UDC 004.42 : 005.3 **doi: 10.32620/reks.2025.3.04**

Manpreet KAUR[1], Dhavleesh RATTAN[2], Madan LAL[2]

[1] ***Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, Punjab, India***
[2] ***Punjabi University, Patiala, Punjab, India***

EMPIRICAL EVALUATION OF FEATURE SELECTION AND MACHINE LEARNING TECHNIQUES TO RECOMMEND CLONES FOR SOFTWARE REFACTORING

*The **article's subject matter** deals with the management of software clones. Software clones are duplicate code fragments that can exist in the same or different software files. Software clone detection and management has become a well-established research area. Software clones should be managed to minimize their ill-effects, as the presence of clones can increase the software's maintenance cost and resource requirements. Refactoring is a commonly used technique for managing clones. A software clone detection tool can detect many clones from the software, but not all detected clones are suitable for refactoring. A developer needs a subset of detected clones that can be easily refactored. This study aims to suggest software clones for refactoring using machine learning techniques. This study evaluates the performance of fourteen machine-learning algorithms and investigates the influence of three feature selection methods on clone recommendation accuracy. The **tasks** to be solved are as follows: selecting appropriate features from datasets, developing machine learning-based models that can suggest suitable clones for refactoring, and selecting an effective machine learning and feature selection algorithm for recommending clones for refactoring. The **methods** used for feature selection are correlation, InfoGain, and ReliefF. The study is conducted on datasets from six open-source software written in Java. The experimental **results** show that the Decision Tree and LogitBoost classifiers achieve the highest accuracy of 94.44 % on the Lucene dataset. ReliefF yields the best performance among the feature selection methods, particularly when used with the Decision Tree algorithm. This study **concludes** that Random Committee, Random Forest, and Decision Tree perform best when paired with correlation, InfoGain, and ReliefF, respectively. Overall, the Decision Tree classifier, combined with the ReliefF feature selection method, delivers the highest average precision, recall, and F-score across datasets.*

Keywords: *Software clones; Clone management; Clone recommendation, Clone refactoring, feature selection, machine learning.*

1. Introduction

Software clones are duplicate code fragments in the same or different source code files. When a programmer/developer copies a piece of code and pastes it at various locations in the source code with or without modification, software clones are generated in the software. Copying and pasting source code fragments is known as software cloning. The presence of such clones can increase the maintenance of software. For example, software cloning propagates the same bug at different locations if a bug exists in a copied code fragment. The presence of clones can also increase the size of the source code, which is a critical issue for devices with limited memory.

Software clones can be of various types, as discussed below:

- *Type-1 clones/ Exact clones:* The duplicate code fragments with minor differences, such as changes in comments or whitespaces;
- *Type-2 clones/Parameterised clones:* The duplicate code fragments with differences in variable and function names, comments, and whitespaces;
- *Type-3 clones/ Near Miss clones:* The duplicate code fragments with modifications, such as adding new source code lines or deleting existing source code;
- *Type-4 clones/Semantic clones:* The two code fragments are functionally similar but have different syntaxes. Such duplicate fragments are also known as semantic clones.

In previous research, many techniques have been developed to detect various types of software clones. Interested readers can read previous surveys [1, 2] to understand the working principle of these techniques. Since clones in software can be harmful, clones must be managed. Clones can be managed in several ways. Clone management refers to a group of activities that help to detect, remove, or avoid clones [3]. Such activities include clone detection [4], clone documentation, clone visualization [5], clone analysis [6], clone refactoring [7], and clone tracking [8].

1.1. Motivation and Objective

Clone refactoring is a popular method for managing clones [9]. Various refactoring techniques [10, 11] such

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

as the extract method [12], extract class, and pull-up method help to manage clones. The major challenge is selecting a set of clones that can be managed through refactoring [13]. A clone detection tool can find many clones of different types and granularities in the software. Manually selecting suitable clones for applying any refactoring technique is difficult.

