

Software reuse analytics using integrated random forest and gradient boosting machine learning algorithm

Amandeep Kaur Sandhu | Ranbir Singh Batth 

School of computer Science and Engineering, Lovely Professional University, Phagwara, India

Correspondence

Ranbir Singh Batth, School of computer Science and Engineering, Lovely Professional University, Phagwara, Punjab, India.
Email: ranbir.21123@lpu.co.in

Abstract

The term Cleaner Production (CP) for Production Companies is contemplated as influential to get sustainable production. CP mainly deals with three R's that is, reuse, reduce, and recycle. For software enterprise, the software reuse plays a pivotal role. Software reuse is a process of producing new products or software from the existing software by updating it. To extract useful information from the existing software data mining comes into light. The algorithms used for software reuse face issues related to maintenance cost, accuracy, and performance. Also, the currently used algorithm does not give accurate results on whether the component of software can be reused. Machine Learning gives the best results to predicate if the given software component is reusable or not. This paper introduces an integrated Random Forest and Gradient Boosting Machine Learning Algorithm (RFGBM) which test the reusability of the given software code considering the object-oriented parameters such as cohesion, coupling, cyclomatic complexity, bugs, number of children, and depth inheritance tree. Further, the proposed algorithm is compared with J48, AdaBoostM1, LogitBoost, Part, One R, LMT, JRip, DecisionStump algorithms. Performance metrics like accuracy, error rate, Relative Absolute Error, and Mean Absolute Error are improved using RFGBM. This algorithm also utilizes data preprocessing with the help of an unsupervised filter to remove the missing value for efficiency improvement. Proposed algorithm outperforms existing in term of performance parameters.

KEYWORDS

AdaBoostM1, confusion matrix, DecisionStump, gradient boosting machine, J48, JRip, LMT, LogitBoost, one R, part, random forest, software metrics, software reuse

1 | INTRODUCTION

In IT Domain Cleaner Production of Software reuse is considered as a classification problem. Software plays a significant role in our life. Either it is science, technology, business or defense all of the communication and working rely on the specific software. With change in the protocols, design and features, software is needed to be updated. Instead of creating a new software from scratch the components of older software can be used.¹ Because of the fact that “if the product already worked before it will work fine if reused”, this concept of Reuse is universally accepted.¹ Software Development Life Cycle (SDLC) is followed to create a new software. There are different types of reuse ideas that can be used during the software life cycle development. For software reuse, the primary need is the Source code of existing software. Reuse

of software benefits by increasing productivity, maintainability, quality, reliability, and reduction of costs and implementation time. The research on software reuse Analytic focuses on testing whether software components can be reused or not. To extract useful components of software from existing software data a technique called Data Mining is used. It acts as a bridge between software repository and the knowledge required.² Using different techniques patterns are analyzed to make concrete decisions. For decision-making Machine Learning comes into light. This integration of data mining with artificial intelligence gives a new term Software Intelligence.³ Currently used softwares refuse algorithm that lack in the performance and do not give a precise result wheather the code can be reused or not.

This paper presents a software reuse integrated Random Forest and Gradient Boosting Machine (RFGBM) technique which will improve the Accuracy, Error rate, Kappa statistics, and other metrics of mining rate by using the R-tool. Computational speed is improved by removing the missing values using preprocessing filtering techniques. To validate the output of proposed algorithm comparison is done with eight classifiers named: J48, AdaBoostM1, LogitBoost, Part, One R, LMT, JRip, DecisionStump. Dataset used for experimentation is taken for UCI repository. It is created by National Institute of Engineering having 768 instances. Eight object-oriented attributes are used in the dataset.

Rest of the paper is organized as: Section 2 gives the basic terms related to software reuse. Description of the algorithms used to compare the results is also given. Literature survey is done in Section 3. Section 4 describes methodology. In Section 5 results are evaluated and Section 6 concludes the paper providing future scope.

2 | BACKGROUND

This section gives an overview of the basic concepts required to understand the software reuse process in detail. For software reuse, the core thing is source code. Along with using the code from existing source code software reuse process also consider the design using class libraries, application frameworks, and design patterns. Reuse of software benefits by increasing productivity, quality, reliability, and reduces of costs and time of implementation. The method of software reuse follows four steps.¹ The first step is to search for the components in the software repository. Second step is to understand and analyze the design structure of the programs. In the third step software reuse techniques are applied to get usable components from the repository. Last step gives final results by evaluating and testing. To get reusable components Data Mining¹ is used to obtain meaningful information from the inputted data. In this phase transformed data is analyzed using algorithms to get meaningful patterns and rules which further are used to make Predictive Models.⁴ Once patterns are created decisions are made with machine learning algorithms. The research on software reuse analytic focuses on testing whether software can be reused or not. Number of classifiers are available which helps in predicting the reusability factor of the software component. In this work eight classifiers are used to validate the result of proposed algorithm explained in coming sections.

2.1 | Software metrics and dataset used

Features are the measurable attributes of raw data.⁵ Machine learning works on these set of attributes or features to make decision. In software engineering these features are termed as metrics. To measure the reusability aspects software metrics are required. These metrics are evaluated from the classes, their properties, interconnection between classes, methods and functions of the source code.⁶ A good Empirical research is done on these software metrics.⁷⁻¹³ These metrics varies from procedure-oriented languages to object-oriented languages. We targeted eight software metrics in this paper that helps is measuring the reusability factors. These are Coupling, Cohesion, Depth Inheritance Tree (DIT), Bugs, Design Structure Quality Index (DSQI), Number of children (NOC), weighted method per class (WMP), Cyclomatic Complexity (CC). These metrics are summarized in Table 1.

Dataset used for experimentation is taken for UCI repository. It was created by National Institute of Engineering having 768 instances. Eight attributes are used in the dataset which are listed below in Table 2 and “Class” is the response variable. Class attribute has two values “0” and “1”. Where “0” interpret the component is Reusable and “1” indicate nonreusable component. Table 2 describes the mean and SD for these attributes.

