Asia Pacific Journal of Information Systems
Vol. 30 No. 3 (September 2020), 457-496

ISSN 2288-5404 (Print) / ISSN 2288-6818 (Online)
https://doi.org/10.14329/apjis.2020.30.3.457

# A Pragmatic Framework for Predicting Change Prone Files Using Machine Learning Techniques with Java-based Software

Loveleen Kaur[a,*], Ashutosh Mishra[b]

[a] *Ph.D. Research Scholar, Department of Computer Science and Engineering, Thapar University, Patiala, India*
[b] *Assistant Professor, Department of Computer Science and Engineering, Thapar University, Patiala, India*

**ABSTRACT**

This study aims to extensively analyze the performance of various Machine Learning (ML) techniques for predicting version to version change-proneness of source code Java files. 17 object-oriented metrics have been utilized in this work for predicting change-prone files using 31 ML techniques and the framework proposed has been implemented on various consecutive releases of two Java-based software projects available as plug-ins. 10-fold and inter-release validation methods have been employed to validate the models and statistical tests provide supplementary information regarding the reliability and significance of the results. The results of experiments conducted in this article indicate that the ML techniques perform differently under the different validation settings. The results also confirm the proficiency of the selected ML techniques in lieu of developing change-proneness prediction models which could aid the software engineers in the initial stages of software development for classifying change-prone Java files of a software, in turn aiding in the trend estimation of change-proneness over future versions.

*Keywords:* Software Component, Software Change, Source Code Metrics, Software Prediction, Machine Learning

## Ⅰ. Introduction

Software is relentlessly predisposed to changes which are obligatory to acclimate to a different environment, to add new features, refactor the source code, or to fix bugs (Purushothaman and Perry, 2005; Ying et al., 2004). An effective software change prediction mechanism predicts those source code elements that are likely to be employed with some change from one version of software to the next. In an actual scenario, change prediction models are capable of being directly incorporated in software developers' analytics dashboards (e.g., BITERGIA[1]) via which these models could help in providing a continuous

*Corresponding Author. E-mail: loveleen.kaur@thapar.edu

feedback on the code that is more prone to undergo some change in the future (Beller et al., 2017; Catolino and Ferrucci, 2018). This feedback can be act as input for executing preventive maintenance activities before putting the code into production. For example, in a continuous integration (CI) milieu, software developers may want to refactor the source code before the CI pipeline starts with the goal of evading code quality-related warnings by build failures or static analysis tools (Catolino et al., 2018; Vassallo et al., 2018). Likewise, software change prediction models might prove to be useful for project managers for the purpose of properly scheduling maintenance activities.

The last two decades have witnessed many researchers employing various methods for conducting an empirical study to investigate and confirm the capability of Object Oriented (OO) metrics in predicting change-prone code elements (Giger et al., 2012; Honglei et al., 2009; Malhotra and Bansal, 2014; Malhotra and Khanna, 2013; Malhotra and Khanna, 2017; Romano and Pinzger, 2011; Zhou et al., 2009). However, there exists a necessity to re-evaluate this topic for the following reasons (Lessmann et al., 2008):

- Assessment of the performance proficiency of the change-proneness prediction models is essential for assessing their pragmatic pertinence. Although the model results have been depicted using various metrics in literature, the analysis or the conclusions have been drawn based on a single performance measure.
- The authenticity of empirical analyses can be only ascertained via statistical testing (Arcuri and Briand, 2011; Menzies et al., 2007). Some studies call attention to the fact that statistical significance of the experimental results obtained in software prediction is hardly inspected (Myrtveit et al., 2005).
- Former findings validate the generated predictive models on the same data that was used for their training and consequently do not study the efficacy of these models to predict the trend of change-proneness of the components across imminent versions of a software project.
- There are other ML models and OO metrics in literature that could produce better results and still remain unexplored as far as change-proneness prediction is involved.

Apropos to this, the investigation carried out in this research article concentrates on the development of version to version change-proneness prediction models with the aid of 31 ML techniques including those (Islam and Giggins, 2011; Shalev-Shwartz et al., 2011; Ting and Witten, 1997) which have been not analysed previously for change-proneness prediction. With the intention of conducting an empirical validation, successional releases of two Java-based plugin projects "JFreeChart" and "Heritrix" have been selected as target projects. Numerical values corresponding to 17 OO source code metrics are calculated for every Java file present in each of the selected versions of the two projects. These numerical values of the metrics corresponding to an individual file, together with the change statistic collectively yield data points. The relative performance of the built change-proneness prediction models is quantified using six performance measures. Additionally, this work also conducts an inter-release validation of the models with the purpose of examining the efficiency of the selected ML techniques in predicting the trend of change-proneness of files in the upcoming versions.

1) https://bitergia.com

As there exists a need for verifying the comparative disparity among performances of the generated models over various validation scenarios, Kruskal Wallis test and the Scott-Knott cluster analysis are applied to ascertain if there exists a statistical dissimilarity among the predictive performances of the selected ML techniques. The pragmatic assessment conducted in this article therefore aids in supplying valid answers to the research questions (RQs) stated as follows:

- RQ1: What is the predictive capability of the various ML techniques, by and large, with respect to predicting version to version change-proneness of Java files on the various releases of the two projects when 'k'-fold cross-validation with feature selection is employed?
- RQ2: What is the performance of the models with respect to predicting the trend of version to version change proneness of files?
- RQ3: Is the predictive performance of the change-proneness prediction models developed via k-fold cross-validation statistically similar to or different from the performance of the models constructed by means of an inter-release validation?
- RQ4: Which are the best and the worst techniques for change-proneness prediction of files of the selected projects?

We believe that no former research has been performed which specifically: (1) Conducts a broad comparative analysis of statistical approaches and ML techniques in the context of version to version change-proneness prediction, (2) Employs data gathered from multiple releases of two plugin projects using 17 OO metrics to obtain generalized and unbiased results, (3) Statistically analyses the attained results using multiple statistical tests for the performance comparison of the selected prediction techniques, and (4) Performs an inter-release validation of the models with the purpose of examining the efficiency of the selected techniques in predicting the trend of change-proneness of Java files in the upcoming versions. Moreover, the testimony acquired from such exhaustive data-based comparative pragmatic analyses can assist the software researchers and practitioners to cultivate ample corpus of knowledge to accept/reject a given hypothesis (Aggarwal et al., 2009).

The rest of the paper have been organized as follows: Section 2 comprises of a summary elaborating the related work of the study and Section 3 reports the pragmatic framework designed for change-proneness prediction along with the dependent and independent variables incorporated, target projects employed in the study, and empirical data collection. Section 4 elaborates the experimental setup of the study while Section 5 states and discusses the results obtained in accordance with each of the given RQs. Section 6 scrutinises the different threats to validity of our work. To close, Section 7 re-counts the conclusions of the analysis performed and proposes future work.

## Ⅱ. Related Work

Wide array of techniques have been employed to create change-proneness prediction models in the existing literature. For example, Romano and Pzinger (2011) employed Naive Bayes (NB), Support Vector Machine (SVM), and Neural Nets (NN) on eight Eclipse and two Hibernate datasets for predicting change-prone interfaces using ten-fold cross validation. Giger et al. (2012) investigated the Bayesian networks

(BN) and NNs to identify if a code file gets affected by a particular kind of code change, via the static source code dependency graph of 19 plugin projects of Eclipse and Azureus software. An exploration of the AUC values generated using ten-fold cross validation concluded that the performance of the prediction techniques varies between the categories of changes that concern the body or declaration of a method or class. Malhotra and Bansal (2005) employed techniques like AdaBoost (ADB), Bagging, NB, LogitBoost (LB), MultiLayer Perceptron (MLP), Classification and Regression Trees (CART), and Random Forest (RF) to identify change prone classes using a threshold methodology. The models were applied on Freemind software projects and validated on one version of Frinika software. The results indicated CART algorithm as the best performer for inter-release validation and NB for inter-project validation. Kumar et al. (2017) examined five types of feature selection techniques and ten ML algorithms for constructing efficient change-proneness prediction models. Two versions of Eclipse were used to conduct the study, and the results indicated that change prediction models can achieve higher accuracies when the feature selection methods are applied as compared to simply using the metrics as it is. Recently, Catolino and Ferrucci (2018) performed an extensive comparison between the prediction performances of standard machine learning classifiers (ie., LR, NB, Simple Logistic, and MLP) with four ensemble techniques (ie., Boosting, RF, Voting, and Bagging). The study utilized 33 releases of 10 open-source systems and the results indicated the superiority of ensemble methods and in particular RF to predict software change in terms of F-measure.

Apart from the application of the ML techniques, some of the authors only utilize the statistical techniques for estimating change-prone components. For example, Chaumun et al. (2002) employed the statistical technique of ANOVA to establish an association between the selected CK metrics and the impact of change on a system when a method's signature in its body of code is changed. Lu et al. (2012) utilized 102 Java-based software systems to assess the association between change-proneness and 62 OO metrics by means of the statistical random effect technique. Elish and Al-Zouri (2014) employed standard Logistic Regression (LR) techniques developed on the principle of maximum likelihood estimation for predicting version to version change-proneness of classes using coupling metrics. Datasets were created from successive versions of Stellarium and LabPlot (both C++ based open-source systems) and the models were evaluated using accuracy as the performance measure.

Certain studies related to change prediction were also found which consisted of a pragmatic comparative analysis between the statistical and the ML techniques. Malhotra and Jangra (2013) evaluated and equated the predictive performance of a statistical methodology with ten ML techniques using datasets generated from two releases each of SweetHome-3D and Art-of-Illusion. Malhotra and Khanna (2013) examined two releases each of OrDrumbox, FreeMind and, Frinika to investigate the performances of LR, RF, Bagging and MLP. AUC was employed as the measure for comparing the selected techniques. It was stated by both the articles (Malhotra and Jangra, 2013; Malhotra and Khanna, 2013) that majority of the ML methodologies, especially RF, considerably do better than the statistical LR methodology. Malhotra and Khanna (2014) established the WMC, SLOC and the CBO metric to be competent change-proneness predictors on datasets generated using drJava, DSpace and Robocode and the results were compared to the statistical technique of LR via Freidman test and ten-fold cross validation.

ML techniques like MLP, Group Method of Data Handling (GMDH), J48 and Bagging were observed to outperform LR.

In addition to the usage of statistical and ML techniques for change-proneness prediction, few hybridized techniques (HBT) and search-based techniques (SBT) have also been employed and their performance has been compared with certain selected ML techniques. Malhotra and Khanna (2018) adopted a Particle Swarm Optimization (PSO)-based classifier for the prediction of change at a class level. PSO is different from the other SBTs and does not involve of the selection of individuals for evolution. Instead all members of the population endure till the completion of computation. These collaborations among the individuals cause a reiterative up gradation in the solution's quality which leads to greater performances than those of basic SBTs. Bansal (2017) evaluated various SBTs and compared their predictive capability to four ML techniques (ADB, NB, LB, BN) using Accuracy and G-Mean. Although the SBTs outperformed the ML techniques, statistical tests indicated that that there does not exist any statistically significant dissimilarity in the results of the best performing SBT and the selected ML models. Malhotra and Khanna (2017) examined the performance of five HBTs, five SBTs and four ML algorithms with the help of six application packages of open-source Android data set. The software change was assessed at a class level. The ML techniques C4.5, CART, MLP, LB and SVM were observed to obtain comparative results to the selected HBTs and SBTs with respect to performance measures like Average balance and Average G-Mean. Kaur and Mishra (2018) constructed datasets from four sequential releases of the JFreeChart software and judged the proficiency of six HBTs/Evolutionary models apropos to change-proneness prediction with the performance indicator as Accuracy. The results indicated that the HBTs obtain similar performance to classifiers like LDA and LR and in some cases, even outperform them.

It has been observed from the literature read that most of the articles only consist of a ten-fold validation of the selected prediction techniques. Only two studies (Elish and Al-Zouri, 2014; Malhotra and Bansal, 2015) analyse the effectiveness of the ML techniques using inter-project evaluation and only one study employs an inter-release validation. Additionally, there have been certain articles (Bansal, 2017; Kaur and Mishra, 2018; Malhotra and Khanna, 2017; Malhotra and Khanna, 2018) that state that the ML techniques underperform in comparison to evolutionary and search-based techniques. However, these articles have not included RF technique in their comparative analysis, which, has been touted as the most efficient classifier to predict change-proneness of software among most of the available ML techniques. There has also been some application of statistical tests with authors in (Malhotra and Jangra, 2013) employing the t-test and most of the other studies (Giger et al., 2012; Malhotra and Khanna, 2017; Malhotra and Khanna, 2018; Romano and Pinzger, 2011) employing the Wilcoxon test for establishing the disparities between the performances of several techniques. However, the t-test relies on many presumptions like the normal distribution of data and it is not advisable to use Wilcoxon's test sans Bonferroni correction, as family-wise error is not considered.

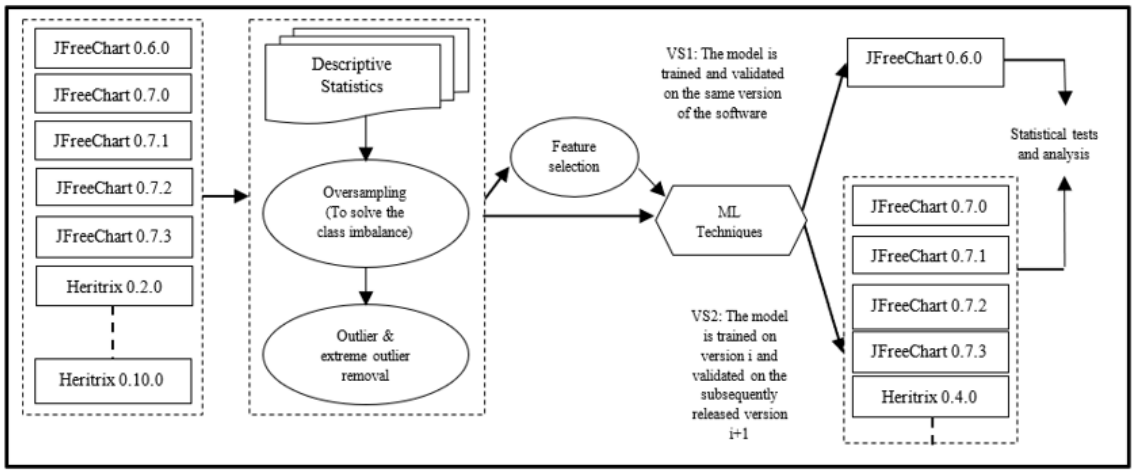
The investigation performed in this research article is disparate from existing works on software change-proneness prediction as it examines, calculates and compares ML methods for creating version to version change-prediction models on Java files over two validation scenarios (10-fold intra release and an inter-release validation). Most of the ML

techniques analyzed in this work have not been assessed before with respect to change-proneness prediction. Besides, an inter-release validation aids in estimating the trend of change-proneness. Additionally, a non‐parametric statistical test: the Kruskal Wallis test is performed in our study which identifies if the ML techniques perform differently under various validation scenarios and whether this disparity is statistically significant or not. To gather the final conclusions, we employ the Scott-Knott cluster analysis to compare the performances all ML techniques over a specific validation scenario according to their AUC values and then cluster them into homogenous subgroups, wherein each subgroup contains the techniques that are significantly indifferent. This is followed by Ranked Voting (RV) method with Borda counting to rank the ML techniques in the best performing sub-group across multiple performance measures. The RV method has never been employed in the existing literature for ranking software change prediction models, and is therefore considered to be novelty in this research article. Therefore, a thoroughly pervasive framework with repeatable results has been provided in this study which is capable of assisting the software developers in an expert selection of a prediction approach amid a massive choice of prevailing methods.

## Ⅲ. Pragmatic Framework for Change-proneness Prediction

A pragmatic framework for change-proneness prediction is presented in this section for the purpose of performing an extensive assessment and comparison of the selected ML techniques. As seen in <Figure 1>, various successively released versions of the JFreeChart and Heritrix plugin projects are selected and their corresponding descriptive statistics are gathered. Real world datasets commonly show the particularity to have anomalies along with a number of samples of a given class under-represented compared to other classes (Khoshgoftaar et al., 2010). Therefore, the selected datasets are pre-processed to solve the problem of class-imbalance, post which the outliers of the datasets are identified and removed. The datasets are also individually scrutinized for finding the best subset of variables for prediction via an appropriate feature selection technique. After this, all the selected ML techniques are exclusively applied on the each of the selected version datasets and the models generated are subjected to two validation settings (VS1 and VS2). In VS1, every prediction model generated apropos to version to version change-proneness is trained and tested using the same version dataset. In VS2, the model is trained using version $V_i$ and validated using the successively released version $V_{i+1}$. The procedure is repeated over all the chosen releases of the two projects. The results of change prediction models generated are assessed via various performance metrics. We further employ Kruskal-Wallis test and Scott-Knott cluster analysis with Ranked Voting for statistically scrutinizing the results of the study.

It is imperative to note here that we do not directly proceed with an inter-release validation (VS2) before analysing the selected ML techniques via a 'k-fold' validation (VS1). This is due to the fact that a prediction model that exhibits a very poor performance during VS1 i.e., when it has been trained and tested using the same version dataset, will not exhibit high performance when validated via VS2 i.e., trained using version $V_i$ and validated using the successively released version $V_{i+1}$. Such models are therefore identified via VS1 and not included for analysis




<Figure 1> Framework for Predicting the Change-proneness of Java Files Using Two Validation Settings

with VS2.

The following sub-sections consist of the description of the variables employed, as well as the target projects selected for analysis. The procedure employed to gather the data corresponding to the variables employed is also provided.

### 3.1. Variables in the Study

The independent variables symbolize the causes or inputs, and are also called as features or predictor variables. OO metrics along the lines of size, coupling, cohesion, complexity etc. have often been observed to have an impact on software change, particularly software maintenance. While not a quality attribute per se, the sizing of source code is a software characteristic that obviously impacts maintainability. It is a common general belief that large modules are more difficult to understand and modify than small ones, and maintenance costs will be expected to increase with average module size. Also, if the modules are too large they are unlikely to be devoted to single purpose. Complexity, on similar lines, refers to characteristics of software which make it difficult to understand and work with thereby hindering software maintainability. Cohesion metrics estimate the degree of relatedness among class members while coupling simply connotes the interconnectedness of modules within a software or a program. Source code components should be loosely coupled (fanout, intermediaries) to avoid propagation of modifications and highly cohesive so that they are easier to maintain and are less frequently in need of changes. Such components are more usable than others simply because their design follows a well-focused purpose.