To address this limitation, this study aims to evaluate the effectiveness of multiple machine learning algorithms along with three feature selection methods to automatically recommend suitable clones for refactoring. This approach not only reduces manual effort but also enhances clone management scalability. The novelty of our work lies in integrating and comparatively analyzing machine-learning based classification with feature selection methods to improve clone recommendations for refactoring, which is an underexplored area in the current literature.

1.2. Major Contribution of the Study

The major contributions of this study toward advancing automated clone refactoring are as follows:

- *Systematic feature selection:* This study applied three widely used feature selection methods: Correlation, InfoGain, and ReliefF, to identify the most relevant attributes of the clone refactoring dataset, enabling more focused and efficient learning.
- *Extensive model evaluation:* A detailed performance comparison is conducted using fourteen machine learning algorithms, each tested along with three feature selection methods. This provides a robust assessment of how various machine learning algorithm and feature-selection pairs perform in recommending suitable clones for refactoring.
- *Demonstration of the impact of feature selection: This* study highlights how feature selection improves classification accuracy, emphasizing its importance in enhancing clone recommendation systems.

This work fills a notable gap in existing research by combining and evaluating diverse machine learning models and feature selection methods in the context of clone refactoring, offering a practical and scalable solution for real-world software maintenance.

1.3. Paper Organization

The remainder of this paper is organized as follows. Section 2 discusses related work. This section elaborates on previous studies that proposed approaches for selecting suitable clones for refactoring. Section 2 also highlights the difference between our study and previous studies. The methodology is discussed in Section 3. Section 3 elaborates in detail the study's approach, including feature selection, datasets, machine learning algorithms, and evaluation metrics. Section 4 presents the results. The performance evaluation of various feature selection and ML algorithms is presented, and the results are compared with those of existing approaches. Section 5 discusses the results, and Section 6 concludes and provides future directions.

2. Related Work

Higo et al. [14] designed a filter named CCShaper to filter refactoring-oriented clones for the clone detection results of the CCFinder tool. CCShaper identified structural blocks in the code clone that are easy to combine and move. A tool Aries [15, 16] uses CCShaper to find structural blocks of code clones and then uses metrics like DCH (Dispersion of class Hierarchy), NRV (Number of Referred variables), or NAV (Number of Assigned variables) to identify clones suitable for Pull-Up method and Extract Method.

Schulze et al. [17] provided guidelines for clone refactoring by adding additional information, such as the clone location and the statement type of the code clones, to clone detection results. This information is used to provide refactoring proposals; however, applying the suggested refactoring for clone removal requires a separate refactoring tool.

Choi et al. [18] proposed a technique that integrates clone metrics to filter clones for refactoring. They used a web application to conduct an empirical investigation. They demonstrated that filtering clones for refactoring via a combination of clone metrics is more efficient than using a single clone metric.

Mondal et al. [19] developed a method for finding clones that are important for refactoring. They suggested that SPCP (Similarity Preserving Change Pattern) clones be considered while making refactoring decisions. SPCP clones are defined as two or more clone fragments from the same clone class that preserve similarity during clone evolution. They identified that SPCP clones with lower change couplings with other classes are good candidates for refactoring.

Wang and Godfrey [20], Rongrong et al. [21], Sheneamer [22], and Yue et al. [23] used machine learning algorithms to identify clones for refactoring. Wang and Godfrey [20] used the code, context, and history features of code clones. They built a machine-learning model to recommend clones for refactoring using a decision tree classifier. To prepare the dataset for training and testing the classifier, 323 clone instances with refactoring histories and 323 without refactoring histories were used. The proposed machine learning-based model achieved precision from 77.3% to 87.9% within-project testing, whereas cross-project testing generated precision from 73.2% to 88.5%.

Rongrong et al. [21] used various code clone features, including 13 static features and three evolution features, to build a method for identifying clones for refactoring. They employed Bayesian network, Naïve Bayes, and C4.5. Seven open-source projects written in C were evaluated. They concluded that the decision tree-based prediction model has higher accuracy than other models.

Sheneamer [22] proposed an approach to automatically advise clone refactoring. The strategy is based on AST features and uses K-nearest neighbor, Forest PA, Bagging, and Random Forest. They observed that Random Forest achieved better outcomes among all classifiers.