2.2 | Prediction model algorithms

This section summarizes the prediction model used in the creating Integrated RFGBM.

TABLE 1 Reuse factors

Acronym	Attribute name	Definition	Remark
Coupling	Coupling	It is the degree to which the objects of one class depend on object of other class.	Reusability decreases when methods of one class interact with other class. Therefore, lower the coupling value higher the reusability.
Cohesion	Cohesion	It is the degree to which components are independent to perform a task.	Higher cohesion value gives high reusability.
NOC	Number of Children	Number of subclasses (child) to be inherited in the method of Parent Class	More the value of NOC, higher the reusable factor.
WMPC	Weighted Method per Class	It is the measure of complexity of class.	Higher the WMPC complex is the class, difficult to reuse. So lower WMPC is required to reuse the component.
CC	Cyclomatic Complexity	It indicates the complexity of methods used in particular class	Higher CC means code is difficult to maintain and test. So, to get better reusability value of CC should be low
DIT	Depth Inheritance Tree	Class level in the inheritance hierarchy	Higher DIT means more methods are inherited which makes the class behavior complex to measure. But also, more methods are inherited if DIT value is high.
BUGS	BUGS	Flaws/faults/ error in the code that makes code behave incorrectly	Value of Bugs should be low to get higher reusability.

TABLE 2 Dataset attributes statistics

	Coupling	Cohesion	DIT	NOC	BUGS	WMPC	DSQI	CC
Mean	3.8	120.9	69.1	20.5	79.8	32.0	0.5	33.2
SD	3.4	32.0	19.4	16.0	115.2	7.9	0.3	11.8
Min	0	0	0	0	0	0	0.078	21
Max	17	199	122	99	846	67.1	2.42	81

Abbreviations: CC, Cyclomatic Complexity; DIT, Depth Inheritance Tree; DSQI, Design Structure Quality Index; NOC, number of children; WMP, weighted method per class.

2.2.1 | Unsupervised filtering

Most of the Machine Learning algorithms use Supervised learning method. In supervised learning, we have input data (X) and a mapping function with which we can predict output variables (Y). On the other hand, if we are unaware about the output then Unsupervised Machine Learning methods are used. Clustering methods are examined using Unsupervised Methods. In this method clusters with similar objects are collected. The objects which do not belong to any cluster are considered as anomalies. So, using unsupervised filtering technique errors are removed from the dataset of software reuse.¹⁴ These error values are the anomalies that do not obey any cluster.

2.2.2 | Random forest

It is mostly used algorithm for its simplicity and ease to measure the prediction factors. It works on the concept of collection of large correlated decision trees (DTs).¹⁵ To get accurate and stable results Random Forest create multiple trees and then combine at last.

Random forest combines the decisions of individual trees and helps to improve the accuracy. Software reuse analytic approach uses the random forest along with gradient boosting machine (GBM) to improve the accuracy. It uses the bagging for randomness. Bagging stands for bootstrap aggregating in which accuracy and stability of the algorithm is improved.¹⁶ Mathematically, as

$$\text{bagging} = \frac{1}{B} \sum_{b=1}^B f_b(x'),$$

where x' is the predictions for unseen samples b is the number of trees that is, $b = 1, 2, 3 \dots B$; and f_b = Train a DT f_b on X_b, Y_b .

2.2.3 | Gradient boosting machine

To make model prediction GBM learning algorithm has become a popular. GBM works in three steps. Step 1: optimize the loss function. Step 2: predict the weaker learner. Step 3: create adaptive model by adding trees to the weaker learner, so as to reduce the loss function.

The major drawback of random forest is its speed. So, to avoid the issue of speed GBM is combined with it to get the better results for software reuse analytics. GBM is a technique used for regression and classification techniques for the better prediction results and also helps in reducing errors by decreasing bias.

2.3 | Overview of classifiers used in RStudio

Before discussing the results of different classifiers on our dataset, we will have overview to different classifiers algorithms that are used in this work. There are number of algorithms used for the classification purpose of software reuse analytic. The J48 and PART are mostly used algorithms for testing the reusability of software. Other algorithms used are as follows:

- I. **J48 Algorithm:** J48 is a DT that is implementation of ID3 algorithm. If we have a dataset in which there are predictors list, targets dependent or independent variables then j48 DT will allow you to find out the target variable automatically and make out tree of dataset also.¹⁷
- II. **AdaBoostM1 Algorithm:** It is Adaptive Boosting that is a machine learning meta-algorithm. It can be utilized along with several different types of understanding formulas to boost their performance.¹⁸
- III. **LogitBoost Algorithm:** This algorithm is used to create LR design. Designs are made at every node of the tree. After design C4.5 is used to separate the nodes resulting tree pruning.¹⁸
- IV. **Part Algorithm:** PART is a separate-and-conquer principle learner. The algorithm making pieces of principles called choice lists which are in the pipeline pair of rules. A new data is compared to each principle in the list consequently, and that is assigned the type of the initial corresponding rule.¹⁷
- V. **One R Algorithm:** OneR is just a very simple, quicker and one-level DT algorithm. It chooses one-by-one characteristics from the dataset and produces a different group of rules based on problem rate from the training set. Eventually, it decides the feature that gives rules with minimal problem and constructs the last DT. The algorithm sees loads of discrete attributes based on very simple association rules involving just one feature in situation part. It generally realizes attribute fast and then produces easy rule and estimate its error.¹⁵
- VI. **LMT Algorithm:** LMT stands for Logistic Model tree. It is classifier which combines DT learning and logistic regression (LR).¹⁸
- VII. **JRip Algorithm:** JRip (RIPPER) is a standard algorithm used for setting rules. It uses incremental reduce error to identify all members of classes.¹⁹
- VIII. **DecisionStump Algorithm:** A decision stump is a one level DT algorithm. It make decision using only one input value.²⁰

