compared with back-propagation networks, *Proc. of the International Joint Conference on Neural Networks*, pp.887-892, 1991.

[44] T. Kohonen, The self-organizing map, *Proc. of the IEEE*, vol.78, no.9, pp.1464-1480, 1990.

[45] D. F. Specht, Probabilistic neural networks, *Journal of Neural Networks*, vol.3, no.1, pp.109-118, 1990.

[46] A. G. Ivakhnenko and Y. U. Koppa, Regularization of decision functions in the group method of data handling, *Soviet Automatic Control*, vol.15, no.2, pp.28-37, 1970.

[47] D. F. Specht, General regression neural networks, *IEEE Trans. Neural Networks*, vol.2, no.6, pp.568-576, 1991.