Yue et al. [23] developed a refactoring clone recommendation tool. The tool is based on 34 clone instances' features extracted from open-source projects in Java. They used AdaBoost to build a machine-learning-based model that automatically recommends clones for refactoring. In both with-in-project and cross-project testing, they concluded that AdaBoost recommended clones for refactoring with improved accuracy.

Fanqi [24] used SOM (Self-Organized Mapping) clustering to find refactorable clones. They retrieved metrics like POP (number of clones in a clone group), NIF (number of files in which clones of a clone group are distributed), LEN (length of code clone in terms of token), etc using CCFinder. The metrics of selected refactorable code clones were used to train the SOM model, which was then used to categorize the unknown code clones.

While previous studies have attempted to fill the gap between clone detection and meaningful recommendation of clones for refactoring, most have notable limitations. Recent studies have applied machine-learning algorithms to classify clones as refactorable or not, yielding promising results. However, these studies neither considered the impact of feature selection on model performance nor conducted broad comparisons among diverse algorithms. These existing studies used a limited set of machine learning classifiers, which raises concerns about the generalizability and robustness of their findings. Despite the well-documented influence of feature relevance on classification outcome in other domains, no previous studies provide a comparative analysis and evaluation of feature selection methods in the context of clone refactoring recommendations. These gaps highlight the need for a comprehensive, scalable, and generalizable framework that integrates a broad range of machine learning classifiers with diverse feature selection methods. Motivated by this, the current study conducts a large-scale empirical evaluation that not only evaluates fourteen machine-learning classifiers across six datasets but also integrates and compares three well-known feature selection methods: Correlation, Infogain, and ReliefF to generate accurate and generalizable clone recommendations for refactoring. The goal is to identify optimal combinations for accurately recommending clones for refactoring.

3. Methodology

Figure 1 shows the workflow of the current study. A labeled dataset related to clone refactoring was required to train the machine-learning models. Six open-source projects [23] were used as the labeled dataset of clone refactoring. Three feature subset selection methods were employed to select optimal features from six clone refactoring datasets and trained machine learning algorithms to classify refactorable and non-refactorable clones.

3.1. Feature selection

Feature selection algorithms are important in selecting the best features for machine learning algorithms. The accuracy and execution speed of the machine learning algorithms may be enhanced by this feature subset selection. There are two categories of feature selection algorithms: wrapper techniques and filter methods. While wrapper techniques use learning algorithms to select a subset of the best features, filter methods select features based on their relationship to the target. Three feature selection techniques were employed in the current work: Correlation, InfoGain, and ReliefF.

3.2. Datasets

The dataset used in this study comprises six open-source subject systems [23]. The dataset includes 666 clone instances belonging to six open-source software in Java (see Table 1). It consists of 333 clone refactoring instances and 333 clone instances without refactoring history. Each clone instance is represented in the form of 34 features. These 34 features belong to five types: code features, history features, syntactic difference features, relative location features, and co-change features among clones.

Table 1

Clone refactoring dataset [23]

Subject System	Number of refactored clones	Number of Non-refactored clones
Axis2	43	43
Eclipse. jdt. core	106	106
Elastic Search	33	33
JFreeChart	59	59
JRuby	65	65
Lucene	27	27