3 | RELATED WORK

Cai et al.²¹ gave Component-Based Software Engineering (CBSE) Model which focused on using the reusable components by making design and constructing computer based. It was more as theory. The research does not focus on practical aspects. General, technical, and nontechnical software issues were given by Kim et al.²² Wangoo et al.³ in 2018 analyzed three artificial intelligence techniques discussing the significance of artificial intelligence in software reuse. Parkash et al.¹ prepared 167 instances of dataset using open source projects. Researchers applied clustering and classification techniques

to identify reusable components. A Reverse Support System was introduced by the researchers which elucidate the design and document of existing software. The outcome of the system helped to rebuild a new software. As this work only gave description of the existing software ignoring the construction of new software from existing one. Sandhu et al.²³ proposed an approach to reuse object-oriented software components which worked on hybrid K-means and DT to predict reusability value. Results produced high precision and recall values with lower error rates. Rajkumari et al.²⁴ in 2019 worked on code clones in software reuse process. Researchers proposed a token-based code clone method to remove Type-I (exact clone) and Type-II (Functional clone). Sidhu et al.⁵ introduced machine-learning UML model to perform software refactoring. Their major research focused on detecting design flaws at higher level of granularity. Julia et al.²⁵ in 2015 compared four software reuse strategies for embedded and nonembedded systems. The researchers concluded that reuse of embedded system code does not benefit much. Padhy et al.²⁶ introduced an aging and surveillance aware resolving optimal model (ASROM). But this model only focused on web service softwares. Also, the algorithm worked on Line of code. Kaur et al.¹⁹ used J48 learning algorithm to detect the design patters of the code. When large number of code of lines were applied on the algorithm, its performance degrades which gave rise to maintenance problems. Zhang et al.²⁷ focused on to make better CP and PLM decisions based on the big amount of real-time data and multisource heterogeneous big data. Ratzinger et al.²⁸ in their findings concluded that the bugs rate decreases when the software refactoring rate is increased. Rodríguez et al.¹⁴ focused on complexity of Service-Oriented Computing (SOC). An unsupervised learning method was proposed by the authors for SOC which used clustering and visualization to evaluate the results. A learning-based approach named CREC was proposed by Yue et al.²⁹ in their work. It focused on extracting the features from the past and current history of software repository. Machine learning named DT and Layer Recurrent Neural Network (LRNN) was used by Dwivedi et al.³⁰ to create a reusable document.

Table 3 summarizes some of the work related to reuse of code using machine learning algorithms. The datasets used by the researchers for performing prediction to check the reusability are also given along with prediction model and algorithms used.

TABLE 3 Prediction model used for software reuse

References	Year	Prediction model	Algorithm used	Dataset used	Outcome
28	2008	Classification	LMT, C4.5, Rip, NNge	XDclet, ArgoUML, Liferay Portal, Spring Framework, JBoss Cache	Authors concluded that with the increase in software refactoring rate, the bugs rate decrease.
23	2010	Clustering	Hybrid K-means, Decision Tree	NASA Repository	It was observed that embedded systems are not benefited with reuse process.
1	2012	Classification/ Clustering	REP, J48	167 instances from open source project (sourceforge.net)	A Reverse Support System was introduced by the researchers which elucidate the design and document of existing software
15	2015	Classification/ Clustering	CLOPE, Decision Tree, Random Forest, OneR, Support Vector Machine, JRip, K-means, ZeroR	10 open-source software systems	TO evaluate quality of instance MARPLE-DPD was developed
14	2016	Regression	partitioning around medoids (PAM), X-Means, K-means, COBWEB	Dataset from WSDL Document	unsupervised learning method was proposed by the authors for SOC which used clustering and visualization to evaluate the results.
30	2016	Classification	Layer Recurrent Neural Network (LRNN) and Decision Tree	Data used from Source Code	Used machine learning approach to design patters
19	2018	Classification	J48	Data used from Source Code	Used prediction algorithm to detect the design patters of the code.
29	2018	Classification	C4.5, Sequential minimal optimization (SMO), Naïve Bayes (NB).	Eclipse.jdt.core, Elastic Search, Axis2, JRuby, Lucene	Authors suggested the extraction of features from the past and current history of software repository.

Abbreviation: LMT, Logistic Model tree.

From the literature survey it was found that mostly theoretical models for software reuse are given by the researchers. Some of the gaps found in the literature survey are listed as follow where improvement can be done.

1. Software reuse analytics is not covered in the review.
2. One of the major gaps found in work is Computational speed.
3. When large number of code of lines are implemented on the models, the performance is degraded.

To overcome the listed issues a machine learning technique is proposed which works on random forest-based algorithm for software reuse analytics to enhance the performance.

4 | PROPOSED METHODOLOGY

The proposed research methodology is presented in this section. R tool is used to validate performance. The methodology contains two parts: Training data and Testing data. As mentioned in Section 2.1 dataset is taken from UCI repository. From total of 768 instances data is divided into two parts for training and testing taking different split ratios. Dividing is done to validate the performance of proposed algorithm. Working of algorithm is presented in the form of flow chart in Figure 1.

The steps of flow chart for training data are given as follow:

- a. For training data
 - Step 1:* load software reuse data from the dataset for training the data.
 - Step 2:* Apply unsupervised filtering on data for unknown clusters.
 - Step 3:* Apply hybrid random forest and neural network for accuracy and performance, respectively.
 - Step 4:* So, finally get the training data with potentially predictive relationships among the data.
- b. For testing data: Once the components are extracted, they are passed through testing phase so as to know whether component can be reused or not. Figure 2 shows the working of testing phase.
 - Step 1:* Load software reuse data for testing the data.
 - Step 2:* Same as training data, here also apply unsupervised filtering for unknown data.
 - Step 3:* Apply random forest for constructing trees and use gradient boosting for speed. Returns the final prediction as output with accurate results.
 - Else
mark it as nonreusable and return the results.
 - Step 4:* If the software is reusable then mark it as reusable and return the results.