This analysis employs a combination of OO metrics for change-proneness prediction which are summarized in <Table 1>.

The objective of our prediction analysis is to classify the change-prone files for a specific release. Therefore the binary statistic of 'Change' is the dependent variable employed in this analysis that indicates whethere or not a file has been used with change in the subsequent version.

<Table 1> Software Metrics[5] Selected

| Software Metric and Description |
|---|
| *Source lines of code (SLOC)*: Sum of lines in the code file that have source code. However, a line can have source code as well as a comment and therefore amounts to several metrics. |
| *Total Cyclomatic Complexity (CycloC)* (McCabe, 1976): CycloC computes the number of linearly independent paths within a particular piece of code. The tool calculates CycloC by adding 1 to the total number of keywords for decision points. |
| *Cumulative Halstead Length (CHL)* (Halstead, 1979): The total of number of operators(OP) and operands(OD) in a code is called as the Cumulative Halstead Length. |
| *Cumulative Halstead Volume (CHV)* (Halstead, 1979): CHV indicates the complete information that a person reading a source code needs to comprehend so as to understand its meaning. CHV is calculated as = CHL * log2(V); where V is Halstead Vocabulary and is the total of the number of distinct operators(UOP) and the number of distinct operands(UOD). |
| *Cumulative Halstead Effort (CHE)* (Halstead, 1979): CHE of a source code denotes the total intellectual effort required for its refabricating and is given as = CHV* DIF; where DIF is Halstead Difficulty and is calculated as (UOP/2) * (OD/UOD). |
| *Cumulative Halstead Bugs (CHB)* (Halstead, 1979): CHB calculates the number of bugs that could be present in a given source code and is calculated as CHV/3000. |
| *Maintainability Index (MI)* (Oman and Hagemeister, 1992): MI indicates the ease of maintaining a particular source code file and is given as – MI = 171 - 3.42ln(aE) - 0.23aV(g') - 16.2ln(aLOC);<br>Where aE indicates the average Halstead Effort of the code file, aV(g') indicates the average extended cyclomatic complexity of the code file and aLOC is the average numbers of lines of code per module in that file. |
| *Afferent Coupling (AC)* (Martin, 2003): The AC metric identifies the number of interfaces and classes from other files that depend on the classes in a given file. |
| *Efferent Coupling (EC)* (Martin, 2003): EC is determined as the number of types inside a file's class which depend on other classes' types. |
| *Instability* (Martin, 2003): Instability is estimated by calculating the effort needed to alter a file sans the alteration of other files in the software. It is computed as: Instability = EC / (AC + EC). |
| *Weighted Methods per Class (WMC)* (Chidamber and Kemerer, 1994): WMC measures the sum of the complexities of all class methods. It gauges the amount of effort needed for the development and maintenance of a specific class. |
| *Depth of inheritance (DIT)* (Chidamber and Kemerer, 1994): The depth of a class in the inheritance hierarchy is evaluated as the highest count of nodes originating from the class to the root node. |
| *Number of children (NOC)* (Chidamber and Kemerer, 1994): Number of Children (NOC) is the total count of direct subclasses of a given class. |
| *Coupling between objects (CBO)* (Chidamber and Kemerer, 1994): CBO relates to the notion that if the declared methods of one class use instance variables or methods defined by the other class, then the two classes are termed to be coupled to each other. It is evaluated according to the formula: CBO = AC + EC |
| *Response for a class (RFC)* (Chidamber and Kemerer, 1994): This metric is the sum of all the methods present in a class and all the other methods which get called by methods of this class. Since this is a set, every called method is evaluated just once irrespective of the number of times it is called. |
| *Lack of cohesion (LCOM)* (Chidamber and Kemerer, 1994): This metric calculates what percentage of methods contained in a class use a given instance variable belonging to the class. A low percentage indicates a high cohesion between class methods and data. |
| *Cognitive Complexity (CogC)* (Kaur and Mishra, 2019): CogC evaluates the software developer's amount of complication in understanding a source code component. It is appraised via calculating the cognitive weights of the Basic Control Structures(BCS) contained in the source code of the software and is given as:<br>$$CogC = \sum_{k=1}^{x}[\prod_{j=1}^{m}\sum_{i=1}^{n} W_c(k,j,i)]$$<br>where CogC is the total of cognitive weights of x linear chunks of the software lines present in specific BCSs where each chunk may well be comprising of m levels of nesting BCS's, wherein every level includes n linear BCSs. |

### 3.2. Description of the Target Projects

The usage of source code sans alteration from version to version and within the same product category has been obtained in the JFreeChart and the Heritrix software family. JFreeChart[2] allows the construction of an extensive range of non-interactive as well as interactive charts. On the other hand, Heritrix[3] is a web crawler aimed for tasks like web archiving, built by the Internet Archive. Both of the selected software projects are accessible via a free software license, use SourceForge[4] as the version control system, have been developed in Java and are available as plug-ins. Additionally, the open source quality of the chosen software projects increases the replicability of the analysis. Five successional releases of each of the software projects are examined, the details of which have been given in <Table 2>.

### 3.3. Empirical Data Collection

The Cognitive complexity (CogC) metric has been calculated manually in terms of Cognitive Weight Units for all the Java files of the selected JFreeChart and Heritrix datasets according to the lucid guidelines given in (Kaur and Mishra, 2019).

Two static code analysis tools have been employed to gather the numeric values corresponding to the remaining sixteen independent variables, in regard to every file that exists in all the selected JFreeChart and Heritrix software versions. These tools: JHawk 6.1.3 (http://www.virtualmachinery.com/jhawkprod.htm) and Stan4J (http://stan4j.com) compute the metrics according to their standard definitions. The dependent variable analysed in this research article is represented via a statistic denoting whether a Java file of a software project's release has been employed with or sans any change in the successional release of the software or not. As sans change suggests identical source code copies in our analysis, hence, the "unchanged" or "stable" Java files have been classified via the AntiCutandPaste[6] tool (Kuo et al., 2012). This software analyses two source code packages and returns those code files that have strictly identical content, that is, those Java files that have been utilized without any change from one version to the next.

After applying the AntiCut&Paste software over two successional versions, we calculate the change statistic (a binary value of Yes/No) and enter this with respect to every Java file in the chosen JFreeChart and Heritrix releases. A change statistic of "Yes" suggests that the file of the version under consideration has been altered in its successional release and "No" suggests that the file has been employed in the successional release minus any alteration.

An outline on the change-proneness of Java files existent in the ten releases considered in our analysis is provided in <Table 3>. We grouped the binary values of the change statistic along with the numeric values of the seventeen independent variables vis-à-vis just those Java files that have been included in the successional version (refer to column 3 of <Table 3>) for each of the ten specific releases, making the total number of data points equal to 1,477.

2) http://www.jfree.org/jfreechart.
3) http://crawler.archive.org/index.html.
4) https://sourceforge.net/
5) Granting most of the metrics included in our analysis are class-level measures, our study is performed at a file-level. When files are found to be constituted of more than one class, the sum of numeric values of the metrics achieved by its every constituent class is taken into consideration.
6) https://www.anticutandpaste.com/.

<Table 2> Version Specifics of the Selected Software Projects

| Software projects | Total size in LOC | Total number of Java files |
|---|---|---|
| JFreeChart 0.6.0 | 5,700 | 86 |
| JFreeChart 0.7.0 | 7,870 | 105 |
| JFreeChart 0.7.1 | 8,894 | 128 |
| JFreeChart 0.7.2 | 9,222 | 130 |
| JFreeChart 0.7.3 | 9,318 | 131 |
| Heritrix 0.2.0 | 8,331 | 125 |
| Heritrix 0.4.0 | 11,688 | 168 |
| Heritrix 0.6.0 | 12,751 | 200 |
| Heritrix 0.8.0 | 42,351 | 223 |
| Heritrix 0.10.0 | 49,441 | 249 |

<Table 3> Change Statistics of the Java Files in the Selected Software Project Releases

| Versions | Total number of Java files | Number of Java files used in the next version | Number of Java files used without change in the next version | Number of Java files used with change in the next release |
|---|---|---|---|---|
| JFreeChart 0.6.0 | 86 | 86 | 67 | 19 |
| JFreeChart 0.7.0 | 105 | 102 | 42 | 60 |
| JFreeChart 0.7.1 | 128 | 122 | 94 | 28 |
| JFreeChart 0.7.2 | 130 | 130 | 112 | 18 |
| JFreeChart 0.7.3* | 131 | 130 | 92 | 38 |
| Heritrix 0.2.0 | 125 | 113 | 48 | 65 |
| Heritrix 0.4.0 | 168 | 155 | 80 | 85 |
| Heritrix 0.6.0 | 200 | 195 | 128 | 67 |
| Heritrix 0.8.0 | 223 | 213 | 89 | 124 |
| Heritrix 0.10.0* | 249 | 231 | 171 | 60 |

Note: We examined the successional version JFreeChart 0.7.4 for the change statistics of JFreeChart 0.7.3 and the successional version Heritrix 1.0.0 for the change information of Heritrix 0.10.0.

## Ⅳ. Experimental Setup

The following sub-sections elaborate the steps undertaken for the purpose of empirically answering the RQs stipulated in Section 1.

### 4.1. Resampling of Unbalanced Datasets

Datasets collected corresponding to the various software development tasks normally experience the class imbalance problem (Gray et al., 2012) which creates difficulties for the prediction techniques as there is an under-portrayal of one class and an over-depiction of the other.

In this analysis, we utilize SMOTE (synthetic minority oversampling technique) (Chawla et al., 2002) for achieving an identical class distribution in each of the ten datasets. SMOTE operates by creating new "synthetic" examples for the under-represented class instead of duplicating prevailing ones. Since SMOTE

employs a user-defined factor that determines the number of new examples to create i.e., 100% more examples will be created with a value of 100, therefore a different parameter value has been employed for each of the ten datasets as stated in column two of <Table 4>. The parameter values have been decided in such a way that SMOTE creates sufficient synthetic instances that render the total number of minority class occurrences equal to the number to occurrences in the majority class.

### 4.2. Outlier Detection and Removal

With the objective of obtaining equitable results, we efficiently detect and eliminate all the outliers and extreme outliers from each of the ten datasets by means of the Inter Quartile Range (IQR) filter (Tallón-Ballesteros and Riquelme, 2014). We employed a multivariate outlier and extreme value detection data by calculating the limits on each of the 17 selected independent variables in each of the version dataset. Outliers were considered to be those observations with that had distance more than 1.5 times the IQR and those more than 3 times the IQR were considered to be as extreme outliers.

Columns three and four in <Table 4> show the number of outliers and extreme outliers identified and removed in each data set after application of SMOTE and also indicates the final values of the change statistic obtained post these two steps of data pre-processing.

### 4.3. Feature Selection Methods

Research articles in effect indicate that extraneous attributes, in conjunction with redundant attributes, are capable of adversely affecting the accuracy of the predictors (Kumar et al., 2017). This study employs Correlation based Feature Selection (CBFS) (Kaur and Mishra, 2019), to effectually shortlist the best independent variables for change-proneness prediction of files out of the seventeen selected OO metrics. CBFS relies on the hypothesis that an efficient attribute subset holds attributes/features highly correlated with the outcome, but uncorrelated with one another, and chooses an efficient subset of attributes by scrutinizing their individual predictive capability and their redundancy among one another.

<Table 4> Change Statistics After Resampling and Outlier and Extreme Outlier Detection and Removal

| Versions | SMOTE | Inter Quartile Range filter | | Change statistics after data pre-processing | |
|---|---|---|---|---|---|
| | *% of minority class balancing* | *Number of outliers detected* | *Number of extreme outliers detected* | *Number of Java files used without change in the next release* | *Number of Java files used with change in the next release* |
| JFreeChart 0.6.0 | 253 | 17 | 5 | 50 | 66 |
| JFreeChart 0.7.0 | 17 | 19 | 9 | 39 | 51 |
| JFreeChart 0.7.1 | 221 | 24 | 12 | 70 | 88 |
| JFreeChart 0.7.2 | 522 | 22 | 17 | 92 | 103 |
| JFreeChart 0.7.3 | 142 | 27 | 11 | 67 | 89 |
| Heritrix 0.2.0 | 34 | 22 | 32 | 34 | 39 |
| Heritrix 0.4.0 | 20 | 23 | 41 | 58 | 61 |
| Heritrix 0.6.0 | 91 | 39 | 69 | 68 | 103 |
| Heritrix 0.8.0 | 39 | 47 | 61 | 76 | 90 |
| Heritrix 0.10.0 | 185 | 63 | 93 | 90 | 141 |

### 4.4. Prediction Techniques Incorporated

<Table 5> provides a brief report of the 31 ML techniques used for generating models with respect to change-proneness prediction of Java files using the ten datasets (given in <Table 4>). It is crucial to state here that the prediction techniques have been carefully chosen in a way so that each category has a minimum of one technique being analysed. Additionally, ML techniques that have not been explored vis-à-vis change-proneness prediction in the existing literature have also been analysed for their competency.

We performed a brute-force hyper-parameter optimization for each of the selected 31 techniques using the Weka Experiment Environment via an empirical process of trial and error. The final hyper-parameters that yielded the highest AUC values were incorporated for every technique.

<Table 5> Prediction Techniques Incorporated in this Analysis

| Prediction Technique | | Hyperparameter values | Description |
|---|---|---|---|
| *Bayesian Classification* | | | |
| BN | Bayesian Network (Van Koten and Gray, 2006) | Bayes net estimator; Search technique: Simulated Annealing for 100 runs | The Bayesian classifiers are focused on determining the degree to which the probability that a hypothesis is correct depends on former unaware information. The *BN* allows the user to specify which attributes are conditionally independent. *NB*, on the other hand assumes conditional independence among the attributes. Batch Size = 100 was employed for both the Bayesian classification techniques in this article. |
| NB | Naïve Bayes (Van Koten and Gray, 2006) | Kernel estimator without supervised discretization | |
| *Functions* | | | |
| FLDA | Fisher's Linear Discriminant Analysis (Kaur and Mishra, 2018) | Ridge value = 1.0E-6 | *FLDA* performs linear classification by projecting the data to a lower dimension in a manner that the projected means of categories are distant keeping the range of the projected data minor. *LR* is a conservative modelling methodology that assists in the description of the relationship between certain Xs (independent variables) and a twofold categorical dependent variable (Y), demonstrating an event's pragmatic occurrence or non-occurrence. *MLP* uses back-propagation algorithms to train the connection weights so that it takes long time to train, because the back-propagation algorithms rely on greedy search algorithms like gradient decent. *RBFN* differs from MLP, because in RBF networks the hidden layer performs some computation using a Gaussian Radial Basis Function (RBF) which has linear parameters. The *SMO* technique picks the size of the working set to be two and employs a simple method for solving the reduced minor quadratic programming issues which occur while training the Support Vector Machines (SVMs). The *SPegasos* algorithm, on the other hand, is a SVM model taking advantage of the Stochastic Gradient Descent. The SPegasos technique has the straightforwardness and rapidity that online learning techniques possess but is sure to approach to a realistic dichotomous SVM result. Batch Size = 100 was employed for all the Function-based classification techniques in this article. |
| LR | Logistic Regression (Kaur and Mishra, 2019) | Ridge value = 1.0E-6; With conjugate gradient descent | |
| MLP | Multi-Layer Perceptron (Kaur and Mishra, 2019; Malhotra and Khanna, 2013) | Momentum = 0.2; Learning rate = 0.3; Number of layers = (attributes + classes)/2 | |
| RBFN | Radial Basis Function Neural Network (Peng et al., 2011) | Number of clusters = 2; Ridge value = 1.0E-6 | |
| SMO | Sequential Minimal Optimization (Peng et al., 2011) | Complexity parameter = 1; Caliberator = LR; Kernel = PolyKernel | |
| SPEG | SPegasos (Shalev-Shwartz et al., 2011) | Loss Function = Logloss; Epochs = 500 | |

<Table 5> Prediction Techniques Incorporated in this Analysis (Cont.)

| Prediction Technique | | Hyperparameter values | Description |
|---|---|---|---|
| *Meta-classification* | | | |
| ADB | ADB (Kaur and Mishra, 2018) | Classifier: RF with resampling | The *ADB* methodology groups diverse results from learning techniques in an effort to obtain a combined prediction wherein the training case weights are changed in each cycle. This is done to coerce the learning algorithms in giving extra emphasis on instances that were assessed erroneously before and reduced importance to those instances that were predicted correctly. On the other hand, a plurality voting scheme is employed by the *Bagging* technique to group various results from a learning technique with the aim to find a joint single prediction. *Dagging* generates various disjoint, stratified folds from the dataset and every fold is submitted to a copy of a given base classification technique. A majority voting scheme is employed to make the final predictions. *FC* conducts classification on dataset that has been processed via an arbitrary filter that employs some mathematical evaluation. *LB* classifies via a regression technique as the base learner that is principally enhanced to cater to noisy data and handles multi-class problems. In a *MCC*, performance measures are estimated with respect to each class by viewing it as a binary classification problem and after all the residual classes have been merged to be second class entities. Post this step, a weighted average (weighted by class frequency) or a macro average (consider every class to be equal) metric is estimated by taking the average of the dichotomous metric with respect to all classes. The *RSS* classifier resembles the bagging technique but in RSS, random subsets from the given dataset are selected to be random subsets of the attributes as opposed to in bagging wherein the samples are selected with replacement. |
| BAG | Bagging (Ting and Witten, 1997; Kaur and Mishra, 2018) | Classifier: RF | |
| DAG | Dagging (Ting and Witten, 1997) | Classifier: RF with 10 folds | |
| FC | Filtered Classifier (Kaur and Mishra, 2018) | Classifier: J48; with discretize filter | |
| LB | LogitBoost (Bansal, 2017) | Classifier: J48 | |
| MCC | Multi-Class Classifier (Kaur and Mishra, 2018) | Classifier: LR | |
| RSS | Random Sub Space (Kaur and Mishra, 2018) | Classifier: J48 | |
| *Miscellaneous* | | | |
| CHIRP | Composite Hypercubes on Iterated Random Projections (Wilkinson et al., 2011) | Batch size = 100; seed = 1; num Voters = 7 | *CHIRP* is a non-parametric classifier that deals with nonlinear separability, computational complexity, and the curse of dimensionality. For categorized inputs in a high-dimensional setting, CHIRP uses computationally-effective approaches for generating 2D projections and sets of rectangular regions on projections comprising of data from a particular class category. CHIRP systematizes these region and projection sets into a list of decisions for allocating scores to new data points. The *FLR* classifier induces expressive, decision-making rules in a mathematical lattice data domain including space RN. Learning is performed quickly and incrementally via the computation of disjunctions of join-lattice interval conjunctions (a hyperbox in $R^N$). It has been claimed to be a better classifier in comparison to C4.5 decision trees well as back-propagation neural networks. *VFI*, on the other hand enables each feature to participate in the classification. Every feature submits a vote for one of the classes out of the 'n' available classes and the class with the highest votes is declared to be the predicted class. |
| FLR | Fuzzy Lattice Reasoning (Kaburlasos et al., 2007) | Batch size = 100; Vigilance parameter value = 0.5 | |
| VFI | Voting Feature Intervals (Malhotra et al., 2016) | Bias = 0.6; with weight feature by intervals set to TRUE | |