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

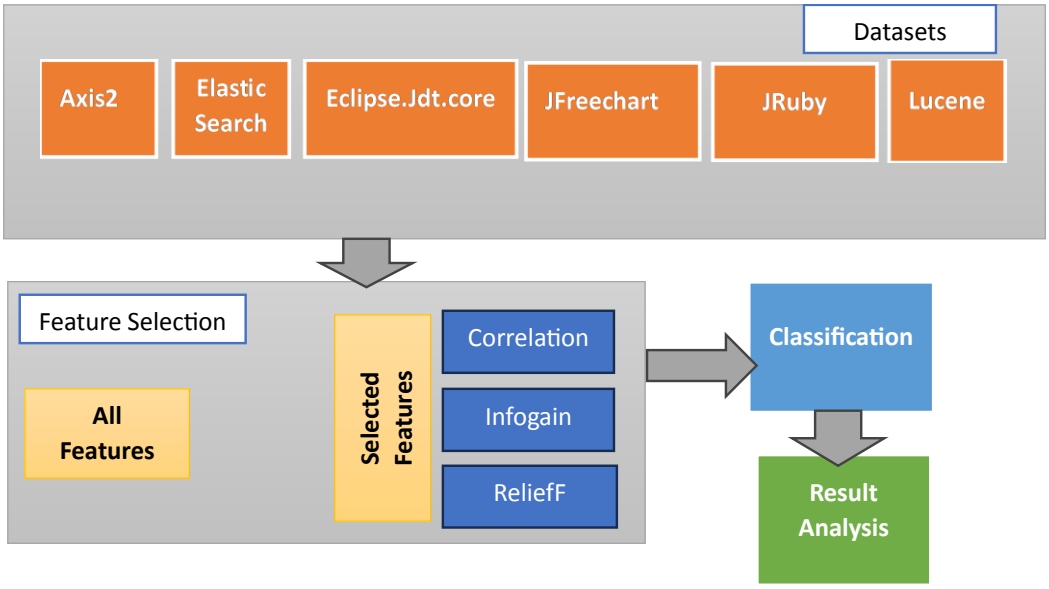

Figure 1. Workflow of the proposed approach

3.3. Machine learning algorithms used

This experiment used the following machine learning algorithms [25]:

- Decision Tree,
- Random Forest,
- Random Tree,
- Decision Table,
- AdaBoost,
- Bagging,
- LogitBoost,
- MulticlassClassifier,
- RandomCommitte,
- Random Subspace,
- IBK,
- SMO,
- Naïve Bayes,
- Multilayer Perceptron.

3.4. Evaluation Metrics

Many evaluation metrics exist to measure machine learning models' performance [25, 26]. In this experiment, the performance of different machine learning algorithms was measured using the following metrics.

***Precision*:** Ratio of relevant instances to retrieved instances

$$\text{Precision}=\frac{\text{TP(True Positives)}}{\text{TP(True Positives)}+\text{FP(False Positives)}}.$$

Here, **TP** (True Positives) represents the correctly identified refactorable clones. **FP** (False Positives) represents instances where the model incorrectly identifies a clone as a refactorable clone.

Therefore, precision measures how many clones are refactorable among all the recommended clones for refactoring.

***Recall*:** Ratio of retrieved relevant instances to total relevant instances

$$\text{Recall}=\frac{\text{TP(True Positives)}}{\text{TP(True Positives)}+\text{FN(False Negatives)}}.$$

Here, **TP** (True Positives) represents the correctly identified refactorable clones and **FN** (False Negatives) represents instances where the model fails to identify a refactorable clone, respectively.

Therefore, recall measures how many clones are recommended for refactoring among all known refactorable clones.

***F-Measure/ F1-score*:** Weighted average recall and precision values.

$$\text{F1-Score}=\frac{2*(\text{Recall}*\text{Precision})}{(\text{Recall}+\text{Precision})}.$$

F1-Measure provides a single measure for balancing precision and recall to check the model performance.

***Accuracy*:** Ratio of correctly retrieved instances to the total number of instances [27].

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}.$$

Here, **TP** (True Positives) represents the correctly identified refactorable clones, **TN**(True Negatives) represents instances where the model correctly identifies the clones which are not refactorable. **FP** (False Positives) represents instances where the model incorrectly identifies a clone as a refactorable clone and **FN** (False Negative) represents instances where the model fails to identify a refactorable clone.