The pseudocode of the proposed algorithm is explained in simple layman language. The label “INPUT” is used to represent the inputs given to the algorithm. OUTPUT for the algorithm in the “Decision tree” and the “METHOD” label gives the working of the algorithm.

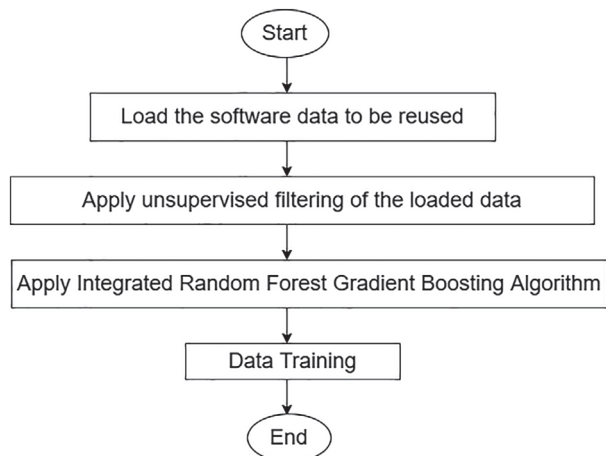
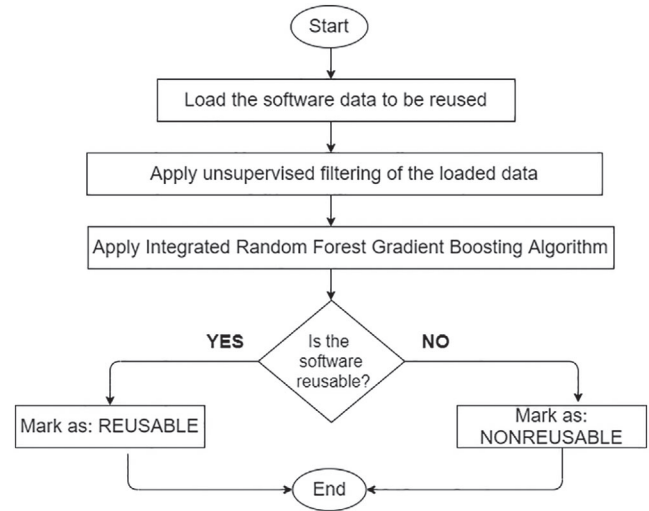


FIGURE 1 Flow chart for training data

FIGURE 2 Flow chart for testing data**INPUT:**

1. X = Data partition
- X is the set of attributes in the data set and their associated labels of classes.
2. A = Class
 3. List_of_attributes = set of candidate attributes
 4. Selection_method = best fit partition process of splitting data attributes to individual class.
- Here in the attribute selection process : split_criteria and attribute_split are used.

5 | METHOD

1. Create Node N
2. If (X attribute belong to same class A) Then
RETURN: N = leaf node labelled with Class A
3. If (list_of_attribute = NULL) Then
RETURN: N = leaf node labelled with majority in class A
4. Apply Selection_Method (X , list_of_attribute)
 N = best split_criteria
If (multiway splitting allowed AND Spilt_attribute = descrete Value) then
list_of_attribute = split_attribute
5. For each outcome
(i of split_criteria)

Let X_i = set of attributes in X satisfying outcome i

If (X_i = NULL) Then

To the node N , leaf node will be attached with majority class in X

Else

To node N the generated Tree (X_i , list_of_attribute) will be attached

END for

RETURN N ;

OUTPUT:

Decision Tree

Cross-validation of 10-folds is performed. Value of tuning parameter is set to constant of 0.1. The resampling of the results for tuning parameters is given in Table 4. Where column one represents iteration depth, number of trees are given in column two and the value for accuracy and kappa statics is evaluated in columns 3 and 4.

From the above table final values of model noted are the number of trees = 100, interaction. depth = 1, tuning parameter = 0.1.

The boosting iterations using different values for max tree depth for accuracy is shown in Figure 3. For the same, kappa statistics values are shown in Figure 4.

Parameters taken:

This part gives a brief of the parameters taken to evaluate the results. Total seven parameters are taken. These are: Accuracy (Correctly Classified Instances), Kappa statistic, Error rate (Incorrectly Classified Instances), Mean absolute error, Relative absolute error, Root relative squared error, Root mean squared error.

Confusion Matrix:

Confusion Matrix is a performance measurement table used for machine learning classification problem, also known as error matrix. It basically shows how many times the classification model was confused while making the predications.³¹ Matrix evaluate the performance by checking the actual values and the predicated values.³² A visualized matrix is shown in Table 5. Confusion matrix uses four parameters.

Interaction depth	Number of trees	Accuracy	Kappa
1	50	0.7599301	0.4396229
1	100	0.7608456	0.4509605
1	150	0.7577079	0.4464262
2	20	0.7576974	0.4433047
2	100	0.7532669	0.4392949
2	150	0.7501083	0.4348482
3	50	0.7560377	0.4438508
3	100	0.7474948	0.4260947
3	150	0.7419217	0.4172719

TABLE 4 Tree pruning statistic of model

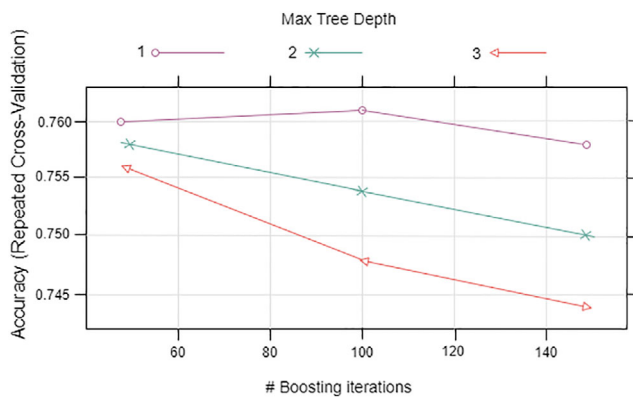


FIGURE 3 Boosting iterations for accuracy with maximum value 0.760 [Colour figure can be viewed at wileyonlinelibrary.com]