<Table 5> Prediction Techniques Incorporated in this Analysis (Cont.)

| Prediction Technique | | Hyperparameter values | Description |
|---|---|---|---|
| *Decision Rules* | | | |
| DTNB | Decision Table/Naive Bayes hybrid classifier (Malhotra et al., 2016) | Default settings with backward elimination as search technique | The *DTNB* hybrid classifier appraises the importance of segregating the features into two disjoint groups: one for NB and the other one for the decision table. All features are modeled by the decision table to begin with. At every step of a forward selection search, certain features are using the NB and the remaining use the decision table. *FURIA* extends the well-known RIPPER algorithm, while conserving its benefits. In addition, instead of conventional rules and rule lists, FURIA learns via fuzzy rules and consists of unordered rule sets. Furthermore, it utilizes an effective rule stretching approach for dealing with uncovered examples. The *Modlem* algorithm performs induction of decision rules within Cluster Splitting based Resource Allocation and directly handles numerical attributes during rule induction. *MOEFC* constructs a fuzzy rule based classifier by using the ENORA or NSGA-II Multi-objective Evolutionary Algorithm. We employed ENORA in our work as it is configured to maximize accuracy, to maximize area under ROC curve, and to minimize root mean squared error. *NNGE* algorithm uses non-nested generalized exemplars (that can exhibit just one example from the training database or hyperrectangles signifying two or more examples of a particular class from the training database that can be viewed as if-then rules). This new example is then categorized as a class member of the closest exemplar using Euclidean distance. *PART* algorithm uses a divide-and-rule method, constructs a partial C4.5 decision tree during every run and labels the most appropriate leaf node as a rule. |
| FURIA | Fuzzy Unordered Rule Induction Algorithm (Prati, 2015) | Default with rule stretching for uncovered instances | |
| MOD | Modlem (Malhotra et al., 2016) | Conditions measure = Laplace estimator with full matching | |
| MOEFC | Multi Objective Evolutionary Fuzzy Classifier (Jiménez et al., 2019) | Algorithm: ENORA; Generations: 20; Evaluation value: Accuracy | |
| NNGE | Non-Nested Generalised Exemplars | Attempts of gene option: 5 | |
| PART | PARTial decision lists (Malhotra et al., 2016) | Use MDL correction = TRUE, while finding splits on numeric attributes | |
| *Decision Trees* | | | |
| ADT | Alternating Decision Trees | Search path = Expand all paths ; number of boosting iteration = 10 | *ADT*s comprise of interchanging decision nodes that indicate a base condition, and prediction nodes that hold a single number. To classify an instance, ADT follows all the paths having decision nodes as true, adding up all the prediction nodes that are crossed. The *HFT* makes use of a pre-pruning policy based on the Hoeffding bound to incrementally grow a decision tree. The *J48* technique creates a decision tree by iterative data splitting and Depth-first strategy is employed for decision growth. *LMT* is an amalgamation of logistic regression (LR) and decision trees. Information gain employed to split the dataset, the LogitBoost (LB) technique produces a LR model at every tree node, and the CART algorithm is employed to prune the trees. *RF* is an assemblage of unpruned classification or regression trees, produced from bootstrap training dataset samples, by means of random feature selection during the tree generation procedure. The predictions of the |
| HFT | Hoeffding Trees | Leaf prediction strategy: Bayesian classification; | |
| J48 | J48 (Malhotra and Khanna, 2014) | Split criterion: Info-Gain split; Split confidence: 1.0E-7 Confidence factor: 0.25 with no pruning | |
| LMT | Logistic Model Trees | Fast regression with no split on residuals | |

<Table 5> Prediction Techniques Incorporated in this Analysis (Cont.)

| Prediction Technique | | Hyperparameter values | Description |
|---|---|---|---|
| *Decision Trees* | | | |
| RF | Random Forest (Malhotra and Khanna, 2013; Kaur and Mishra, 2019) | Number of iteration = 100; Seed = 1 | ensemble are aggregated via a voting scheme to deliver a conclusive prediction. The *CART* employs Gini Index to form subsections of the dataset via all independent variables and creates two child nodes repetitively. The ultimate objective is to construct dataset subsets that are of maximum homogeneity with the predictor variable. *SYSFOR* is a gain ratio-based multiple tree building algorithm. Multiple trees are built with the purpose of gaining enhanced knowledge via the extraction of multiple patterns which continues up until the user stated numbers of trees are constructed. Batch Size = 100 was employed for all the Tree-based classification techniques in this article. |
| CART | Classification & Regression Trees (Malhotra and Khanna, 2017) | Heuristic and Pruning = TRUE | |
| SYSFOR | SysFor (Islam and Giggins, 2011) | Number of trees = 60; Separation = 0.3; Goodness = 0.3; Confidence = 0.25 | |

## 4.5. Performance Evaluation Metrics

The work performed in this article makes use of binary classification methodologies to predict those files of Java-based software projects that are likely to undergo change in the subsequent release. We evaluated the classification results from the stand-point of performance measures described below:

### 4.5.1. Accuracy

The accuracy of a predictive model is specified to be the percentage of Java files that are correctly predicted to the total number of Java files that exist in the dataset.

### 4.5.2. Area under the ROC Curve (AUC)

The ROC curve (Kaur and Mishra, 2019; Kumari and Kumar, 2019) is acquired by graphing sensitivity vs. 1-specificity and is a resourceful technique for the quality assessment of the generated models. The Area Under the ROC Curve (AUC) is a combination of sensitivity and specificity and signifies the position where both the specificity and sensitivity are maximum. For the purpose of determining conclusive observations in regard to the performance of the models, standard understandings of the AUC values were employed which assert the following: models with AUC values less than 0.6 display an undesirable classification; AUC values higher than or equal to 0.6 and lesser than 0.7 imply poor classification; AUC values higher than or equal to 0.7 and lesser than 0.8 indicate an acceptable classification; AUC values higher than or equal to 0.8 and lesser than 0.9 imply excellent classification; and AUC values higher than or equal to 0.9, indicate an outstanding classification between the unchanged and change-prone files.

### 4.5.3. F-measure

The F-measure (Elish and Elish, 2008), or F-score, is the weighted average, or the harmonic mean of recall and precision. Put another way, the F- score conveys the balance between the precision and the recall.

$$Fmeasure = 2 \times \frac{(Precision \times reacall)}{(Precision + reacall)}$$

### 4.5.4. Geometric Mean 1 (g-mean1) and Geometric Mean 2 (g-mean2)

G-mean1 (Espíndola and Ebecken, 2005; Malhotra

and Khanna, 2017) is a single-measure statistic wherein the resultant value between 0 and 100 signifies poor to perfect prediction performance of the model. It is calculated as:

$$\mathrm{G-mean1}=\sqrt{Sensitivity\times Precision}$$

G-mean2 (Espíndola and Ebecken, 2005) on the other hand is calculated as:

$$\mathrm{G-mean2}=\sqrt{Sensitivity\times Precision}$$

#### 4.5.5. Matthews Correlation Coefficient

The Matthews correlation coefficient (MCC) (Shepperd et al., 2014), also referred to as mean square contingency coefficient or the phi coefficient ($\phi$), is primarily suitable for performance evaluation when there exists a wide mismatch between the size of the two classes of a dataset. The calculated value of MCC lies between -1 and 1 where: a value of -1 indicates a perfect inverse match performance, 0 indicates a random match performance, and 1 indicates a perfect match performance. It is calculated as:

$$\mathrm{MCC}=\frac{TP\times TN-FP\times FN}{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}$$

Let's consider a case where there exists an extreme imbalance in the dataset wherein the numbers of cases for either the positive or negative class are too low. In such a situation the classifier might evaluate the value of TP or TN to be 0. Upon averaging the True Negative Rate and True Positive Rate, a score without any direction will be returned. On the contrary, since MCC involves values of all the four quadrants of a confusion matrix, it is therefore a balanced measure and returns a value with a direction (+ve and -ve).

The performance measures MCC and AUC are robust to data imbalance.

### 4.6. Validation Methodologies

Two validation approaches: k-fold cross-validation and inter-version or inter-release validation are employed in our study in order to acquire an additional pragmatic and conclusive estimation regarding the efficacy of the selected ML models with respect to version to version change-proneness prediction of Java files. In a k-fold cross validation (VS1 in <Figure 1>), the dataset is arbitrarily segregated into approximately 'k' equal subsets. For every evaluation, the test set is created from either of the k subsets and the training set is created using the rest of the 'k-1' subsets. This procedure is performed for all the 'k' subsets. In our work, the model generated results are validated with the value of 'k' equal to 10.

On the contrary, in an inter-release validation (VS2 in <Figure 1>), the model training is performed via a certain release and this trained model is validated on its subsequently released version. For instance, as per our context of application, JFreeChart version 0.6.0 is employed to train a model by means of a ML technique and then the developed model is further validated on JFreeChart version 0.7.0. This is repeated for all the datasets. This validation methodology is primarily useful to predict the change-prone files of a specific release of a software whose successive version is yet to be released. A predictive analysis like so would aid in providing insights to the software developers apropos to the trend of change-proneness of files over upcoming releases of a software project.

### 4.7. Statistical Evaluations

We employ the following statistical tests (Demšar, 2006) to evaluate the findings of this study and to provide statistical substantiation to the answers corresponding to the RQs.

#### 4.7.1. Kruskal-Wallis Test

The Kruskal-Wallis test (Bethea, 2018) is used in RQ3 to test if the selected ML techniques perform with a significant difference when they are subjected to different validation settings over the various JFreeChart and Heritrix releases using the AUC and MCC performance measure values. The Kruskal-Wallis test relies on the rank-ordering of data and allows you to test whether the mean ranks are the same in all the groups of three or more independent samples. If each group consists of 4 or more observations, the Kruskal-Wallis test approaches a chi-square distribution. The null hypothesis indicating that at least one of the samples belong to a dissimilar population is accepted if the estimated value of the Kruskal-Wallis test is less in comparison to the chi-square's critical value. If the estimated value of Kruskal-Wallis test is higher in comparison to the critical chi-square value, then the alternative hypothesis is agreed upon and the null hypothesis is overruled.

#### 4.7.2. Scott-Knott Cluster Analysis

We employ the Scott-Knott cluster analysis (Jelihovschi et al., 2014) in RQ4 at 95% confidence level to conduct the statistical comparison of all ML techniques over a specific project according to their AUC values and then segregate them into homogenous subdivisions, wherein every subdivision consists of those techniques that are significantly similar. The Scott-Knott analysis has been previously employed in various defect prediction studies (Ghotra et al., 2015). The analysis proceeds by recursively ranking the selected techniques through hierarchical clustering analysis. In every recursive iteration, the Scott-Knott test segregates the classifiers into two groups on the basis of a specific performance measure's value. If the two groups contain a statistically significant dissimilarity in their performance measure's value, the Scott-Knott test executes yet again within each group. The Scott-Knott test terminates only when no statistically different groups can be created.

Since the Scott-Knott assumes provisionally that the performance measure values should be normally distributed, we made sure that the AUC values obtained with respect to each technique over each of the software projects follow normal distribution. Data is said to follow a normal distribution if the test yields significant values. However, the power of tests like Shapiro-Wilk (S-W) and Kolmogorov-Smirnov test (K-S) for testing the normality of data is low for small sample size.

Since the sample size of our data to perform a Scott-Knott cluster analysis is low, we therefore opt for another rapid way to check if a distribution is normal. We compare the mean and median of AUC values obtained per ML technique per project. In a normal distribution mean is equal to median and thusly the ratio mean/median should be 1. We took a confidence interval of 95% and therefore ascertained a normal distribution of data if the mean to median ratio for every ML technique lied between 0.95-1.05.

#### 4.7.3. Ranked Voting (RV) Using Borda Counting

As elucidated Section 4.7.3, the Scott-Knott analy-

sis allows us to identify the ML techniques having the highest AUC for each of the validation scenarios for both the software project versions. However, when it comes to identifying the best techniques with respect to software change proneness prediction, we should assess the ML techniques using not only AUC, but also taking other performance measures into concern.

Therefore, Ranked Voting (RV) method is employed in this article post the application of the Scott Knott's test in RQ 4 to aggregate ranks across multiple performance measures. RV is a metric of specific preferences and interests as a cumulative towards a joint decision (Bauer and Kohavi, 1999; Klamler, 2005). For example, suppose we have 5 ML techniques (ML1, ML2, ML3, ML4, ML5) and 4 performance measure (pm1, pm2, pm3, pm4). Each of the performance measures rank the techniques in a definitive order given below:

pm1: ML2 > ML1 > ML4 > ML3 > ML5;
pm2: ML1 > ML4 > ML2 > ML3 > ML5;
pm3: ML2 > ML4 > ML1 > ML5 > ML3; and
pm4: ML1 > ML4 > ML5 > ML2 > ML3.

We create a majority margins matrix (MM) as illustrated in <Table 6> in order to calculate the collective decision via a Borda counting scheme (Klamler, 2005; Koch and Mitlöhner, 2009). Every entry in MM corresponds to the number of times a ML technique 'x' precedes another ML technique 'y' across all performance measures. This value is estimated by calculating the difference between the number of times that x beats y (|x > y|) from the times that y beats x (|y > x|). For instance, the first row and third column tell us that $MM_{ML1,ML3}$ = |ML1 > ML3| -|ML3 > ML1| = 4-0 = 4, which indicates that the machine learning technique ML1 beats the technique ML3 by a margin of four.

Post summing of such values for every ML technique over each performance measure, the ML techniques are then ranked in a descending order on the basis of this final summation wherein the ML technique having the largest score is adjudged to be the best performing technique with respect to all the performance measures considered.

From the last column in <Table 6>, we get the following ranking: ML1 > (ML2 ~ ML4) > ML5 > ML3 where ~ symbol means indifference between two techniques. This implies that out of the five ML techniques, ML1 the best performing technique according to the selected performance measures.

## Ⅴ. Empirical Results and Analysis

The following sub-sections exhibit the empirical results gathered with respect to the prediction techniques as responses to the various RQs specified in Section 1. <Table 7> indicates the relevant OO metrics

<Table 6> Majority Margin Matrix where Rows and Columns Headers Represent the ML Techniques

| | ML1 | ML2 | ML3 | ML4 | ML5 | Score |
|---|---|---|---|---|---|---|
| ML1 | | 0 | 4 | 2 | 4 | **10** |
| ML2 | 0 | | 4 | 0 | 2 | **6** |
| ML3 | -4 | -4 | | -4 | 0 | **-12** |
| ML4 | -2 | 0 | 4 | | 4 | **6** |
| ML5 | -4 | -2 | 0 | -4 | | **-10** |

<Table 7> CBFS Results

| Datasets | Metrics selected after CBFS |
|---|---|
| JFreeChart 0.6.0 | SLOC, CHV, AC, DIT, NOC, CBO, RFC |
| JFreeChart 0.7.0 | EC, Instability, CBO, CogC |
| JFreeChart 0.7.1 | CHL, CHV, MI, AC, Instability, CBO, RFC, LCOM |
| JFreeChart 0.7.2 | CHL, CHV, CHB, MI, EC, Instability, DIT, NOC, CBO, LCOM, CogC |
| JFreeChart 0.7.3 | EC, DIT, CBO, RFC, CogC |
| Heritrix 0.2.0 | SLOC, CHL, CHB, EC, WMC, CBO, CogC |
| Heritrix 0.4.0 | CHV, EC, DIT, CBO, RFC |
| Heritrix 0.6.0 | CHE, EC, WMC, DIT, CBO, CogC |
| Heritrix 0.8.0 | SLOC, CHE, EC, CBO, RFC, CogC |
| Heritrix 0.10.0 | SLOC, CHE, EC, WMC, CBO |

that were selected after applying CBFS with the best first search methodology over all the selected JFreeChart and Heritrix versions. As observed from <Table 7>, the CBO metric is found to be selected in all the ten version datasets of JFreeChart and Heritrix software, signifying it to be the most important feature among the seventeen selected features with respect to change-proneness of Java files. Apart from the CBO metric, the EC metric is also observed to be selected in all the five releases of the Heritrix software. Additionally, CogC, RFC, DIT and SLOC metrics are commonly found to be relevant to change-proneness of files in most of the selected releases of both the selected target projects. These results are consistent with the results of feature selection performed in pervious analyses (Kaur and Mishra, 2019; Malhotra and Khanna, 2013; Malhotra and Khanna, 2017) with respect to change-proneness prediction.

### 5.1. RQ1:

*What is the predictive capability of the various ML techniques, by and large, with respect to predicting version to version change proneness of files on the various releases of the two Java-based software projects when 'k'-fold cross-validation with feature selection is employed?*

After the application of the CBFS method, the results acquired from the prediction models developed by means of the 31 ML methodologies on the five JFreeChart and five Heritrix datasets have been stated in <Table 8> and <Table 9>. A 10-fold cross-validation technique has been utilized to assess the results, after the application of CBFS (VS1 as explained in <Figure 1>). The columns in <Table 8> and <Table 9> specify the numerical values of the performance measures attained by every selected prediction technique on each of the selected version datasets (Highest performance measure value for every ML technique apropos to each version has been showed in bold).