Therefore, accuracy measures the proportion of correct predictions (both refactorable and non-refactorable clones) made by the model out of total predictions

4. Results

4.1. Performance evaluation of different Feature Selection (FS) techniques with various classifiers

Table 2 presents the evaluation of various feature selection approaches with different classifiers for the Axis2 dataset. For the correlation and ReliefF method, the classifiers Decision Tree, Decision Table, AdaBoost, Bagging, LogitBoost, MulticlassClassifier, RandomSubspace, IBK, Naïve Bayes, and MLP increase precision, recall, and F-measure with top 70% features. Decision Tree, Random Tree, Decision Table, AdaBoost, LogitBoost, MulticlassClassifier, RandomSubspace, IBK, and MLP provide better precision, recall, and F-measure with top 70% features for the Infogain method. The average performance of all classifiers increased with the use of the correlation, Infogain, and ReliefF feature selection methods for dataset Axis2.

As shown in Table 3, for the Eclipse.Jdt.core dataset, the performance of nine classifiers for the Infogain method with the top 70% features increased. For the ReliefF and correlation method, the performance of seven and four classifiers increased, respectively. The average performance of all classifiers is higher in the Infogain selection method than ReliefF and Correlation. In the case of Infogain, average precision is 81.5, the average recall is 81.21 and the average F-measure is 81.15. Random forest provides the highest performance in the eclipse.jdt.core dataset. With all features, the F-measure is 86.8. For Correlation and Infogain, it gives F-measure of 87.3, and for ReliefF selection method, it achieves the highest value for F-measure, i.e., 89.6.

As shown in Table 4, the performance of nine classifiers for the ReliefF method with top 70% features is increased for the Elastic search dataset. For the correlation and Infogain methods, the performance of eight and seven classifiers increased, respectively. The SMO classifier achieved the highest F-Measure with all features. After applying the feature selection method correlation and Infogain, the classifier LogitBoost achieved an F-Measure of 74.2, whereas the whereas with ReliefF, classifier AdaBoost achieved F-Measure of 74.2.

In dataset Jfreechart, after applying feature selection methods, the performance of six classifiers increased, as shown in Table 5. For the correlation method, the classifier Decision Table, LogitBoost, Bagging, AdaBoost, Random Subspace, and IBK show performance improvement compared to their results with all features. Similarly, classifier Bagging, Multiclassclassifier, RandomCommittee, RandomSubspace, IBK, and Naïve Bayes performed better with Infogain selection method, whereas Decision Tree, RandomForest, AdaBoost, Random Subspace, IBK and Naïve Bayes show better performance with ReliefF selection method. As we observed, the classifier Decision Tree achieved a maximum F-Measure of 93.2 for all features; however, with the selection method reliefF, the classifier Decision Tree achieved F-Measure of 94.1. In the correlation and Infogain selection method, the classifiers LogitBoost and RandomCommittee respectively achieved an F-Measure of 94.1.

In dataset JRuby (see Table 6), with the feature selection method Infogain, maximum performance improvement is achieved as seven classifiers, i.e., Random Tree, AdaBoost, Bagging, Random Subspace, IBK, SMO and Naïve Bayes show an increase in Precision, Recall, F-Measure. We observed that the classifier Random Forest gave a maximum F-measure of 85.3 with all features with dataset JRuby. In the case of correlation, the RandomCommittee classifier gave a maximum F-Measure of 83.7. For the Infogain selection method, the RandomForest classifier achieved a maximum F-measure of 83.8, whereas for the ReliefF selection method, the Random Tree classifier achieved the highest F-measure of 85.4.

In the Lucene dataset, as shown in Table 7, the Infogain feature selection method helps achieve better performance of five classifiers, Random Tree, AdaBoost, Multiclass, IBK and Naïve Bayes as compared to the performance of these classifiers with all features. With classifiers Decision Table, Bagging, Random Subspace and MLP gave the same performance with all features of the dataset and using the Infogain selection method with the top 70% features. The highest F-Measure of 94.4 was achieved with the classifier Decision Tree using all features and ReliefF selection method. With the selection method, correlation, the AdaBoost classifier achieved the highest F-Measure of 90.7. With the Infogain selection method, the Random Tree and AdaBoost classifiers achieved a maximum F-Measure of 90.7

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 2

Performance evaluation of different FS techniques with various classifiers using dataset Axis2