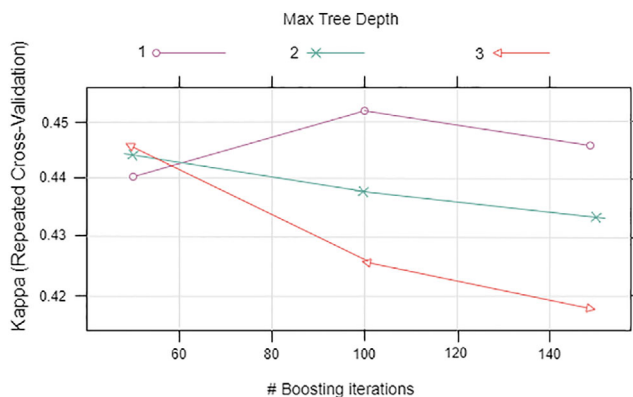


FIGURE 4 Shows boosting iterations for kappa statistics with maximum value more than 0.44. Comparison of proposed algorithm is done with existing Logistic Model tree, JRip, PART, DecisionStump, J48, AdaBoostM, LogitBoost, Decision Stump and OneR classifiers. The comparison is depicted in the form of bar graphs from Figures 5–11 [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 5 Confusion matrix

n = number of instances	Predicted: True	Predicted: False
Actual: True	TP Correct Prediction	FN Incorrect Prediction
Actual: False	FP Incorrect Prediction	TN Correction Prediction

P (Positive): observation is True.

N (Negative): observation is False.

TP (True Positive): Predicted observation of Model is True and the actual observation is also true. The model gave correct prediction.

FP (False Positive): Predicted observation of Model is True but the actual observation is False. The model gave incorrect prediction.

TN (True Negative): Predicted observation of Model is False and the actual observation is also False. The model gave correct prediction.

FN (False Negative): Predicted observation of Model is False but the actual observation is True. The model gave incorrect prediction.

These parameters used for evaluating the results are summarized in the Table 6 with a general definition, mathematical representation and relation of these parameters to the performance of model.

6 | RESULT AND DISCUSSIONS

For testing of proposed algorithm, the training dataset is taken from UCI Repository. The training dataset has been used to test the working of base algorithms. The analysis of proposed algorithm is done on the basis of seven parameters mentioned in Table 6. The experimentation further compares performance evaluation of proposed algorithm with various classifiers which are LMT, JRip, PART, DecisionStump, J48, AdaBoostM, LogitBoost, Decision Stump, and OneR.

Among all the classifiers, J48 has the highest performance than all other classifiers. Table 7 shows the comparison of proposed algorithm with other algorithms. The results of comparisons are presented in the form of Bar graphs. There is

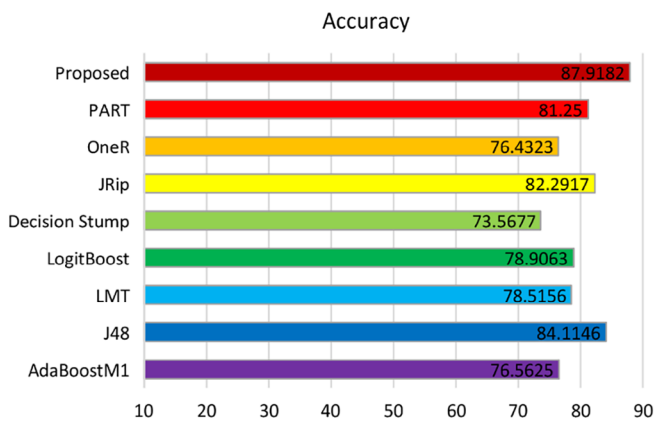
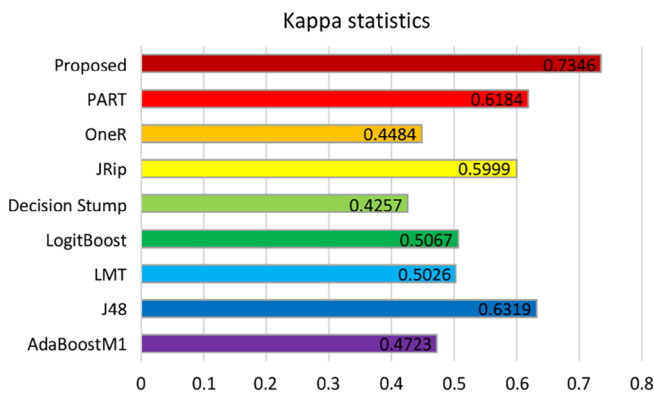
TABLE 6 Various parameters used for implementation