As studied from <Table 8> and <Table 9>, the RF technique shows highest values with respect to all the six performance measures out of the 31 ML techniques, consistent for two out of five JFreeChart datasets and three out of five Heritrix datasets. Apart from this, even though the ML technique of MOEFC obtains acceptable values of AUC for six out of ten (five JFreeChart and one Heritrix) version datasets,

very high RMSE values (> 0.7) (though not included in the article) are observed for the same for all the ten datasets. Overall, among the 31 selected ML techniques, the results of CBFS + 10-fold validation indicate that there is no one technique that consistently proves to be the best or worst for change-proneness prediction of Java files for all the selected versions of JFreeChart and Heritrix software projects with respect to any of the performance measures used. Although techniques like SMO, SPEG, FLR, VFI and MOD exhibit acceptable AUC values over the JFreeChart versions, they underperform for most of the Heritrix releases. However, when analyzed in terms of average AUC values for both the software projects, the RF technique shows the best performance overall followed by the LB and SYSFOR techniques. On the other hand, FLR being the only technique that obtains poor cumulative average AUC values among the 31 selected ML algorithms, performs the worst.

### 5.2. RQ2:

*What is the performance of the models with respect to predicting the trend of version to version change proneness of files?*

The efficacy of selected ML techniques is also assessed via an inter-release validation (refer to VS2 in <Figure 1> and Section 4.6), wherein, a model generated via a particular version is validated on its successional release. This procedure is implemented on all the selected versions data sets for JFreeChart and Heritrix as given in <Table 10> and <Table 11>. Having evaluated the performance of the 31 ML techniques using CBFS + 0-fold validation in RQ1 Section 5.1, it is vital to note here that we eliminated those ML techniques which were deemed unsuitable for change-proneness prediction of files over the selected releases of the two projects and therefore we were left with twenty five ML techniques. These techniques[7] (SMO, SPEG, FLR, CHIRP, VFI, MOEFC) have been excluded from further analysis owing to their poor predictive ability in terms of AUC as per the results in <Table 8> and <Table 9>. Also, since the features in the training dataset should match to the features of the testing dataset, therefore only 10-fold validation (sans the CBFS) is employed to train the models and no feature is eliminated from the training or the testing datasets.

<Table 10> and <Table 11> again show that none of the twenty five shortlisted ML techniques exhibit a consistently highest predictive performance over the selected version datasets so as to be ascertained as the best performing ML technique for predicting the trend of change-proneness of files. Particularly, 24 out of the 25 shortlisted ML techniques exhibit their worst performance on Heritrix 0.10.0, with only RF exhibiting acceptable AUC values. This could be due to the fact that the ML techniques, in general, underperform for Heritrix 0.8.0 even during the CBFS + 10 fold validation with 17 out of 25 techniques exhibiting unacceptable AUC values as seen in <Table 9>. Since in <Table 11>, the ML techniques are trained on Heritrix 0.8.0 using 10 fold-validation and tested on Heritrix 0.10.0 during inter-release validation,

7) Even though acceptable prediction (in terms of cumulative average AUC) is exhibited by these techniques for the five releases of the JFreeChart datasets, this is not true for the Heritrix software project with these techniques underperforming for majority of the Heritrix releases. Since we were looking for apposite ML techniques for the prediction of version to version change-proneness of files for Java projects in general, therefore, only those techniques that majorly exhibited acceptable performances for both the selected Java- based software projects are examined further for additional empirical analysis in this research work.

therefore, poor results in terms of predictive performance are observed. This is because majority of the ML techniques perform below the acceptable range of AUC for Heritrix 0.8.0 even during ten-fold validation indicating poor efficiency on the part of the ML techniques for predicting change-proneness of files for that version. Such poorly trained models when tested on a fresh set of data such as Heritrix 0.10.0 therefore severely underperform.

Overall, the results obtained during inter-release validation for trend estimation (VS2) are worse than those achieved by the intra-release analysis (VS1), with the shortlisted ML techniques, in general, performing better for the JFreeChart datasets in comparison to the Heritrix datasets. Decision tree technique of HFT performs the best as far as average AUC values for both the target projects are concerned, followed by RF and SYSFOR techniques. Bayesian classifiers also exhibit acceptable AUC values. However, all the techniques selected under the Decision rule category perform the worst, barring the DTNB approach. Overall, the RF technique which performed the best during the CBFS + 10-fold validation is also observed to perform well during the inter-release validation for change-proneness trend estimation.

### 5.3. RQ3:

*Is the predictive performance of the change-proneness prediction models developed via k-fold cross-validation statistically similar to or different from the performance of the models constructed by means of an inter-release validation?*

We validate if there is any statistical difference amongst the results obtained using 10-fold, CBFS + 10-fold validation (VS1) and inter-release validation (VS2) on all the data sets using Kruskal-Wallis test (Bethea 2018), the results of which is presented in <Table 12>. We used PASW Statistics 18 (SPSS, Chicago, IL, USA) to perform the Kruskal-Wallis test. The AUC values obtained during the three validation procedures by the ML techniques over the various versions of each of the two software projects have been analysed in the test.

It can be seen from <Table 12> that there exists a vast difference among the mean ranks obtained by the ML techniques during the three validation scenarios. The ML techniques exhibit the lowest performance with respect to the AUC values during the Inter-release validation (VS2) even though the models are trained using the 10-fold validation wherein the models are seen to perform well. This is a common phenomenon in applied machine learning and is called as the "Model performance mismatch" problem wherein model skills on the training dataset do not match the skills of the model on the test dataset. This occurs because some small over-fitting of the training dataset is inevitable given hyperparameter tuning thus rendering the training scores optimistic. Therefore, even though the models trained using 10-fold validation indicate acceptable to outstanding AUC values, there could exist a major disparity in performance when they are tested on a new sample during the inter-release validation (VS2).

Additionally as observed in <Table 12>, the test does yield significant results for both the performance measures, due to which we accept the null hypothesis that states that that the performance of the developed models using the three validation techniques is not comparable to each other.

<Table 8> Prediction Results of the 31 ML Techniques over the five versions of JFreeChart Using CBFS + 10 Fold Validation (VS1)

| Tech. | JFreeChart 0.6.0 | | | | | | JFreeChart 0.7.0 | | | | | | JFreeChart 0.7.1 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC |
| BN | 84.3 | 0.942 | 81.1 | 81.1 | 83.8 | 0.649 | 81.1 | 0.856 | 72.3 | 72.8 | 76.5 | 0.536 | 78.0 | 0.829 | 75.3 | 75.4 | 77.9 | 0.528 |
| NB | 86.2 | 0.931 | 83.7 | 83.7 | 86.0 | 0.698 | 78.7 | 0.835 | 73.3 | 73.8 | 77.2 | 0.560 | 75.1 | 0.827 | 67.1 | 67.7 | 71.9 | 0.455 |
| FLDA | 84.6 | 0.921 | 81.7 | 81.7 | 84.2 | 0.663 | 74.4 | 0.850 | 69.5 | 69.5 | 73.5 | 0.446 | 72.0 | 0.789 | 68.3 | 68.3 | 72.2 | 0.430 |
| LR | 87.3 | 0.912 | 85.2 | 85.2 | 87.2 | 0.719 | 75.4 | 0.826 | 70.5 | 70.6 | 74.8 | 0.489 | 74.5 | 0.818 | 68.2 | 69.9 | 74.1 | 0.454 |
| MLP | 84.7 | 0.920 | 81.8 | 81.8 | 84.3 | 0.664 | 84.2 | 0.787 | 76.8 | 78.7 | 79.5 | 0.675 | 77.7 | 0.836 | 73.5 | 73.6 | 76.8 | 0.535 |
| RBFN | 83.9 | 0.906 | 80.8 | 80.8 | 83.4 | 0.646 | 81.1 | 0.829 | 71.7 | 72.9 | 75.9 | 0.569 | 76.6 | 0.787 | 71.0 | 71.2 | 75.0 | 0.497 |
| SMO | 85.0 | 0.841 | 81.3 | 81.3 | 83.9 | 0.664 | 77.4 | 0.763 | 70.0 | 70.1 | 74.0 | 0.465 | 68.7 | 0.660 | 51.2 | 53.7 | 59.5 | 0.310 |
| SPEG | 84.3 | 0.838 | 81.1 | 81.1 | 83.8 | 0.649 | 76.4 | 0.749 | 71.2 | 71.3 | 75.2 | 0.489 | 75.3 | 0.737 | 67.8 | 69.0 | 72.5 | 0.504 |
| ADB | 85.8 | 0.907 | 83.3 | 83.3 | 85.6 | 0.684 | 78.6 | 0.835 | 73.0 | 73.2 | 76.9 | 0.534 | 78.2 | 0.792 | 73.4 | 73.4 | 76.3 | 0.509 |
| BAG | 88.1 | 0.906 | 86.8 | 86.9 | 88.5 | 0.743 | 82.9 | 0.851 | 80.7 | 81.1 | 83.2 | **0.677** | 76.7 | 0.821 | 75.0 | 75.0 | 78.0 | 0.538 |
| DAG | 83.7 | 0.918 | 83.0 | 83.0 | 85.3 | 0.682 | 77.6 | 0.819 | 76.5 | 77.8 | 79.7 | 0.639 | 77.1 | 0.810 | 73.1 | 73.1 | 76.1 | 0.498 |
| FC | 84.5 | 0.877 | 83.8 | 84.3 | **90.2** | 0.687 | 76.1 | 0.760 | 70.1 | 70.5 | 74.5 | 0.510 | 74.3 | 0.777 | 70.9 | 70.9 | 73.9 | 0.450 |
| LB | 88.7 | **0.951** | 87.1 | 87.2 | 88.9 | 0.744 | 80.1 | 0.891 | 76.9 | 76.9 | 80.1 | 0.585 | 79.3 | 0.846 | 75.0 | 75.0 | 78.0 | 0.538 |
| MCC | 87.5 | 0.914 | 85.4 | 85.4 | 87.4 | 0.720 | 75.2 | 0.824 | 70.3 | 70.4 | 74.6 | 0.487 | 75.0 | 0.824 | 68.7 | 68.9 | 73.0 | 0.457 |
| RSS | 85.4 | 0.907 | 83.6 | 83.7 | 85.7 | 0.687 | 78.9 | 0.836 | 78.9 | 79.1 | 81.9 | 0.632 | 78.8 | 0.857 | 77.3 | 77.4 | 80.1 | 0.600 |
| CHIRP | 81.8 | 0.813 | 78.3 | 78.3 | 81.2 | 0.607 | 75.4 | 0.740 | 66.7 | 67.0 | 71.7 | 0.440 | 74.6 | 0.742 | 73.3 | 73.3 | 76.5 | 0.510 |
| FLR | 68.1 | 0.703 | 70.1 | 71.3 | 68.4 | 0.390 | 63.6 | 0.635 | 60.6 | 60.6 | 64.5 | 0.267 | 64.4 | 0.634 | 61.6 | 57.5 | 62.5 | 0.273 |
| VFI | 82.1 | 0.900 | 79.9 | 80.0 | 82.3 | 0.617 | 75.0 | 0.771 | 69.2 | 70.0 | 73.8 | 0.513 | 61.6 | 0.792 | 69.0 | 71.8 | 57.7 | 0.330 |
| DTNB | 79.3 | 0.891 | 72.0 | 72.0 | 76.5 | 0.527 | **86.8** | 0.828 | 80.7 | 80.7 | 83.2 | 0.634 | 80.0 | 0.820 | 78.0 | 78.0 | 80.1 | 0.571 |
| FURIA | 85.0 | 0.886 | 83.3 | 83.5 | 85.4 | 0.675 | 82.2 | 0.837 | 77.1 | 77.3 | 80.3 | 0.608 | 73.9 | 0.785 | 68.7 | 68.7 | 72.3 | 0.431 |
| MOD | **88.7** | 0.882 | 86.2 | 85.8 | 88.1 | 0.739 | 79.9 | 0.791 | **81.0** | **81.1** | **83.6** | 0.373 | 84.4 | 0.838 | 80.4 | 80.5 | 82.9 | 0.642 |
| MOEFC | 83.8 | 0.883 | 80.2 | 80.2 | 82.8 | 0.643 | 81.6 | 0.725 | 67.0 | 67.9 | 71.9 | 0.590 | 73.6 | 0.798 | 74.4 | 74.6 | 77.8 | 0.468 |
| NNGE | 85.8 | 0.859 | 83.7 | 83.7 | 85.9 | 0.686 | 78.5 | 0.776 | 78.1 | 78.2 | 81.0 | 0.605 | 79.1 | 0.790 | 74.7 | 74.7 | 77.5 | 0.526 |
| PART | 84.6 | 0.876 | 83.9 | 84.4 | 85.3 | 0.709 | 77.2 | 0.785 | 70.9 | 71.2 | 75.1 | 0.509 | 80.1 | 0.761 | 71.6 | 71.7 | 75.0 | 0.484 |
| ADT | 84.3 | 0.915 | 81.9 | 81.9 | 84.3 | 0.652 | 83.9 | 0.861 | 76.9 | 76.9 | 80.0 | 0.581 | 76.7 | 0.810 | 71.1 | 71.3 | 75.1 | 0.497 |
| HFT | 86.3 | 0.929 | 83.7 | 83.7 | 86.0 | 0.698 | 80.0 | 0.837 | 74.6 | 75.2 | 78.2 | 0.587 | 75.4 | 0.830 | 67.4 | 67.9 | 72.1 | 0.457 |
| J48 | 85.2 | 0.851 | 84.5 | 85.1 | 86.0 | 0.693 | 80.6 | 0.770 | 74.1 | 74.7 | 77.7 | 0.584 | 73.8 | 0.749 | 73.7 | 73.8 | 75.9 | 0.492 |
| LMT | 87.2 | 0.909 | 84.3 | 84.3 | 86.5 | 0.716 | 81.0 | 0.794 | 74.4 | 75.1 | 78.0 | 0.586 | 76.9 | 0.833 | 76.8 | 76.8 | 79.2 | 0.564 |
| RF | 88.4 | 0.934 | 86.7 | 86.8 | 88.6 | 0.741 | 80.6 | **0.891** | 76.9 | 76.9 | 80.0 | 0.581 | **86.5** | **0.897** | **80.7** | **80.8** | **83.1** | **0.644** |
| CART | 85.6 | 0.855 | 84.6 | 84.9 | 86.2 | 0.697 | 80.8 | 0.814 | 73.8 | 73.8 | 77.3 | 0.535 | 79.4 | 0.765 | 73.5 | 73.5 | 76.7 | 0.512 |
| SYSFOR | 88.1 | 0.915 | **87.6** | **88.1** | 89.0 | **0.760** | 77.3 | 0.859 | 71.9 | 72.0 | 75.8 | 0.509 | 73.3 | 0.847 | 72.9 | 73.1 | 75.0 | 0.471 |

<Table 8> Prediction Results of the 31 ML Techniques over the five versions of JFreeChart Using CBFS + 10 Fold Validation (VS1) (Cont.)

| Tech. | JFreeChart 0.7.2 | | | | | | JFreeChart 0.7.3 | | | | | | Average values over the JFreeChart versions | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC |
| BN | 92.4 | 0.959 | 92.7 | 93.0 | 92.5 | 0.831 | 86.5 | 0.901 | 81.8 | 82.3 | 84.1 | 0.706 | 84.5 | 0.897 | 80.7 | 80.9 | 83.0 | 0.650 |
| NB | 80.8 | 0.863 | 78.3 | 78.4 | 80.2 | 0.584 | 83.7 | 0.909 | 78.2 | 78.7 | 81.4 | 0.639 | 80.9 | 0.873 | 76.1 | 76.5 | 79.3 | 0.587 |
| FLDA | 85.3 | 0.903 | 84.9 | 84.9 | 85.4 | 0.687 | 85.0 | 0.900 | 78.5 | 79.7 | 81.3 | 0.673 | 80.3 | 0.873 | 76.6 | 76.8 | 79.3 | 0.580 |
| LR | 84.3 | 0.893 | 84.2 | 84.3 | 84.4 | 0.670 | 83.8 | 0.890 | 78.3 | 78.8 | 81.2 | 0.648 | 81.1 | 0.868 | 77.3 | 77.8 | 80.3 | 0.596 |
| MLP | 93.4 | 0.948 | 93.1 | 93.2 | 93.5 | 0.840 | 82.7 | 0.864 | 77.1 | 77.6 | 80.4 | 0.621 | 84.5 | 0.871 | 80.5 | 81.0 | 82.9 | 0.667 |
| RBFN | 83.9 | 0.881 | 83.5 | 83.5 | 84.0 | 0.648 | 82.9 | 0.868 | 77.0 | 77.5 | 80.3 | 0.624 | 81.7 | 0.854 | 76.8 | 77.2 | 79.7 | 0.597 |
| SMO | 83.2 | 0.817 | 82.8 | 82.8 | 83.3 | 0.646 | 85.0 | 0.826 | 78.2 | 79.7 | 81.0 | 0.678 | 79.9 | 0.782 | 72.7 | 73.5 | 76.3 | 0.553 |
| SPEG | 85.1 | 0.836 | 85.7 | 86.1 | 85.1 | 0.693 | 83.6 | 0.816 | 77.2 | 78.0 | 80.4 | 0.642 | 80.9 | 0.795 | 76.6 | 77.1 | 79.4 | 0.595 |
| ADB | 94.3 | 0.983 | 94.8 | 94.8 | 95.1 | 0.877 | 83.6 | 0.912 | 79.6 | 79.7 | 82.5 | 0.636 | 84.1 | 0.886 | 80.8 | 80.9 | 83.3 | 0.648 |
| BAG | 93.3 | 0.974 | 92.4 | 92.7 | 93.5 | 0.840 | 83.3 | 0.934 | 78.6 | 78.8 | 81.6 | 0.634 | 84.9 | 0.897 | 82.7 | 82.9 | 85.0 | 0.686 |
| DAG | 88.1 | 0.923 | 88.9 | 89.4 | 87.9 | 0.782 | 85.2 | 0.912 | 80.8 | 81.1 | 83.5 | 0.674 | 82.3 | 0.876 | 80.5 | 80.9 | 82.5 | 0.655 |
| FC | 92.4 | 0.939 | 92.5 | 92.6 | 92.5 | 0.832 | 81.9 | 0.810 | 78.0 | 78.0 | 81.1 | 0.606 | 81.8 | 0.833 | 79.1 | 79.3 | 82.4 | 0.617 |
| LB | **97.7** | **0.992** | **97.5** | **97.5** | **97.7** | **0.930** | 83.6 | 0.934 | 79.6 | 79.7 | 82.4 | 0.635 | 85.9 | 0.923 | 83.2 | 83.2 | 85.4 | 0.686 |
| MCC | 84.4 | 0.893 | 84.3 | 84.4 | 84.5 | 0.670 | 83.9 | 0.891 | 78.4 | 78.9 | 81.2 | 0.649 | 81.2 | 0.869 | 77.4 | 77.6 | 80.1 | 0.597 |
| RSS | 95.1 | 0.982 | 95.0 | 95.1 | 95.2 | 0.875 | 85.5 | 0.921 | 81.0 | 81.4 | 83.7 | 0.676 | 84.7 | 0.901 | 83.2 | 83.3 | 85.3 | 0.694 |
| CHIRP | 92.3 | 0.901 | 92.5 | 92.7 | 92.5 | 0.827 | 85.6 | 0.844 | 81.2 | 81.5 | 83.9 | 0.678 | 81.9 | 0.808 | 78.4 | 78.6 | 81.1 | 0.612 |
| FLR | 84.8 | 0.825 | 81.8 | 82.2 | 83.6 | 0.682 | 72.8 | 0.736 | 71.3 | 71.6 | 73.3 | 0.445 | 70.7 | 0.707 | 69.1 | 68.7 | 70.5 | 0.411 |
| VFI | 90.2 | 0.953 | 90.4 | 90.8 | 90.4 | 0.793 | 82.8 | 0.858 | 75.0 | 76.6 | 78.3 | 0.635 | 78.3 | 0.855 | 76.7 | 77.8 | 76.5 | 0.578 |
| DTNB | 94.1 | 0.956 | 93.5 | 93.5 | 93.9 | 0.855 | 82.8 | 0.881 | 76.5 | 77.2 | 79.8 | 0.625 | 84.6 | 0.875 | 80.1 | 80.3 | 82.7 | 0.643 |
| FURIA | 94.3 | 0.927 | 94.1 | 94.1 | 94.4 | 0.861 | 82.8 | 0.819 | 77.6 | 77.8 | 80.8 | 0.621 | 83.6 | 0.851 | 80.2 | 80.3 | 82.6 | 0.639 |
| MOD | 95.0 | 0.929 | 94.7 | 94.7 | 95.0 | 0.876 | 86.0 | 0.849 | 82.0 | 82.2 | 84.5 | 0.689 | 86.8 | 0.858 | **84.9** | **84.9** | **86.8** | 0.664 |
| MOEFC | 86.4 | 0.899 | 85.5 | 85.5 | 86.3 | 0.706 | 84.4 | 0.885 | 79.3 | 79.6 | 82.2 | 0.651 | 81.9 | 0.838 | 77.3 | 77.6 | 80.2 | 0.612 |
| NNGE | 86.3 | 0.847 | 85.7 | 85.8 | 86.4 | 0.707 | 82.0 | 0.825 | 80.1 | 80.4 | 82.4 | 0.619 | 82.3 | 0.819 | 80.5 | 80.5 | 82.6 | 0.629 |
| PART | 95.7 | 0.951 | 95.4 | 95.4 | 95.7 | 0.889 | 82.5 | 0.871 | 80.6 | 80.7 | 82.8 | 0.630 | 84.0 | 0.849 | 80.5 | 80.7 | 82.8 | 0.644 |
| ADT | 95.3 | 0.982 | 95.1 | 95.1 | 95.4 | 0.880 | 82.5 | 0.905 | 79.2 | 79.2 | 82.0 | 0.621 | 84.5 | 0.895 | 80.8 | 80.9 | 83.3 | 0.646 |
| HFT | 80.7 | 0.863 | 78.5 | 78.7 | 80.4 | 0.591 | 83.2 | 0.902 | 77.7 | 78.2 | 80.8 | 0.616 | 81.1 | 0.872 | 76.4 | 76.7 | 79.5 | 0.590 |
| J48 | 93.9 | 0.930 | 93.6 | 93.6 | 94.0 | 0.855 | 81.6 | 0.805 | 77.8 | 77.8 | 80.8 | 0.595 | 83.0 | 0.821 | 80.8 | 81.0 | 82.9 | 0.644 |
| LMT | 93.4 | 0.963 | 93.0 | 93.0 | 93.4 | 0.840 | 81.5 | 0.892 | 76.9 | 77.0 | 80.2 | 0.594 | 84.0 | 0.878 | 81.1 | 81.2 | 83.5 | 0.660 |
| RF | 94.6 | 0.990 | 94.4 | 94.5 | 94.7 | 0.869 | **87.4** | **0.949** | **84.4** | **84.5** | **86.7** | **0.715** | **87.5** | **0.932** | 84.6 | 84.7 | 86.6 | **0.710** |
| CART | 93.8 | 0.927 | 93.7 | 93.8 | 94.0 | 0.853 | 82.8 | 0.854 | 78.5 | 78.5 | 81.5 | 0.620 | 84.5 | 0.843 | 80.8 | 80.9 | 83.2 | 0.643 |
| SYSFOR | 93.9 | 0.984 | 93.6 | 93.7 | 94.0 | 0.851 | 85.2 | 0.910 | 81.2 | 81.3 | 83.7 | 0.674 | 83.6 | 0.903 | 81.4 | 81.7 | 83.5 | 0.653 |

<Table 9> Prediction Results of the 31 ML Techniques Over the Five Versions of Heritrix Using CBFS + 10 Fold Validation (VS1)

| Tech. | Heritrix 0.2.0 | | | | | | Heritrix 0.4.0 | | | | | | Heritrix 0.6.0 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC |
| BN | 76.0 | 0.822 | 74.6 | 74.5 | 76.0 | 0.492 | 66.3 | 0.749 | 65.4 | 65.4 | 66.3 | 0.301 | 72.6 | 0.787 | 64.9 | 64.9 | 71.1 | 0.401 |
| NB | 74.3 | 0.824 | 70.5 | 70.6 | 73.4 | 0.459 | 77.0 | 0.820 | 73.8 | 74.1 | 76.2 | 0.516 | 72.6 | 0.718 | 60.0 | 60.9 | 67.3 | 0.407 |
| FLDA | 74.2 | 0.780 | 70.3 | 70.4 | 73.2 | 0.458 | 66.2 | 0.780 | 62.5 | 62.7 | 65.5 | 0.301 | 71.5 | 0.770 | 61.0 | 61.1 | 68.1 | 0.364 |
| LR | 70.0 | 0.813 | 67.2 | 67.2 | 69.7 | 0.374 | 69.5 | 0.779 | 67.4 | 67.4 | 69.2 | 0.368 | 72.3 | 0.758 | 58.8 | 59.5 | 66.4 | 0.379 |
| MLP | 70.4 | 0.747 | 64.0 | 64.5 | 68.6 | 0.375 | 76.9 | 0.813 | 74.3 | 74.5 | 76.3 | 0.512 | 68.1 | 0.730 | 56.3 | 56.5 | 64.1 | 0.296 |
| RBFN | 74.1 | 0.752 | 72.0 | 72.0 | 74.0 | 0.458 | 79.5 | 0.801 | 77.9 | 77.9 | 79.3 | 0.563 | 71.8 | 0.707 | 60.6 | 60.8 | 67.7 | 0.371 |
| SMO | 76.0 | 0.757 | 72.8 | 72.8 | 75.4 | 0.490 | 73.1 | 0.728 | 68.3 | 68.9 | 71.6 | 0.447 | 73.5 | 0.697 | 59.8 | 60.6 | 67.2 | 0.395 |
| SPEG | 73.0 | 0.728 | 70.5 | 70.4 | 72.7 | 0.432 | 63.7 | 0.636 | 61.7 | 61.7 | 63.5 | 0.249 | 72.8 | 0.700 | 61.4 | 61.7 | 68.5 | 0.386 |
| ADB | 77.1 | 0.836 | 75.3 | 75.3 | 77.0 | 0.517 | 69.8 | 0.771 | 68.3 | 68.3 | 69.7 | 0.370 | 76.6 | 0.775 | 66.0 | 66.5 | 72.0 | 0.473 |
| BAG | 74.2 | 0.811 | 72.2 | 72.2 | 74.1 | 0.459 | 74.9 | 0.819 | 72.6 | 72.7 | 74.4 | 0.474 | 79.2 | 0.818 | 68.9 | 69.7 | 74.4 | 0.528 |
| DAG | **83.1** | 0.835 | **81.7** | **81.7** | **83.0** | **0.633** | 73.3 | 0.766 | 69.1 | 69.5 | 72.2 | 0.446 | 74.6 | 0.765 | 63.0 | 63.4 | 69.8 | 0.423 |
| FC | 78.5 | 0.738 | 77.1 | 77.1 | 78.5 | 0.545 | 62.6 | 0.659 | 59.5 | 59.6 | 62.2 | 0.231 | 74.2 | 0.750 | 61.9 | 62.5 | 68.7 | 0.420 |
| LB | 71.9 | 0.827 | 68.6 | 68.6 | 71.4 | 0.404 | 67.1 | 0.755 | 65.2 | 65.3 | 66.9 | 0.317 | 79.4 | 0.802 | 71.7 | 71.9 | 76.7 | 0.540 |
| MCC | 70.1 | 0.814 | 67.3 | 67.3 | 69.8 | 0.375 | 69.8 | 0.782 | 67.7 | 67.7 | 69.4 | 0.370 | 72.2 | 0.757 | 58.8 | 59.5 | 66.3 | 0.379 |
| RSS | 74.7 | 0.781 | 73.4 | 73.4 | 74.7 | 0.465 | 70.6 | 0.792 | 67.8 | 67.9 | 70.1 | 0.389 | 80.2 | 0.845 | 69.3 | 70.5 | 74.3 | 0.556 |
| CHIRP | 77.5 | 0.770 | 72.8 | 72.8 | 75.4 | 0.490 | 73.0 | 0.728 | 68.8 | 69.2 | 71.9 | 0.444 | 71.3 | 0.669 | 54.9 | 56.1 | 63.2 | 0.350 |
| FLR | 67.7 | 0.674 | 64.5 | 64.5 | 67.2 | 0.320 | 68.0 | 0.678 | 65.1 | 65.2 | 67.5 | 0.335 | 64.7 | 0.629 | 54.6 | 54.6 | 62.3 | 0.235 |
| VFI | 71.7 | 0.696 | 60.8 | 63.0 | 66.7 | 0.423 | 72.7 | 0.753 | 61.5 | 65.2 | 67.2 | 0.479 | 54.5 | 0.713 | 63.4 | 67.9 | 50.3 | 0.279 |
| DTNB | 78.9 | 0.785 | 75.9 | 76.0 | 78.3 | 0.546 | 71.9 | 0.738 | 67.4 | 67.8 | 70.7 | 0.412 | 75.3 | 0.763 | 60.7 | 62.1 | 67.8 | 0.437 |
| FURIA | 77.5 | 0.801 | 75.7 | 75.7 | 77.4 | 0.520 | 72.6 | 0.737 | 68.6 | 68.9 | 71.6 | 0.428 | 76.1 | 0.762 | 64.8 | 65.4 | 71.2 | 0.461 |
| MOD | 70.5 | 0.704 | 68.7 | 68.6 | 70.4 | 0.379 | 72.4 | 0.723 | 70.8 | 70.8 | 72.2 | 0.421 | 81.7 | 0.789 | 73.1 | 73.6 | 77.6 | 0.582 |
| MOEFC | 74.1 | 0.798 | 69.1 | 69.5 | 72.7 | 0.458 | 68.3 | 0.679 | 64.8 | 64.9 | 67.7 | 0.337 | 73.1 | 0.687 | 59.5 | 60.3 | 66.9 | 0.393 |
| NNGE | 81.8 | 0.817 | 80.6 | 80.5 | 81.7 | 0.605 | 73.8 | 0.738 | 73.3 | 73.3 | 73.7 | 0.455 | 73.0 | 0.708 | 63.4 | 63.5 | 69.9 | 0.402 |
| PART | 73.1 | 0.712 | 68.5 | 68.7 | 72.0 | 0.432 | 77.0 | 0.831 | 75.9 | 75.9 | 76.9 | 0.510 | 74.7 | 0.737 | 57.8 | 60.3 | 65.2 | 0.437 |
| ADT | 78.6 | 0.828 | 75.6 | 75.7 | 78.0 | 0.544 | 73.2 | 0.823 | 72.0 | 72.0 | 73.2 | 0.439 | 79.8 | 0.807 | 68.9 | 70.0 | 74.2 | 0.543 |
| HFT | 75.8 | 0.824 | 71.6 | 71.9 | 74.7 | 0.489 | 75.7 | 0.814 | 72.7 | 73.0 | 75.1 | 0.494 | 74.4 | 0.718 | 60.6 | 61.7 | 67.8 | 0.421 |
| J48 | 71.9 | 0.661 | 66.4 | 66.7 | 70.4 | 0.404 | 75.6 | 0.838 | 75.2 | 75.2 | 75.6 | 0.490 | 75.2 | 0.747 | 61.3 | 62.6 | 68.3 | 0.435 |
| LMT | 74.1 | 0.846 | 71.2 | 71.2 | 73.6 | 0.457 | 75.0 | 0.830 | 71.6 | 71.9 | 74.2 | 0.478 | 72.9 | 0.764 | 59.3 | 60.2 | 66.7 | 0.392 |
| RF | 77.3 | 0.812 | 74.6 | 74.7 | 76.9 | 0.516 | **80.8** | **0.851** | **79.8** | **79.8** | **80.8** | **0.594** | **84.1** | **0.879** | **76.8** | **77.2** | **80.7** | **0.633** |
| CART | 74.5 | 0.741 | 73.3 | 73.3 | 74.5 | 0.464 | 69.5 | 0.725 | 67.5 | 67.5 | 69.2 | 0.369 | 71.9 | 0.729 | 59.3 | 59.7 | 66.7 | 0.369 |
| SYSFOR | 77.3 | **0.872** | 75.4 | 75.4 | 77.2 | 0.518 | 71.2 | 0.812 | 68.7 | 68.8 | 70.8 | 0.403 | 77.4 | 0.824 | 62.7 | 64.9 | 68.9 | 0.495 |

<Table 9> Prediction Results of the 31 ML Techniques Over the Five Versions of Heritrix Using CBFS + 10 Fold Validation (VS1) (Cont.)