Parameters	Mathematical representation	General definition	Relation to the performance
Accuracy	$\text{Accuracy} = \left(\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \right)$	It is rate of correctly predicted observations over the given dataset by the classifier.	To enhance the performance, the value of accuracy should be maximized.
Error rate	$\text{Error rate} = 1 - \left(\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \right)$	It is rate of incorrectly predicted observations over the given dataset by the classifier.	Error rate needs to be minimized for better results.
Kappa Statistics	$\text{Kappa} = \frac{\text{Total accuracy} - \text{Random accuracy}}{1 - \text{Random accuracy}}$	It is used to compare the observed accuracy with random accuracy that is expected accuracy.	Kappa depends upon the accuracy. So, if accuracy is high, the value of kappa will also high.
Mean Absolute Error	$\text{MAE} = \frac{1}{N} \sum_{i=1}^N \text{Predicted}_i - \text{Actual}_i $	It measures the mean magnitude of errors. This parameter does not consider the direction.	Mean Absolute error should be minimized.
Root Mean Squared Error	$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^N (\text{Predicted}_i - \text{Actual}_i)^2}{N}}$	It considers the direction of errors for measuring the mean magnitude. This parameter avoids large errors.	Root Mean Squared Error needs to be minimized.
Relative Absolute Error	$\text{RAE} = \frac{\sum_{i=1}^N \text{predicted}_i - \text{Actual}_i }{\sum_{i=1}^N \theta_i - \text{Actual}_i }$	It is a ratio of mean error to the error predicted by the trivial model.	Relative Absolute Error Should also be minimized.
Root Relative Squared Error	$\text{RRSE} = \sqrt{\frac{\sum_{i=1}^N (\text{Predicted}_i - \text{Actual}_i)^2}{\sum_{i=1}^N (\theta_i - \text{Actual}_i)^2}}$	It measures the SD of prediction error.	Root Relative Squared Error needs to be minimized for better results.

TABLE 7 Comparison of various existing classifiers on the basis of some parameters with the proposed algorithm

	AdaBoostM1	J48	LMT	LogitBoost	Decision stump	JRip	OneR	PART	Proposed RFGBM algorithm
Accuracy	76.5625	84.1146	78.5156	78.9063	73.5677	82.2917	76.4323	81.25	87.9182
Error rate	23.4375	15.8854	21.4844	21.0938	26.4323	17.7083	23.5677	18.75	12.0818
Kappa statistics	0.4723	0.6319	0.5026	0.5067	0.4257	0.5999	0.4484	0.6184	0.7346
Root relative squared error	82.2787	72.4207	81.9891	79.251	90.4671	79.0719	101.8515	73.6772	64.2588
Mean Absolute error	0.2956	0.2383	0.3069	0.2853	0.3719	0.2841	0.2357	0.2466	0.1877
Relative absolute error	65.0313	52.4339	67.523	62.7697	81.8217	62.5074	51.8551	54.2692	41.2768
Root mean squared error	0.3922	0.3452	0.3908	0.3777	0.4312	0.3769	0.4855	0.3512	0.3064

Abbreviation: LMT, Logistic Model tree.

**FIGURE 5** Comparison of AdaBoostM1, J48, Logistic Model tree, LBst, Decision Stump, JRip, OneR, PART and proposed algorithm with accuracy percentage increase of 4.5% as compared to the existing algorithms [Colour figure can be viewed at wileyonlinelibrary.com]**FIGURE 6** Comparison of AdaBoostM1, J48, Logistic Model tree, LBst, Decision Stump, JRip, OneR, PART and proposed algorithm with kappa statistics increase of 16% to that of existing algorithms [Colour figure can be viewed at wileyonlinelibrary.com]

an increase of 4.5% in accuracy (shown in Figure 5) and 16% increase in kappa value (shown in Figure 6) when the dataset is applied on the proposed integrated random forest gradient boosting algorithm with respect to existing AdaBoostM1, J48, LMT, LBst, Decision Stump, JRip, OneR, PART, and Proposed Algorithms.

Figure 7 depicts that proposed algorithm showed 23% decrease in error rate when compared to existing algorithms. In Figure 8 and Figure 9 Mean Absolute Error and Relative Absolute Error decreased to an average of 21% when the proposed Integrated Random Forest Gradient Boosting algorithm compared with existing AdaBoostM1, J48, LMT, LBst, Decision Stump, JRip, OneR, PART algorithms.

FIGURE 7 Comparison of AdaBoostM1, J48, Logistic Model tree, LBst, Decision Stump, JRip, OneR, PART and proposed algorithm with error rate percentage decrease of 23% as compared to the existing algorithms [Colour figure can be viewed at wileyonlinelibrary.com]

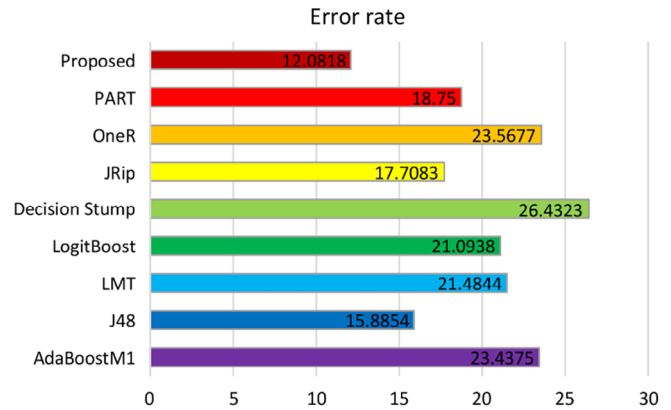


FIGURE 8 Comparison of AdaBoostM1, J48, Logistic Model tree, LBst, Decision Stump, JRip, OneR, PART and proposed algorithm with mean absolute error percentage decrease of 21% to that of existing algorithms [Colour figure can be viewed at wileyonlinelibrary.com]

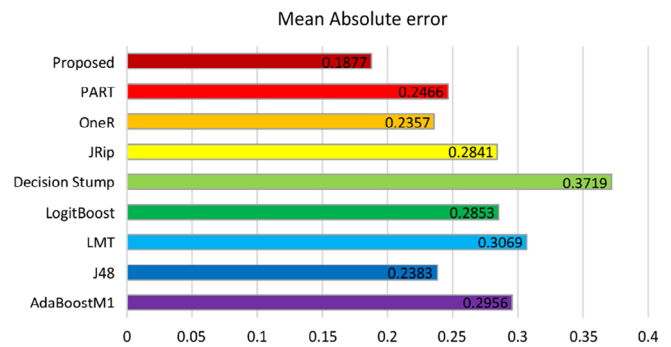
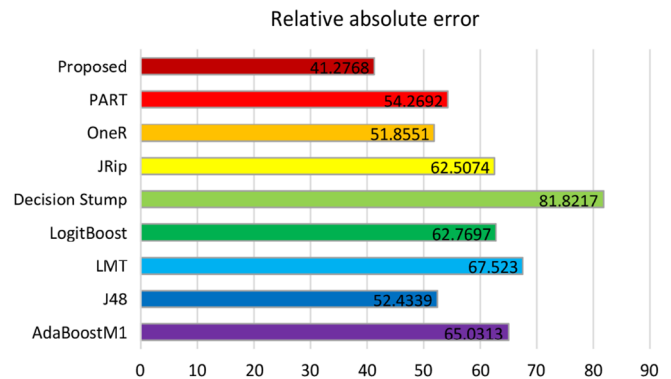