| Tech. | Heritrix 0.8.0 | | | | | | Heritrix 0.10.0 | | | | | | Average values over the Heritrix versions | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC |
| BN | 70.7 | 0.731 | 68.9 | 68.9 | 70.7 | 0.393 | 71.7 | 0.792 | 66.7 | 66.7 | 72.8 | 0.438 | 71.5 | 0.776 | 68.1 | 68.1 | 71.4 | 0.405 |
| NB | 68.9 | 0.709 | 58.3 | 59.8 | 64.7 | 0.352 | 66.5 | 0.701 | 47.1 | 48.3 | 57.4 | 0.239 | 71.9 | 0.754 | 61.9 | 62.7 | 67.8 | 0.394 |
| FLDA | 62.1 | 0.674 | 55.7 | 55.8 | 60.8 | 0.209 | 70.9 | 0.746 | 64.0 | 64.0 | 70.6 | 0.393 | 69.0 | 0.750 | 62.7 | 62.8 | 67.6 | 0.345 |
| LR | 65.3 | 0.676 | 56.4 | 56.9 | 62.5 | 0.271 | 70.3 | 0.748 | 53.8 | 55.0 | 62.7 | 0.335 | 69.5 | 0.755 | 60.7 | 61.2 | 66.1 | 0.345 |
| MLP | 62.3 | 0.635 | 52.4 | 52.9 | 59.1 | 0.206 | 73.0 | 0.744 | 59.9 | 60.6 | 67.7 | 0.391 | 70.2 | 0.734 | 61.4 | 61.8 | 67.2 | 0.356 |
| RBFN | 70.6 | 0.714 | 64.1 | 64.5 | 68.7 | 0.375 | 67.6 | 0.693 | 48.1 | 48.8 | 58.1 | 0.225 | 72.7 | 0.749 | 64.5 | 64.8 | 69.6 | 0.399 |
| SMO | 67.1 | 0.634 | 50.7 | 54.0 | 58.8 | 0.328 | 67.3 | 0.635 | 46.1 | 49.1 | 56.4 | 0.299 | 71.4 | 0.690 | 59.5 | 61.1 | 65.9 | 0.392 |
| SPEG | 63.4 | 0.602 | 55.3 | 56.0 | 61.7 | 0.258 | 70.3 | 0.670 | 55.7 | 56.4 | 64.2 | 0.338 | 68.7 | 0.667 | 60.9 | 61.2 | 66.1 | 0.332 |
| ADB | 72.9 | 0.748 | 70.2 | 70.2 | 72.6 | 0.429 | 76.3 | 0.756 | 62.4 | 62.8 | 69.4 | 0.421 | 74.5 | 0.777 | 68.4 | 68.6 | 72.2 | 0.442 |
| BAG | 70.2 | 0.746 | 65.8 | 65.8 | 69.2 | 0.374 | 78.5 | 0.825 | 67.9 | 68.1 | 73.9 | 0.492 | 75.4 | 0.804 | 69.5 | 69.7 | 73.2 | 0.465 |
| DAG | 69.4 | 0.737 | 63.7 | 63.8 | 68.0 | 0.350 | 71.8 | 0.737 | 53.1 | 53.9 | 62.2 | 0.299 | 74.4 | 0.768 | 66.1 | 66.5 | 71.0 | 0.430 |
| FC | 64.0 | 0.628 | 55.6 | 56.0 | 61.5 | 0.235 | 73.9 | 0.776 | 66.5 | 66.6 | 72.9 | 0.458 | 70.6 | 0.710 | 64.1 | 64.4 | 68.8 | 0.378 |
| LB | **74.4** | **0.765** | **71.6** | **71.6** | **74.1** | **0.462** | 76.7 | 0.799 | 63.7 | 64.0 | 70.6 | 0.432 | 73.9 | 0.790 | 68.2 | 68.3 | 71.9 | 0.431 |
| MCC | 65.7 | 0.676 | 56.8 | 57.3 | 62.9 | 0.273 | 70.2 | 0.746 | 53.7 | 54.9 | 62.6 | 0.335 | 69.6 | 0.755 | 60.8 | 61.3 | 66.2 | 0.346 |
| RSS | 70.1 | 0.759 | 64.2 | 64.4 | 68.5 | 0.372 | 75.8 | 0.820 | 66.5 | 66.9 | 72.7 | 0.480 | 74.3 | 0.799 | 68.2 | 68.6 | 72.1 | 0.452 |
| CHIRP | 70.3 | 0.678 | 64.3 | 64.6 | 68.6 | 0.373 | 73.7 | 0.695 | 59.7 | 60.3 | 67.3 | 0.389 | 73.1 | 0.708 | 64.1 | 64.6 | 69.3 | 0.409 |
| FLR | 55.3 | 0.520 | 35.2 | 37.0 | 46.1 | 0.049 | 59.6 | 0.569 | 46.8 | 46.8 | 55.9 | 0.111 | 63.1 | 0.614 | 53.2 | 53.6 | 59.8 | 0.210 |
| VFI | 54.6 | 0.656 | 65.0 | 67.7 | 46.6 | 0.153 | 50.4 | 0.666 | 60.7 | 65.6 | 44.5 | 0.222 | 60.8 | 0.697 | 62.3 | 65.9 | 55.0 | 0.311 |
| DTNB | 69.9 | 0.758 | 65.6 | 65.7 | 69.1 | 0.363 | 76.6 | 0.813 | 70.5 | 70.6 | 76.1 | 0.522 | 74.5 | 0.771 | 68.0 | 68.4 | 72.4 | 0.456 |
| FURIA | 68.7 | 0.673 | 67.3 | 67.4 | 68.4 | 0.340 | 75.6 | 0.737 | 63.4 | 63.9 | 70.2 | 0.439 | 74.1 | 0.742 | 68.0 | 68.3 | 71.7 | 0.437 |
| MOD | 71.3 | 0.687 | 66.7 | 66.8 | 70.4 | 0.389 | 79.8 | 0.754 | 67.3 | 68.4 | 73.2 | 0.522 | 75.1 | 0.731 | 69.3 | 69.6 | 72.8 | 0.459 |
| MOEFC | 59.4 | 0.623 | 45.6 | 46.7 | 54.2 | 0.139 | 73.5 | 0.443 | 59.2 | 59.7 | 66.8 | 0.379 | 69.7 | 0.646 | 59.6 | 60.2 | 65.6 | 0.341 |
| NNGE | 66.3 | 0.642 | 62.6 | 62.6 | 65.8 | 0.291 | 79.1 | 0.745 | 67.6 | 67.7 | 73.8 | 0.475 | 74.8 | 0.730 | 69.5 | 69.5 | 73.0 | 0.446 |
| PART | 69.3 | 0.690 | 68.1 | 68.1 | 69.4 | 0.361 | 69.6 | 0.773 | 63.1 | 63.2 | 69.2 | 0.353 | 72.8 | 0.749 | 66.7 | 67.2 | 70.5 | 0.419 |
| ADT | 69.0 | 0.701 | 63.9 | 64.0 | 67.9 | 0.349 | 77.8 | 0.808 | 66.5 | 66.6 | 72.7 | 0.456 | 75.7 | 0.793 | 69.4 | 69.7 | 73.2 | 0.466 |
| HFT | 70.1 | 0.711 | 59.9 | 61.2 | 65.9 | 0.379 | 66.6 | 0.690 | 48.2 | 49.3 | 58.3 | 0.251 | 72.5 | 0.752 | 62.6 | 63.4 | 68.4 | 0.407 |
| J48 | 68.8 | 0.683 | 67.7 | 67.8 | 68.9 | 0.351 | 76.4 | 0.764 | 67.5 | 67.7 | 73.6 | 0.461 | 73.6 | 0.738 | 67.6 | 68.0 | 71.4 | 0.428 |
| LMT | 64.0 | 0.660 | 56.8 | 57.0 | 62.1 | 0.246 | 74.4 | 0.768 | 62.1 | 62.3 | 69.3 | 0.396 | 72.1 | 0.774 | 64.2 | 64.5 | 69.2 | 0.394 |
| RF | 69.5 | 0.745 | 64.7 | 64.8 | 68.5 | 0.361 | **82.7** | **0.886** | **71.7** | **71.8** | **77.1** | **0.543** | **78.9** | **0.835** | **73.5** | **73.7** | **76.8** | **0.529** |
| CART | 70.0 | 0.740 | 66.2 | 66.3 | 69.5 | 0.365 | 75.2 | 0.736 | 62.5 | 62.7 | 69.6 | 0.384 | 72.2 | 0.734 | 65.8 | 65.9 | 69.9 | 0.390 |
| SYSFOR | 71.5 | 0.758 | 65.8 | 66.0 | 69.9 | 0.399 | 74.7 | 0.807 | 64.1 | 64.2 | 70.7 | 0.418 | 74.4 | 0.815 | 67.3 | 67.9 | 71.5 | 0.446 |

<Table 10> Prediction Results of the Selected ML Techniques Over the Five Releases of JFreeChart Using Inter-release Validation (VS2)

| Tech. | JFreeChart 0.7.0 using JFreeChart 0.6.0 | | | | | | JFreeChart 0.7.1 using JFreeChart 0.7.0 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC |
| BN | **73.0** | 0.736 | 55.1 | 59.9 | 62.4 | **0.449** | 72.2 | 0.725 | 69.4 | 69.4 | 72.1 | 0.413 |
| NB | 62.7 | 0.685 | 26.2 | 34.9 | 39.5 | 0.205 | 72.2 | 0.725 | 69.4 | 69.4 | 72.1 | 0.413 |
| FLDA | 71.9 | 0.722 | 24.1 | 30.3 | 31.9 | 0.207 | 71.8 | 0.721 | 69.1 | 69.1 | 71.7 | 0.411 |
| LR | 57.1 | 0.596 | 23.7 | 27.4 | 37.5 | 0.024 | 72.1 | 0.721 | 66.6 | 66.8 | 72.0 | 0.343 |
| MLP | 69.5 | **0.811** | 62.8 | 62.9 | 68.0 | 0.348 | 72.4 | 0.727 | 69.6 | 69.6 | 72.3 | 0.414 |
| RBFN | 66.3 | 0.799 | 34.3 | 44.0 | 46.1 | 0.315 | 60.5 | 0.617 | 57.3 | 57.3 | 60.4 | 0.181 |
| ADB | 61.7 | 0.621 | 48.5 | 49.0 | 56.9 | 0.170 | 72.4 | 0.733 | 69.1 | 69.1 | 72.1 | 0.421 |
| BAG | 52.4 | 0.535 | 27.4 | 28.6 | 40.1 | -0.059 | 72.4 | 0.740 | 70.0 | 70.1 | 72.4 | 0.417 |
| DAG | 61.3 | 0.627 | 22.2 | 30.9 | 35.9 | 0.169 | 67.5 | 0.665 | 67.1 | 67.4 | 67.9 | 0.338 |
| FC | 70.7 | 0.758 | 51.3 | 55.9 | 59.6 | 0.390 | 69.6 | 0.709 | 69.4 | 69.8 | 70.1 | 0.381 |
| LB | 53.6 | 0.538 | 30.3 | 31.5 | 42.6 | -0.029 | 68.0 | 0.747 | 68.7 | 68.9 | 69.7 | 0.383 |
| MCC | 57.2 | 0.596 | 23.8 | 27.5 | 37.5 | 0.024 | 61.4 | 0.589 | 55.9 | 55.9 | 60.6 | 0.195 |
| RSS | 62.5 | 0.664 | 32.1 | 37.8 | 44.5 | 0.189 | 74.0 | 0.738 | 71.1 | 71.2 | 73.8 | 0.451 |
| DTNB | 56.9 | 0.512 | 23.6 | 27.3 | 37.4 | 0.024 | 74.4 | 0.724 | 71.9 | 72.0 | 74.4 | 0.463 |
| FURIA | 60.4 | 0.566 | 28.2 | 33.1 | 41.2 | 0.122 | 62.2 | 0.643 | 60.0 | 60.1 | 62.4 | 0.223 |
| MOD | 69.0 | 0.714 | **72.0** | **73.1** | **69.3** | 0.415 | 69.1 | 0.714 | 72.0 | 73.2 | 69.4 | 0.416 |
| NNGE | 63.9 | 0.598 | 40.4 | 43.9 | 51.2 | 0.214 | 69.1 | 0.697 | 68.1 | 68.5 | 69.5 | 0.362 |
| PART | 61.2 | 0.533 | 18.6 | 29.3 | 32.4 | 0.183 | 65.5 | 0.611 | 64.3 | 64.6 | 65.9 | 0.293 |
| ADT | 63.5 | 0.564 | 26.6 | 37.0 | 39.6 | 0.253 | 65.4 | 0.685 | 59.2 | 59.2 | 64.2 | 0.270 |
| HFT | 68.0 | 0.788 | 45.3 | 50.6 | 54.9 | 0.338 | **75.7** | **0.779** | 72.9 | 73.0 | **75.5** | **0.488** |
| J48 | 62.7 | 0.522 | 26.2 | 34.9 | 39.5 | 0.205 | 65.3 | 0.636 | 63.7 | 63.8 | 65.6 | 0.289 |
| LMT | 65.8 | 0.703 | 45.7 | 48.7 | 55.3 | 0.268 | 69.9 | 0.721 | 69.0 | 69.0 | 68.7 | 0.411 |
| RF | 66.9 | 0.607 | 40.0 | 47.1 | 50.6 | 0.323 | 74.4 | 0.749 | **73.4** | **73.7** | 74.8 | 0.475 |
| CART | 71.0 | 0.665 | 49.4 | 55.2 | 49.2 | 0.402 | 72.6 | 0.722 | 69.2 | 69.2 | 72.2 | 0.422 |
| SYSFOR | 67.1 | 0.784 | 44.6 | 49.2 | 54.6 | 0.304 | 72.0 | 0.723 | 69.2 | 69.2 | 71.9 | 0.412 |

<Table 10> Prediction Results of the Selected ML Techniques Over the Five Releases of JFreeChart Using Inter-release Validation (VS2) (Cont.)

| Tech. | JFreeChart 0.7.2 using JFreeChart 0.7.1 | | | | | | JFreeChart 0.7.3 using JFreeChart 0.7.2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC |
| BN | **79.2** | 0.850 | **79.0** | **79.1** | **79.3** | **0.565** | 77.3 | 0.845 | 71.0 | 71.2 | 75.3 | 0.509 |
| NB | 66.3 | 0.784 | 57.6 | 58.4 | 49.0 | 0.299 | 74.6 | 0.763 | 60.0 | 63.3 | 66.4 | 0.475 |
| FLDA | 66.7 | 0.659 | 68.0 | 68.0 | 65.8 | 0.295 | 67.2 | 0.664 | 68.5 | 68.5 | 66.3 | 0.297 |
| LR | 65.4 | 0.703 | 53.8 | 55.4 | 60.8 | 0.281 | 66.6 | 0.764 | 49.0 | 51.0 | 58.1 | 0.270 |
| MLP | 66.9 | 0.614 | 46.5 | 53.1 | 55.6 | 0.368 | 57.0 | 0.648 | 58.3 | 59.2 | 57.5 | 0.152 |
| RBFN | 67.1 | 0.812 | 60.2 | 60.7 | 65.1 | 0.317 | 68.1 | 0.790 | 59.3 | 59.5 | 65.4 | 0.317 |
| ADB | 74.3 | 0.855 | 72.6 | 72.6 | 73.9 | 0.456 | 72.3 | 0.806 | 59.7 | 61.1 | 66.4 | 0.402 |
| BAG | 67.2 | 0.817 | 59.4 | 60.1 | 64.7 | 0.319 | 70.9 | 0.779 | 56.0 | 58.2 | 63.5 | 0.375 |
| DAG | 70.7 | 0.786 | 61.3 | 62.9 | 66.9 | 0.395 | 67.0 | 0.819 | 43.7 | 48.3 | 53.8 | 0.295 |
| FC | 75.0 | 0.789 | 67.7 | 69.0 | 71.9 | 0.491 | 70.9 | 0.653 | 47.9 | 54.8 | 56.7 | 0.416 |
| LB | 67.4 | 0.751 | 57.7 | 59.0 | 63.8 | 0.324 | 68.3 | 0.817 | 46.9 | 51.1 | 56.3 | 0.326 |
| MCC | 65.4 | 0.703 | 53.8 | 55.4 | 60.7 | 0.281 | 66.3 | 0.761 | 48.8 | 50.7 | 57.8 | 0.269 |
| RSS | 68.8 | 0.767 | 60.9 | 61.8 | 66.2 | 0.352 | 68.3 | 0.763 | 49.1 | 52.2 | 58.1 | 0.319 |
| DTNB | 72.3 | 0.817 | 68.6 | 68.7 | 71.5 | 0.421 | 77.5 | 0.839 | 64.2 | 67.7 | 69.7 | 0.540 |
| FURIA | 68.0 | 0.685 | 60.3 | 61.1 | 65.5 | 0.340 | 71.4 | 0.691 | 49.4 | 56.0 | 57.8 | 0.429 |
| MOD | 77.5 | 0.771 | 73.0 | 73.5 | 76.0 | 0.536 | 66.3 | 0.619 | 43.2 | 47.4 | 53.5 | 0.276 |
| NNGE | 71.8 | 0.707 | 63.2 | 64.6 | 68.4 | 0.416 | 72.2 | 0.675 | 50.1 | 57.4 | 58.4 | 0.454 |
| PART | 68.7 | 0.800 | 53.4 | 57.3 | 61.0 | 0.376 | 68.5 | 0.726 | 44.6 | 50.2 | 54.5 | 0.336 |
| ADT | 70.9 | 0.834 | 59.7 | 62.3 | 65.8 | 0.416 | 68.2 | 0.822 | 50.1 | 52.8 | 58.9 | 0.316 |
| HFT | 67.0 | 0.804 | 56.9 | 58.3 | 63.2 | 0.314 | 84.8 | 0.871 | 79.3 | 80.0 | 82.2 | 0.667 |
| J48 | 78.9 | 0.743 | 77.7 | 77.7 | 78.9 | 0.551 | 66.5 | 0.660 | 43.3 | 47.5 | 53.7 | 0.276 |
| LMT | 66.9 | 0.618 | 56.7 | 58.2 | 63.1 | 0.323 | 69.2 | 0.792 | 48.7 | 52.7 | 57.8 | 0.341 |
| RF | 77.7 | **0.883** | 76.9 | 77.0 | 77.8 | 0.530 | 72.0 | 0.823 | 65.4 | 65.4 | 70.3 | 0.401 |
| CART | 61.2 | 0.545 | 48.3 | 49.7 | 56.2 | 0.188 | 71.2 | 0.645 | 53.5 | 56.9 | 61.5 | 0.386 |
| SYSFOR | 66.4 | 0.798 | 56.4 | 57.8 | 62.7 | 0.311 | **85.0** | **0.872** | **79.4** | **80.2** | **82.3** | **0.668** |

<Table 11> Prediction Results of the Selected ML Techniques Over the Five Releases of Heritrix Using Inter-release Validation (VS2)

| Tech. | Heritrix 0.4.0 using Heritrix 0.2.0 | | | | | | Heritrix 0.6.0 using Heritrix 0.4.0 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC |
| BN | 69.2 | 0.702 | 75.6 | 76.1 | 65.1 | 0.353 | 61.5 | 0.739 | 61.8 | 63.1 | 62.8 | 0.267 |
| NB | 70.9 | 0.763 | 77.9 | 78.6 | 65.2 | 0.390 | **72.4** | 0.721 | 65.2 | 65.2 | 71.2 | 0.402 |
| FLDA | 67.1 | 0.663 | 68.5 | 68.5 | 66.2 | 0.297 | 66.5 | 0.683 | 57.2 | 57.2 | 64.5 | 0.274 |
| LR | 74.5 | 0.752 | 79.1 | 79.4 | 72.1 | 0.462 | 67.5 | 0.668 | 60.5 | 60.5 | 66.7 | 0.310 |
| MLP | 69.8 | 0.701 | 74.1 | 74.2 | 68.4 | 0.365 | 70.6 | 0.738 | 64.3 | 64.4 | 70.1 | 0.373 |
| RBFN | 70.9 | 0.704 | 74.4 | 74.4 | 70.2 | 0.382 | 70.4 | 0.736 | 62.1 | 62.1 | 68.7 | 0.351 |
| ADB | 70.7 | 0.770 | 78.4 | 79.4 | 63.1 | 0.401 | 69.7 | 0.745 | 62.7 | 63.0 | 67.8 | 0.327 |
| BAG | 71.6 | 0.750 | 77.9 | 78.6 | 66.8 | 0.401 | 70.9 | 0.735 | 65.7 | 65.9 | 70.9 | 0.385 |
| DAG | 72.9 | 0.769 | 78.7 | 79.2 | 68.7 | 0.428 | 67.7 | 0.745 | 63.3 | 63.7 | 68.2 | 0.333 |
| FC | 68.0 | 0.655 | 76.2 | 77.3 | 60.0 | 0.338 | 71.2 | 0.712 | 61.4 | 61.5 | 68.3 | 0.365 |
| LB | 73.8 | **0.799** | 79.1 | 79.6 | 70.2 | 0.452 | 65.7 | **0.762** | 61.4 | 61.8 | 66.2 | 0.298 |
| MCC | 74.9 | 0.757 | 79.6 | 79.9 | **72.5** | 0.465 | 67.5 | 0.668 | 60.5 | 60.5 | 66.7 | 0.310 |
| RSS | 68.5 | 0.696 | 75.7 | 76.4 | 63.0 | 0.342 | 71.6 | 0.749 | 64.8 | 64.8 | 70.6 | 0.383 |
| DTNB | 68.8 | 0.773 | 76.0 | 76.7 | 63.2 | 0.343 | 71.4 | 0.742 | 67.0 | 67.4 | 71.8 | 0.402 |
| FURIA | 67.7 | 0.702 | 72.4 | 72.5 | 66.1 | 0.313 | 70.3 | 0.706 | 64.7 | 65.0 | 70.1 | 0.369 |
| MOD | 72.6 | 0.704 | 79.1 | 79.9 | 66.7 | 0.435 | 69.4 | 0.702 | 65.6 | 65.9 | 70.1 | 0.372 |
| NNGE | 73.1 | 0.707 | 79.8 | 80.8 | 66.5 | 0.456 | 71.6 | 0.724 | **68.1** | **68.5** | **72.3** | **0.419** |
| PART | 69.6 | 0.662 | 76.1 | 76.6 | 65.5 | 0.356 | 72.3 | 0.703 | 62.2 | 62.4 | 69.0 | 0.387 |
| ADT | **75.1** | 0.787 | **80.6** | **81.1** | 71.3 | **0.484** | 64.6 | 0.716 | 61.7 | 62.3 | 65.6 | 0.289 |
| HFT | 70.3 | 0.721 | 77.2 | 78.0 | 64.8 | 0.374 | 66.5 | 0.730 | 59.1 | 59.2 | 65.6 | 0.286 |
| J48 | 70.2 | 0.656 | 75.7 | 76.0 | 67.2 | 0.366 | 65.8 | 0.694 | 61.5 | 62.0 | 66.4 | 0.299 |
| LMT | 71.7 | 0.771 | 79.3 | 80.5 | 63.7 | 0.436 | 66.9 | 0.726 | 59.5 | 59.5 | 65.9 | 0.295 |
| RF | 68.2 | 0.771 | 75.0 | 75.6 | 63.4 | 0.327 | 65.9 | 0.707 | 62.6 | 63.2 | 66.8 | 0.309 |
| CART | 70.5 | 0.659 | 75.8 | 76.1 | 67.8 | 0.378 | 69.9 | 0.679 | 60.4 | 60.4 | 67.3 | 0.342 |
| SYSFOR | 69.8 | 0.709 | 76.7 | 77.5 | 64.4 | 0.372 | 67.2 | 0.730 | 59.7 | 59.7 | 66.1 | 0.296 |

<Table 11> Prediction Results of the Selected ML Techniques Over the Five Releases of Heritrix Using Inter-release Validation (VS2) (Cont.)

| Tech. | Heritrix 0.8.0 using Heritrix 0.6.0 | | | | | | Heritrix 0.10.0 using Heritrix 0.8.0 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | AUC | F-S | G-m1 | G-m2 | MCC | Acc. | AUC | F-S | G-m1 | G-m2 | MCC |
| BN | 68.8 | **0.753** | **60.2** | 60.8 | **65.9** | 0.338 | 59.9 | 0.709 | 61.6 | 63.6 | 61.4 | 0.259 |
| NB | 58.3 | 0.670 | 34.3 | 37.6 | 46.0 | 0.106 | 66.6 | 0.684 | 60.8 | 61.1 | 66.7 | 0.302 |
| FLDA | 60.9 | 0.695 | 43.7 | 45.8 | 53.2 | 0.179 | 64.1 | 0.646 | 57.8 | 58.0 | 63.9 | 0.253 |
| LR | 60.9 | 0.684 | 40.4 | 43.5 | 50.8 | 0.181 | 63.9 | 0.634 | 56.3 | 56.5 | 63.2 | 0.236 |
| MLP | 61.0 | 0.583 | 42.7 | 45.2 | 52.5 | 0.179 | 63.2 | 0.635 | 57.1 | 57.5 | 63.2 | 0.239 |
| RBFN | 67.2 | 0.695 | 55.7 | 57.0 | 62.4 | 0.314 | 55.3 | 0.653 | 49.6 | 50.1 | 55.6 | 0.087 |
| ADB | **72.5** | 0.725 | 58.2 | **62.0** | 64.8 | **0.450** | 64.2 | 0.651 | 58.2 | 58.6 | 64.2 | 0.256 |
| BAG | 61.9 | 0.712 | 37.5 | 42.3 | 48.7 | 0.206 | 64.8 | 0.679 | 59.2 | 59.5 | 65.0 | 0.269 |
| DAG | 63.1 | 0.703 | 43.0 | 46.6 | 53.0 | 0.229 | 63.9 | 0.638 | 59.9 | 60.5 | 64.8 | 0.267 |
| FC | 63.4 | 0.678 | 31.5 | 42.3 | 43.6 | 0.308 | 67.1 | 0.657 | 52.5 | 52.8 | 61.6 | 0.257 |
| LB | 67.9 | 0.692 | 49.2 | 53.9 | 57.8 | 0.356 | 63.8 | 0.677 | 60.9 | 61.7 | 65.5 | 0.280 |
| MCC | 60.9 | 0.684 | 40.4 | 43.5 | 50.7 | 0.181 | 64.1 | 0.636 | 56.5 | 56.7 | 63.4 | 0.237 |
| RSS | 60.1 | 0.683 | 29.8 | 36.0 | 42.4 | 0.165 | **69.5** | 0.688 | 63.7 | 64.0 | 69.6 | 0.354 |
| DTNB | 64.0 | 0.661 | 48.5 | 50.5 | 56.8 | 0.249 | 63.4 | 0.694 | 59.9 | 60.7 | 64.4 | 0.263 |
| FURIA | 65.2 | 0.660 | 43.2 | 48.7 | 53.0 | 0.302 | 56.8 | 0.628 | 60.8 | 63.5 | 57.5 | 0.230 |
| MOD | 60.6 | 0.576 | 32.9 | 38.5 | 45.0 | 0.177 | 65.0 | 0.658 | 60.5 | 61.0 | 65.7 | 0.287 |
| NNGE | 66.4 | 0.642 | 51.1 | 53.5 | 59.2 | 0.295 | 60.5 | 0.628 | 59.0 | 60.1 | 62.0 | 0.225 |
| PART | 64.0 | 0.649 | 38.8 | 45.5 | 49.6 | 0.281 | 63.9 | 0.580 | 58.2 | 58.5 | 64.2 | 0.250 |
| ADT | 61.5 | 0.727 | 39.7 | 45.1 | 50.4 | 0.256 | 59.9 | 0.674 | 54.3 | 54.7 | 60.3 | 0.175 |
| HFT | 65.0 | 0.733 | 40.6 | 47.1 | 51.1 | 0.297 | 66.0 | 0.694 | 59.1 | 59.3 | 65.6 | 0.277 |
| J48 | 64.0 | 0.653 | 40.0 | 45.9 | 50.5 | 0.274 | 65.9 | 0.676 | 61.1 | 61.6 | 66.5 | 0.296 |
| LMT | 63.2 | 0.716 | 39.6 | 44.5 | 50.3 | 0.239 | 60.6 | 0.629 | 51.7 | 51.7 | 59.3 | 0.159 |
| RF | 68.8 | 0.740 | 46.8 | 54.3 | 56.0 | 0.402 | 68.8 | **0.727** | **66.2** | **67.0** | **70.3** | **0.383** |
| CART | 69.9 | 0.673 | 58.4 | 60.1 | 64.9 | 0.371 | 60.4 | 0.643 | 49.4 | 49.4 | 58.0 | 0.145 |
| SYSFOR | 63.7 | 0.729 | 37.3 | 44.2 | 48.5 | 0.268 | 65.7 | 0.692 | 58.8 | 59.0 | 65.3 | 0.276 |

<Table 12> Kruskal?Wallis Test Results for Comparison between the Validation Techniques over the JFreeChart and Heritrix Versions Using AUC as a Performance Measure

| Kruskal – Wallis test results | | | Mean Ranks | | |
|---|---|---|---|---|---|
| Test Statistics | JFreeChart | Heritrix | Validation tech. | JFreeChart | Heritrix |
| Chi-square value | 127.43 | 58.16 | 10-fold validation | 179.07 | 148.19 |
| *DF* | 2 | 2 | CBFS + 10-fold validation (VS1) | 200.89 | 198.39 |
| *p*-value | 0.000 | 0.000 | Inter-release validation (VS2) | 71.55 | 104.93 |

Note: $H_0$: The samples do not belong to the same population;
$H_a$: The samples belong to the same population;
Since the evaluated p-value is less than the threshold of significance α, therefore, the null hypothesis H0 is accepted

### 5.4. RQ4:

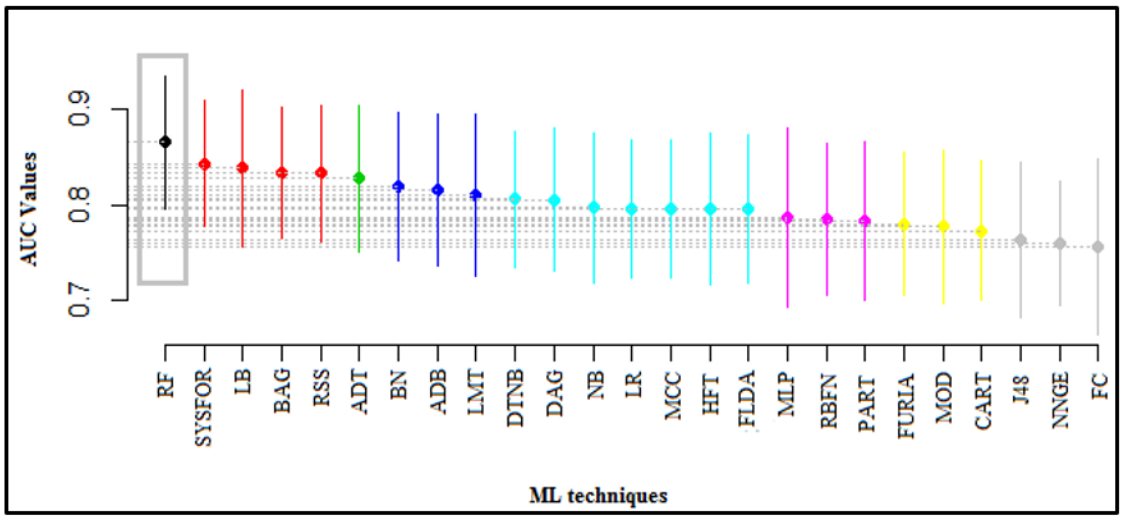
*Which are the best and the worst techniques for change proneness prediction of files of the selected Java-based software projects?*

As observed in <Table 8> ~ <Table 11>, each of the selected ML techniques are evaluated on multiple versions of the two Java-based software projects. As a result, it is hard to evidently establish the predictive pre-eminence of one technique over the other. Therefore, we apply the Scott-Knott cluster analysis (Jelihovschi et al., 2014) (as detailed in Section 4.7.2) to statistically compare the performance of the ML techniques over each of the specific validation scenarios (VS1 and VS2) according to their AUC values and then cluster them into subdivisions, where each subdivision or subgroup consists of those techniques that are significantly similar.

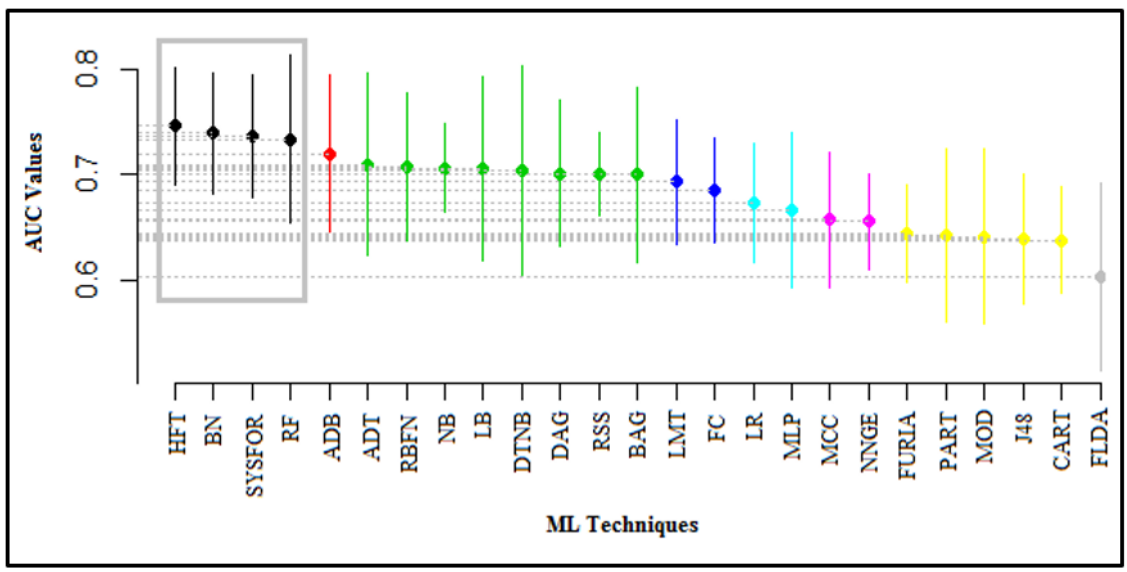
All the shortlisted ML techniques analysed in RQ2 are further analysed using Scott-Knott cluster analysis for the purpose of identifying similar groups of methods and to select the group with highest AUC (i.e., the best group). We used R (version 3.6) and RStudio (version 1.1.456) to perform the Scott-Knott analysis. One could therefore obtain a group of techniques which have a statistically similar highest performance in terms of AUC values via Scott-Knott analysis, as opposed to non-parametric tests like the Friedman's test which allocates mean ranks to individual techniques on the basis their values with respect to a performance measure.

As the Scott-Knott algorithm assumes that the distribution of AUC values is approximately normal, we tested the distribution of AUCs from each of the twenty five ML techniques using the Shapiro-Wilk test and found that they were normally distributed. We also compared the mean and median of AUC values obtained per ML technique per project and observed that mean to median ratio for every ML technique lies between 0.96-1.05. This indicates that the AUC values are normally distributed and could be utilized as input to the Scott-Knott test.

<Figure 2> shows graphical plot of Scott-Knott cluster analysis based on Anova significance test and using AUC values obtained by the JFreeChart and Heritrix datasets via the twenty five shortlisted ML techniques. The x-axis indicates the ML technique organized as per their ranks where better performing techniques start from left hand side. The AUC values are indicated on the y-axis and mean of the AUC values are represented by the small circles on each vertical line. The gray box on the left in each figure indicates the best subgroup of ML techniques that




<Figure 2(a)> Scott Knott Analysis for ML Techniques during the CBFS + 10 Fold Validation (VS1)




<Figure 2(b)> Scott Knott Analysis for ML Techniques during the Inter-release Validation (VS2)

exhibit significantly highest AUC. The Scott-Knott analysis resulted in eight homogeneous clusters wherein techniques in each of the clusters show statistically similar performance in both the <Figure 2(a)> and <Figure 2(b)>.

<Figure 2(a)> presents the Scott Knott analysis test results for the ML techniques during the CBFS + 10 fold validation for both the software projects. As observed from <Figure 2(a)> the RF technique obtains the highest AUC values for software change-prediction over the various JFreeChart and Heritrix software project versions and this performance is statisti-

cally significantly higher from the remaining techniques. SYSFOR, LB, BAG and RSS techniques follow the RF technique and are observed to exhibit statistically similar performance to each other in terms of AUC values.

<Figure 2(b)> presents the Scott Knott analysis test results for the ML techniques during the inter-release validation for both the software projects. As observed from <Figure 2(b)> the HFT, BN, SYSFOR and RF techniques perform statistically significantly higher for estimating the trend of software change-prediction over the various JFreeChart and Heritrix software project versions. The ADB technique follows next.

The Scott-Knott cluster analysis concludes the RF technique to be the best performing technique for software change prediction over the selected JFreeChart and Heritrix datasets during CBFS + 10 fold validation (VS1), with no other selected ML technique exhibiting a statistically similar high predictive performance. However, for the inter-release validation (VS2), four techniques are observed to occupy the best performing cluster. Since each homogeneous cluster consist of ML techniques that have a statistical similarity in performance, thus one cannot just rely only on the ranking obtained by Scott-Knott as shown in <Figure 2> to ascertain the best performing ML technique. Therefore, we consult Borda count method (Koch and Mitlöhner, 2009) to rank all the techniques in the best performing cluster across all the remaining performance measures which are Accuracy (*Acc.*), F-Measure or F-Score (*F-S*), G-mean1 (*G-m1*), G-mean2 (*G-m2*), and Matthews correlation coefficient (*MCC*).

<Table 13> shows the ML techniques sorted by the calculated Borda score seen in all performance measures over each validation scenario. The Borda score for each ML technique is calculated in the same way as explained in the example given in 4.7.4. The ML technique with the largest score is ranked first. We also analysed the techniques in the second best performing cluster using the Borda score, simply to assess their relative performance with respect to each other vis-à-vis the remaining performance measures.

As observed from <Table 13>, the RF technique performs the best for VS1 even when the remaining performance measures are taken into consideration and is concluded to be the best ML technique among the selected techniques for version to version change-proneness prediction of Java files over the selected JFreeChart and Heritrix versions. This is followed by the techniques from the second cluster wherein the ensemble technique of BAG (Bagging) performs the best followed by another ML technique RSS (Random Sub Space).

Unlike the Scott-Knott results, it is clear from the final scores form the Ranked Voting using Borda

<Table 13> Ranks Allotted to Techniques After Borda Ranking

| CBFS + 10 fold validation (VS1) | | Inter release validation (VS2) | |
|---|---|---|---|
| Rank | ML technique | Rank | ML technique |
| 1 | RF | 1 | BN |
| 2 | BAG | 2 | RF |
| 3 | RSS | 3 | ADB |
| 4 | SYSFOR | 4 | HFT |
| 5 | LB | 5 | SYSFOR |

counting scheme that as far as the Inter-release validation(VS2) is concerned, the BN technique outperforms all other techniques in regard to the selected performance measures for version to version software change prediction of Java files. This is followed by the RF technique which is also ascertained to perform the best for the VS1. This is followed by the ensemble technique of ADB (ADaBoost) which was observed to occupy the second best performing cluster in the Scott-Knott analysis as seen in <Figure 2(b)>. This indicates that one should analyse the performance of the ML techniques taking maximum number of performance measures into consideration as a technique performing well as per one measure might not perform well with respect to other measures.

**Summarizing the results drawn from the Scott-Knott cluster analysis:**

The RF technique outperforms all other ML techniques including those selected under the Decision Tree category with respect to change-proneness prediction of Java files during VS1 and obtains the second highest performance (the highest among the decision tree classifiers) during VS2 over the selected releases of both the Java-based software projects. Even though the BAG algorithm comes close in terms of prediction performance during VS1, it is outperformed by the RF technique. This is because RF technique is an improvement over BAG. The chief drawback with decision trees, such as CART, which are observed to underperform in both the validation scenarios, is that they employ the greedy algorithm to choose the splitting feature that minimizes error. Even in the case of the BAG technique, the decision nodes can comprise of many structural semblances and as a result have strong correlations in their estimates. Ensembles merge predictions from several models which is effective only if the predictions of the sub-trees are weakly correlated or uncorrelated. RF alters this behaviour in a way that the sub-models are learned and therefore there is a lesser correlation among the subsequent predictions from all of the sub-trees. In CART the learning algorithm scrutinizes all the features and their values for the purpose of selecting the optimum split-point. The RF technique alters this process as well so as to limit the search of the learning algorithm to a random sample of attributes.