FIGURE 9 Comparison of AdaBoostM1, J48, Logistic Model tree, LBst, Decision Stump, JRip, OneR, PART and proposed algorithm with relative absolute error percentage decrease of 21% as compared to the existing algorithms [Colour figure can be viewed at wileyonlinelibrary.com]



Also Root Relative Squared Error, shown in Figure 10 decreased to 10% and Root mean square error went down to 12%, shown in Figure 11, when the results of proposed algorithm compared with existing AdaBoostM1, J48, LMT, LBst, Decision Stump, JRip, OneR, PART.

7 | CONCLUSION

In our everyday life, software is initiating to be noticed as the core of most of the enterprises, economic and the social situations. Software reuse has a significant impact of software enterprise, but reviews show there in not much work is done in the field of software reuse. This paper presented an algorithm for testing the reusability on basis of some parameters such as bugs, DIT, cohesion and coupling, which uses integrated RFGBM technique. The proposed algorithm predicted whether the given software is reusable or not. This algorithm improved the performance by increasing accuracy and decreasing error rate and also improved the values of other performance measure parameters. The validation of proposed algorithm was done by comparing it with J48, AdaBoostM1, LogitBoost, Part, One R, LMT, JRip, and DecisionStump algorithms Computational speed was one of the challenging issues in previous works. The proposed algorithm helped to increase the computational speed. On an average there was an increase of 23% in performance of proposed RFGBM

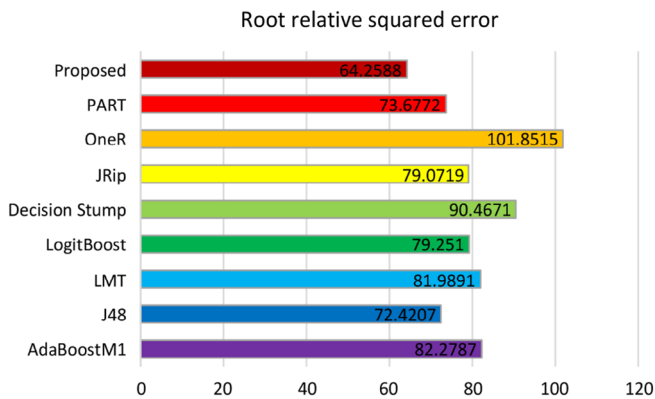


FIGURE 10 Comparison of AdaBoostM1, J48, Logistic Model tree, LBst, Decision Stump, JRip, OneR, PART and proposed algorithm with root relative squared error percentage decrease of 10% as compared to the existing algorithms [Colour figure can be viewed at wileyonlinelibrary.com]

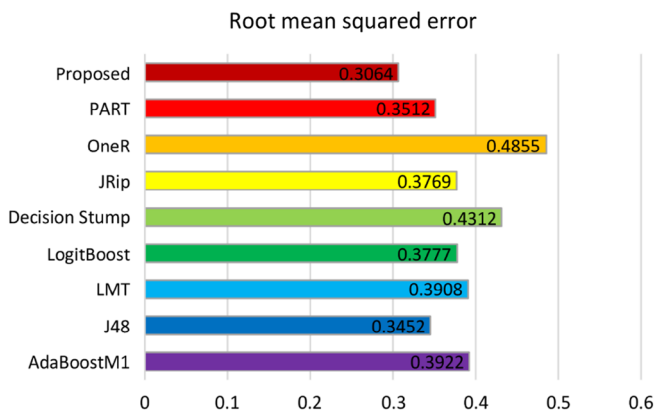



FIGURE 11 Comparison of AdaBoostM1, J48, Logistic Model tree, LBst, Decision Stump, JRip, OneR, PART and proposed algorithm with root mean square error decrease of 12% as compared to the existing algorithms [Colour figure can be viewed at wileyonlinelibrary.com]

algorithm to that of existing algorithms. The proposed technique does not guarantee the higher accuracy rate because certain constants are assigned manually. Therefore, in near future meta-heuristic techniques based deep learning techniques will be designed to enhance the performance and achieve better accuracy.