Another previously unexamined technique, SYSFOR, also exhibits high performance in predicting version to version change-proneness of Java files and is observed to be one of the top five techniques for both the validation scenarios. SYSFOR, also a decision tree classifier, outperforms other selected techniques because contrasting to the prevailing methodologies it does not need to construct a tree that employs a feature or an attribute that has a poor gain ratio. This results in the generation of superior logic rules and prediction accuracy of the trees that is higher than those generated by trees that are constructed by means of poor attributes. Also, disparate from other techniques, SYSFOR permits the selection of a numeric feature multiple times provided the gain ratios are prominently high and the split points are well segregated.

Bayesian classifiers like BN, is observed to perform the best for both the Java-based projects during VS2 taking all the performance measures into consideration. This is because the Bayesian classifiers, as opposed to other prediction techniques, do not rely on asymptotic approximation and provide accurate conclusions that are restricted to the data being analyzed. Bayesian classifiers also evaluate the parameter functions directly, sans the employment of the “plug-in” method (a method for the estimation of the func-

tionals by plugging the projected parameters in the functionals).

This efficiency of the Bayesian classifiers in both the validation settings is utilized for improving the predictive accuracy of the HFT technique by virtue of which it performs the best for predicting the trend of version to version change-proneness of Java files during VS2 for the JFreeChart versions and the best overall for VS2 during the Scott-Knott cluster analysis with AUC as the deciding performance measure. This is done in accordance with the empirical inference drawn by Gama and Medas (2005) which states that the BN technique when employed at the leaves of an HFT instead of the majority class classifier leads to a significant improvement in its classification accuracy.

Ensemble learners like RSS, ADB and LB methodologies show exceptional results for a majority of data sets for predicting the trend of version to version change-proneness of Java files, obtaining high mean ranks during the Scott-Knott cluster analysis. These techniques have also shown to exhibit acceptable and sometimes even excellent AUC values during the CBFS + 10 fold validation over the JFreeChart and Heritrix versions. Ensemble learners generate successful prediction models since they employ various ML methods in unison to find improved prediction ability than the individual ML techniques. Therefore, ensemble learners could similarly be employed on other Java-based data sets for generating change-proneness prediction models.

## Ⅵ. Threats to Validity

Several substantial results have been obtained in this analysis which are sufficient enough to answer the four RQs proposed in Section 1. Nonetheless, certain threats to the validity of our study still persist which are given as follows:

*Threats to internal validity*: The internal validity signifies the degree to which accurate inferences could be obtained in regard to the causative outcome of the independent variables on the change statistic. The datasets incorporated in our analysis have been composed from the source code of successional releases of JFreeChart and Heritrix software projects. The objective of our investigation is to not assert causality, but to assess the performance of various ML techniques using the selected independent variables and the change statistic of a Java file with the aim of constructing competent prediction models for software change. Therefore, the threat to internal validity is found to persist in this analysis.

*Threats to conclusion validity*: Threats to conclusion validity pertain to the association between action and consequence. We applied feature selection using CBFS (Kaur and Mishra, 2019) and corrected the imbalance in the version datasets via the SMOTE technique (Chawla et al., 2002). We are certainly mindful of the effect that configuration plays in determining the performance of a ML technique (Frank et al., 2016), and subsequent work would be dedicated to measuring the degree to which the ML classification results are affected by this. We employed well-established performance measures to evaluate the change prediction models. Additionally, we statistically confirmed the predictive performance disparities in the different models via the Kruskal Wallis test and Scott-Knott's cluster analysis.

*Threats to external validity*: Threats to external validity concern with the generality of the results acquired. This threat could pertain to the versions selected for the purpose of validating the results of the stated RQs. The successional versions are selected from two Java-based software projects, and there

are many other projects available online from which data sets can be constructed for change-proneness prediction. Yet, we believe that the results drawn in this study can be reproduced by means of other Java-based data set versions. The choice of software metrics utilized for the construction of predictors could also pose as another threat to external validity. Even though we employed certain software metrics existing in literature (Chidamber and Kemerer, 1994; Halstead, 1979; Kaur and Mishra, 2019; Martin, 2003; McCabe, 1976; Oman et al., 1992), we acknowledge the fact that other software metrics are capable of displaying dissimilar results. Nevertheless, our objective is to examine the ability of various ML techniques for the purpose of change-proneness prediction rather than to conduct the performance comparison of change-proneness predictor variables.

## Ⅶ. Conclusion and Future Work

A framework for the version to version file change-proneness prediction was proposed in this article using 25 ML techniques on datasets gathered from multiple successional releases of two plugin projects: JFreeChart and Heritrix. Apposite pre-processing techniques like SMOTE, Inter-Quartile Range filter and Correlation-Based Feature Selection were applied on the datasets. The generated models were empirically validated using intra-release and inter-release scenarios on the various selected releases of the two Java-based projects in order to acquire all-pervading results. Additionally, Kruskal-Wallis test and Scott-Knott cluster analysis with Ranked Voting were employed for assessing the statistical significance of the acquired results and to gather valid and generalized inferences.

The primary conclusions of the analysis, most of which were gathered while answering the RQs in Section 5, are stated as follows:

- Among the 17 selected OO metrics, CBO, EC, CogC, RFC, DIT and SLOC were found to be significant predictors of change-proneness of Java files over the JFreeChart and Heritrix software releases using the CBFS method.
- The work authenticates the overall predictive potency of the selected ML techniques with respect to change-proneness prediction of Java files over JFreeChart and Heritrix software project releases under the intra-release (VS1) and inter-release (VS2) validation scenarios. The Kruskal-Wallis test results however indicate that the employment of SMOTE considerably augments the predictive ability of the models against those situations where no feature selection is applied. Moreover, granting that the ML techniques demonstrate their capability for estimating the version to version trend proneness of Java files for most of the selected versions during inter-release validation (VS2), their results are inferior in comparison to those obtained during CBFS + 10-fold validation (VS1).
- Furthermore, certain previously unexamined techniques like the techniques based on the concept of support vector machines like SMO and SPEG, evolutionary technique of MOEFC, the fuzzy classifier FLR, techniques under the miscellaneous category like CHIRP and VFI, show extremely poor performances specifically with respect to the AUC metric and are therefore deemed to be highly unsuitable for prediction of version to version change-proneness of Java files.
- In addition, the Scott-Knott cluster analysis

results conclude the RF technique to be the best performing technique for software change prediction over the selected JFreeChart and Heritrix datasets during CBFS + 10 fold validation (VS1), with no other selected ML technique exhibiting a statistically similar high predictive performance. However, for the inter-release validation (VS2), four techniques: HFT, BN, SYSFOR and RF are observed to statistically similarly highest performance with respect to the AUC values via the Scott-Knott analysis.

- The RF technique performs the best for VS1 even when the remaining performance measures are taken into consideration with the Borda count method and is concluded to be the best ML technique among the selected techniques for version to version change-proneness prediction of Java files over the selected JFreeChart and Heritrix versions. Additionally, as far as the Inter-release validation(VS2) is concerned, the BN technique outperforms all other techniques in regard to the selected performance measures for predicting the trend of version to version software change prediction of Java files, followed by the RF technique.

As a part of the future work, we propose to reproduce the analysis performed in this article via a categorical change statistic (representing the change degree or change type) and compare the predictive ability of the best performing ML techniques with respect to change-proneness prediction against Evolutionary algorithms and Search-Based techniques.

## <References>


[1] Aggarwal, K. K., Singh, Y., Kaur, A., and Malhotra, R. (2009). Empirical analysis for investigating the effect of object oriented metrics on fault proneness: A replicated case study. *Software Process: Improvement and Practice, 14*(1), 39-62.

[2] Arcuri, A., and Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering. *In 2011 33rd International Conference on Software Engineering*, IEEE, 1-10.

[3] Bansal, A. (2017). Empirical analysis of search based algorithms to identify change prone classes of open source software. *Computer Languages, Systems and Structures, 47*, 211-231.

[4] Bauer, E., and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36*(1-2), 105-139.

[5] Beller, M., Gousios, G., and Zaidman, A. (2017). Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. *In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, IEEE, 447-450.

[6] Bethea, R. M. (2018). *Statistical methods for engineers and scientists*. Routledge.

[7] Catolino, G., and Ferrucci, F. (2018). Ensemble techniques for software change prediction: A preliminary investigation. *In 2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, IEEE, 25-30.

[8] Catolino, G., Palomba, F., De Lucia, A., Ferrucci, F., and Zaidman, A. (2018). Enhancing change prediction models using developer-related factors. *Journal of Systems and Software, 143*, 14-28.

[9] Chaumun, M. A., Kabaili, H., Keller, R. K., and Lustman, F. (2002). A change impact model for changeability assessment in object-oriented software systems. *Science of Computer Programming, 45*(2-3),

155-174.

[10] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16,* 321-357.

[11] Chidamber, S. R., and Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476-493.

[12] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1-30.

[13] Elish, K. O., and Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software, 81*(5), 649-660.

[14] Elish, M. O., and Al-Zouri, A. A. (2014). Effectiveness of coupling metrics in identifying change-prone object-oriented classes. *In Proceedings of the International Conference on Software Engineering Research and Practice (SERP), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp)*.

[15] Espíndola, R. P., and Ebecken, N. F. F. (2005). On extending f-measure and g-mean metrics to multi-class problems. *WIT Transactions on Information and Communication Technologies, 35*, 10.

[16] Gama, J., Medas, P., and Rodrigues, P. (2005). Learning decision trees from dynamic data streams. *In Proceedings of the 2005 ACM symposium on Applied computing*, ACM, 573-577.

[17] Giger, E., Pinzger, M., and Gall, H. C. (2012). Can we predict types of code changes? An empirical analysis. *In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, IEEE, 217-226.

[18] Gray, D., Bowes, D., Davey, N., Sun, Y., and Christianson, B. (2012). Reflections on the NASA MDP data sets. *IET Software, 6*(6), 549-558.

[19] Halstead, M. H. (1979). *Advances in software science*. In Advances in Computers, Elsevier, pp. 119-172.

[20] Honglei, T., Wei, S., and Yanan, Z. (2009). The research on software metrics and software complexity metrics. *In 2009 International Forum on Computer Science-Technology and Applications*, IEEE, 131-136.

[21] Islam, Z., and Giggins, H. (2011). Knowledge discovery through SysFor: A systematically developed forest of multiple decision trees. *In Proceedings of the Ninth Australasian Data Mining Conference*, ACM, 195-204.

[22] Jelihovschi, E. G., Faria, J. C., and Allaman, I. B. (2014). ScottKnott: A package for performing the Scott-Knott clustering algorithm in R. *Trends in Applied and Computational Mathematics, 15*(1), 3-17.

[23] Jiménez, F., Martínez, C., Marzano, E., Palma, J. T., Sánchez, G., and Sciavicco, G. (2019). Multiobjective evolutionary feature selection for fuzzy classification. *IEEE Transactions on Fuzzy Systems, 27*(5), 1085-1099.

[24] Kaburlasos, V. G., Athanasiadis, I. N., and Mitkas, P. A. (2007). Fuzzy Lattice Reasoning (FLR) classifier and its application for ambient ozone estimation. *International Journal of Approximate Reasoning, 45*(1), 152-188.

[25] Kaur, L., and Mishra, A. (2018a). A comparative analysis of evolutionary algorithms for the prediction of software change. *In International Conference on Innovations in Information Technology (IIT)*, IEEE, 187-192.

[26] Kaur, L., and Mishra, A. (2018b). An empirical analysis for predicting source code file reusability using meta-classification algorithms. In *Advanced computational and communication paradigms* (pp. 493-504), Springer.

[27] Kaur, L., and Mishra, A. (2019). Cognitive complexity as a quantifier of version to version Java-based source code change: An empirical probe. *Information and Software Technology, 106*, 31-48.

[28] Khoshgoftaar, T. M., Gao, K., and Seliya, N. (2010). Attribute selection and imbalanced data: Problems in software defect prediction. *In 22nd IEEE International Conference on Tools with Artificial Intelligence,* IEEE, 137-144.

[29] Klamler, C. (2005). On the closeness aspect of three

voting rules: Borda – Copeland – Maximin. *Group Decision and Negotiation, 14*(3), 233-240.

[30] Koch, S., and Mitlöhner, J. (2009). Software project effort estimation with voting rules. *Decision Support Systems, 46*(4), 895-901.

[31] Kumar, L., Rath, S. K., and Sureka, A. (2017). Empirical analysis on effectiveness of source code metrics for predicting change-proneness. *In Proceedings of the 10th Innovations in Software Engineering Conference*, ACM, 4-14.

[32] Kumari, D., and Rajnish, K. (2019). *A systematic approach towards development of universal software fault prediction model using object-oriented design measurement*. In Nanoelectronics, Circuits and Communication Systems, Springer, pp. 515-526.

[33] Kuo, J. Y., Huang, F. C., Hung, C., Hong, L., and Yang, Z. (2012). The study of plagiarism detection for object-oriented programming. *In 2012 Sixth International Conference on Genetic and Evolutionary Computing*, IEEE, 188-191.

[34] Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering, 34*(4), 485-496.

[35] Lu, H., Zhou, Y., Xu, B., Leung, H., and Chen, L. (2012). The ability of object-oriented metrics to predict change-proneness: A meta-analysis. *Empirical Software Engineering, 17*(3), 200-242.

[36] Malhotra, L., and Bansal, A. J. (2014). *Prediction of change-prone classes using machine learning and statistical techniques*. In Advanced Research and Trends in New Technologies, Software, Human-Computer Interaction, and Communicability, IGI Global, pp. 193-202.

[37] Malhotra, R., and Bansal, A. (2015). Prediction of change prone classes using threshold methodology. *Advances in Computer Science and Information Technology, 2*, 30-35.

[38] Malhotra, R., and Jangra, R. (2017). Prediction & assessment of change prone classes using statistical & machine learning techniques. *Journal of Information Processing Systems, 13*(4), 778-804.

[39] Malhotra, R., and Khanna, M. (2013). Investigation of relationship between object-oriented metrics and change proneness. *International Journal of Machine Learning and Cybernetics, 4*(4), 273-286.

[40] Malhotra, R., and Khanna, M. (2014). Examining the effectiveness of machine learning algorithms for prediction of change prone classes. *In 2014 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 635-642.

[41] Malhotra, R., and Khanna, M. (2017). An exploratory study for software change prediction in object-oriented systems using hybridized techniques. *Automated Software Engineering, 24*(3), 673-717.

[42] Malhotra, R., and Khanna, M. (2018). Particle swarm optimization-based ensemble learning for software change prediction. *Information and Software Technology, 102*, 65-84.

[43] Malhotra, R., Shukla, S., and Sawhney, G. (2016). Assessment of defect prediction models using machine learning techniques for object-oriented systems. *In 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)*, IEEE, 577-583.

[44] Martin, R. C. (2002). *Agile software development: Principles, patterns, and practices*. Prentice Hall.

[45] McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, (4), 308-320.

[46] Menzies, T., Greenwald, J., and Frank, A. (2006). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2-13.

[47] Myrtveit, I., Stensrud, E., and Shepperd, M. (2005). Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering, 31*(5), 380-391.

[48] Oman, P., and Hagemeister, J. (1992). Metrics for assessing a software system's maintainability. *In Proceedings Conference on Software Maintenance*, IEEE, 337-344.

[49] Peng, Y., Kou, G., Wang, G., Wu, W., and Shi, Y. (2011). Ensemble of software defect predictors:

An AHP-based evaluation method. *International Journal of Information Technology & Decision Making, 10*(1), 187-206.

[50] Prati, R. C. (2015). Fuzzy rule classifiers for multi-label classification. *In 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, IEEE, 1-8.

[51] Purushothaman, R., and Perry, D. E. (2005). Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering, 31*(6), 511-526.

[52] Romano, D., and Pinzger, M. (2011). Using source code metrics to predict change-prone java interfaces. *In 2011 27th IEEE International Conference on Software Maintenance (ICSM)*, IEEE, 303-312.

[53] Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming, 127*(1), 3-30.

[54] Shepperd, M., Bowes, D., and Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering, 40*(6), 603-616.

[55] Tallón-Ballesteros, A. J., and Riquelme, J. C. (2014). Deleting or keeping outliers for classifier training? *In 2014 Sixth World Congress on Nature and Biologically Inspired*, IEEE, 281-286.

[56] Ting, K. M. and Witten, I. H. (1997). *Stacking bagged and dagged models*. Hamilton, New Zealand: University of Waikato, Department of Computer Science.

[57] Van Koten, C., and Gray, A. R. (2006). An application of Bayesian network for predicting object-oriented software maintainability. *Information and Software Technology, 48*(1), 59-67.

[58] Vassallo, C., Panichella, S., Palomba, F., Proksch, S., Zaidman, A., and Gall, H. C. (2018). Context is king: The developer perspective on the usage of static analysis tools. *In IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 38-49.

[59] Wilkinson, L., Anand, A., and Tuan, D. N. (2011). CHIRP: A new classifier based on composite hypercubes on iterated random projections. *In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 6-14.

[60] Ying, A. T., Murphy, G. C., Ng, R., and Chu-Carroll, M. C. (2004). Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering, 30*(9), 574-586.

[61] Zhou, Y., Leung, H., and Xu, B. (2009). Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. *IEEE Transactions on Software Engineering, 35*(5), 607-623.

## ◆ About the Authors ◆

**Loveleen Kaur**

Loveleen Kaur received her B. Tech. degree in Computer Science and Engineering and M.Tech degree in Software Systems. She is currently working as a Ph.D. research scholar with the Department of Computer Science and Engineering, Thapar University, Patiala, India. Her main research interests include Component-based software engineering, Intelligent Computing Methods, and Semantic Web technologies.

**Ashutosh Mishra**

Ashutosh Mishra received his Ph.D. degree in Computer Science and Engineering from Indian Institute of Technology (*BHU*), Varanasi, India. He is currently working as an Assistant Professor with the Department of Computer Science and Engineering, Thapar University, Patiala, India. His research interests include Software Engineering, Data Mining, Semantic Web technologies and Cognitive Computation. He is a member of various professional bodies and has published various research papers in national and international journals and conferences.

Submitted: November 7, 2019; 1st Revision: March 1, 2020; Accepted: April 2, 2020