ORCID

Ranbir Singh Batth  <https://orcid.org/0000-0002-8655-7613>

REFERENCES

- Prakash BVA, Ashoka DV, Aradhya VNM. Application of data mining techniques for software reuse process. *Proc Technol*. 2012;4:384-389.
- Iqbal MS, Luo B, Khan T, Mehmood R, Sadiq M. Heterogeneous transfer learning techniques for machine learning. *Iran J Comput Sci*. 2018;1:31-46. <https://doi.org/10.1007/s42044-017-0004-z>.
- Wangoo DP. Artificial intelligence techniques in software engineering for automated software reuse and design. In: 2018 4th International Conference on Computing Communication and Automation (ICCCA); December 14-15, 2018; Greater Noida, India.
- Iqbal MS, El-Ashram S, Hussain S, et al. Efficient cell classification of mitochondrial images by using deep learning. *J Opt*. 2019;48:113-122. <https://doi.org/10.1007/s12596-018-0508-4>.
- Sidhu BK, Singh K, Sharma N. A machine learning approach to software model refactoring. *Int J Comput Appl*. 2020;1-12. <https://doi.org/10.1080/1206212X.2020.1711616>.
- Cooper KML. Object oriented analysis and design. In: Wah BW, ed, *Wiley Encyclopedia of Computer Science and Engineering*. Wiley; 2009. <https://doi.org/10.1002/9780470050118.ecse278>.
- Murillo-Luna JL, Garcés-Ayerbe C, Rivera-Torres P. Barriers to the adoption of proactive environmental strategies. *J Clean Prod*. 2011;19(13):1417-1425.
- Ngai, E. W., Xiu, L., Chau, D. C., (2009). Application of data mining techniques in customer relationship management: a literature review and classification. *Expert Syst Appl Ther* 2009, 36(2), 2592-2602.
- Rabl T, Gómez-Villamor S, Sadoghi M, Muntés-Mulero V, Jacobsen H-A, Mankovskii S. Solving big data challenges for enterprise application performance management. *Proc. VLDB Endow*. 2012;5(12):1724-1735.
- Silva DA, Delai I, Castro MA, Ometto AR. Quality tools applied to cleaner production programs: a first approach towards a new methodology. *J Clean Prod*. 2013;47:174-187.
- Vinodh S, Prakash NH, Selvan KE. Evaluation of agility in supply chains using fuzzy association rules mining. *Int J Prod Res*. 2011;49(22):6651-6661.

12. Wang HW, Chen S, Xie Y. An RFID-based digital ware-house management system in the tobacco industry: a case study. *Int J Prod Res.* 2010;48(9):2513-2548.
13. Wei FF. ECL Hadoop: "big data" processing based on Hadoop strategy in effective e-commerce logistics. *Comput Eng Sci.* 2013;35(10):65-71.
14. Rodríguez G, Soria Á, Teyseyre A, Berdun L, Campo M. Unsupervised learning for detecting refactoring opportunities in service-oriented applications. In: International Conference on Database and Expert Systems Applications. September 5-8, 2016; Porto, Portugal: Springer: 335-342
15. Zaroni M, Fontana FA, Stella F. On applying machine learning techniques for design pattern detection. *J Syst Softw.* 2015;103:102-117. <https://doi.org/10.1016/j.jss.2015.01.037>.
16. Sidiq SJ, Zaman M, Butt M. An empirical comparison of classifiers for multi-class imbalance learning. *Int J Data Mining Emerg Technol.* 2018;8(1):115-122.
17. Di Stefano JS, Menzies T. Machine learning for software engineering: case studies in software reuse. In: Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02), 2002; November 4-6, 2002; Washington, DC.
18. Landwehr N. *Thesis on Logistic Model Trees.* Germany: University of Freiburg; 2003. https://www.cs.uni-potsdam.de/ml/landwehr/diploma_thesis.pdf. Accessed February 22, 2020.
19. Kaur A, Singh S. Detecting software bad smells from software design patterns using machine learning algorithms. *Int J Appl Eng Res.* 2018; 13:10005-10010.
20. Gupta DL, Saxena K. AUC based software defect prediction for object-oriented systems. *Int J Current Eng Technol.* 2016;6(5):1728-1733.
21. Cai X, Lyu MR, Wong K, Ko R. Component-based software engineering: technologies, development frameworks, and quality assurance schemes. In: Proceedings of the Seventh Asia-Pacific APSEC; 2000: 372-379.
22. Kim Y. Software reuse: issues and research directions. Center for Digital Economy Research Stem School of Business; 1991. Working Paper IS-9 1-15.
23. Shri A, Sandhu PS, Gupta V, Anand S. Prediction of reusability of object oriented software systems using clustering approach. *World Acad Sci Eng Technol.* 2010;43:853-856.
24. Rajakumari KE. Comparison of token-based code clone method with pattern mining technique and traditional string matching algorithms in terms of software reuse. In: IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2019; 2019; Coimbatore, India.
25. Varnell-Sarjeant J, Andrews AA. Comparing reuse strategies in different development environments. *Adv Comput.* 2015;97:1-47.
26. Neelamdhav P, Singh RP, Satapathy SC. Software reusability metrics estimation: algorithms, models and optimization techniques. *Comput Electric Eng.* 2018;69:653-668.
27. Zhang Y, Reh S, Yang L, Si S. A big data analytics architecture for cleaner manufacturing and maintenance processes of complex products. *J Clean Prod.* 2017;142(2):626-641. <https://doi.org/10.1016/j.jclepro.2016.07.123>.
28. Ratzinger J, Sigmund T, Gall HC., (2008). On the relation of refactoring and software defects. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories; 2008; Leipzig, Germany: ACM: 35-38.
29. Yue R, Gao Z, Meng N, Xiong Y, Wang X. Automatic clone recommendation for refactoring based on the present and the past. In: IEEE International Conference on Software Maintenance and Evolution (ICSME); September 23-29, 2018; Madrid, Spain: IEEE: 115-126.
30. Dwivedi AK, Tirkey A, Ray RB, Rath SK. Software design pattern recognition using machine learning techniques. In: IEEE Region 10 Conference (TENCON); November 22-25, 2016; Singapore, Singapore: IEEE: 222-227.
31. Sidiq SJ, Zaman M, Ashraf M, Ahmed M. An empirical comparison of supervised classifiers for diabetic diagnosis. *Int J Adv Res Comput Sci.* 2017;8(1):311-315.
32. Iqbal MS, Khan T, Kausar S, Bin L. Analysis of bioenergy by using linear regression. *SN Appl Sci.* 2019;1:1225. <https://doi.org/10.1007/s42452-019-1270-1>.

How to cite this article: Sandhu AK, Batth RS. Software reuse analytics using integrated random forest and gradient boosting machine learning algorithm. *Softw: Pract Exper.* 2021;51:735-747. <https://doi.org/10.1002/spe.2921>