Classifier	All Features			Correlation + Top 70% Features			Infogain + Top 70% Features			ReliefF+ Top 70% Features		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Decision Tree	76.3	75.6	75.4	**81.1**	**80.2**	**80.1**	**79.3**	**79.1**	**79**	**83.7**	**83.7**	**83.7**
RandomForest	86.4	86	86	83.4	82.6	82.4	**86.8**	86	86	82.7	82.6	82.5
RandomTree	79.3	79.1	79	76.7	76.7	76.7	81.7	81.4	81.4	76.8	76.7	76.7
DecisionTable	84	82.6	82.4	**87.3**	**86**	**85.9**	**87.3**	**86**	**85.9**	**87.3**	**86**	**85.9**
AdaBoost	75.7	75.6	75.6	**81.7**	**81.4**	**81.4**	**78.3**	**77.9**	**77.8**	**76.3**	75.6	75.4
Bagging	81.6	80.2	80	**83.2**	**81.4**	**81.1**	**82.3**	80.2	79.9	**82.5**	**81.4**	**81.2**
LogitBoost	67.5	67.4	67.4	**82.5**	**81.4**	**81.2**	**84.4**	**83.7**	**83.6**	**77**	**76.7**	**76.7**
MulticlassClassifier	61.8	61.6	61.5	**69.8**	**69.8**	**69.8**	**72.1**	**72.1**	**72.1**	**71**	**70.9**	**70.9**
RandomCommitte	84	83.7	83.7	84.4	83.7	83.6	83	82.6	82.5	81.7	81.4	81.4
RandomSubspace	80.7	79	78.8	**84**	**82.6**	**82.4**	**83.2**	**81.4**	**81.1**	**83.4**	**82.6**	**82.4**
IBK	75.9	75.6	75.5	**77**	**76.7**	**76.7**	**78.3**	**77.9**	**77.8**	**79.1**	**79.1**	**79.1**
SMO	73.9	73.3	73.1	**76.3**	**75.6**	**75.4**	70.2	69.8	69.6	72.5	72.1	72
NaiveBayes	72.4	70.9	70.4	**79.6**	**79.1**	**79**	68.9	67.4	66.8	**74**	**72.1**	**71.5**
MLP	73.4	73.3	73.2	**83**	**82.6**	**82.5**	**86.1**	**86**	**86**	**80.2**	**80.2**	**80.2**

Table 3

Performance evaluation of different FS techniques with various classifiers using dataset Eclipse.Jdt.core

Classifier	All features			Correlation + Top 70% Features			Infogain + Top 70% Features			ReliefF + Top 70% Features		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Decision Tree	81.9	81.6	81.6	81.7	81.1	81	79.5	79.2	79.2	81.5	81.1	81.1
RandomForest	86.8	86.8	86.8	**87.3**	**87.3**	**87.3**	**87.3**	**87.3**	**87.3**	**89.6**	**89.6**	**89.6**
RandomTree	80.8	80.7	80.6	78.1	77.8	77.8	80.4	80.2	80.2	79.8	79.7	79.7
DecisionTable	79	78.3	78.2	74.7	74.5	74.5	**79.1**	**78.8**	**78.7**	**80.5**	**80.2**	**80.1**
AdaBoost	77.4	77.4	77.4	76.9	76.9	76.9	**78.8**	**78.8**	**78.8**	*77.4*	*77.4*	*77.3*
Bagging	80.7	80.7	80.6	**82.2**	**82.1**	**82.1**	**83**	**83**	**83**	**82.1**	**82.1**	**82.1**
LogitBoost	82.3	82.1	82	81.7	81.6	81.6	**83.1**	**83**	**83**	**84.5**	**84.4**	**84.4**
MulticlassClassifier	79.3	79.2	79.2	*79.3*	*79.2*	*79.2*	**80.9**	**80.7**	**80.6**	**79.8**	**79.7**	**79.7**
RandomCommitte	86.1	85.4	85.3	**86.7**	**86.3**	**86.3**	**86.9**	**86.8**	**86.8**	**88.2**	**87.7**	**87.7**
RandomSubspace	85	84.9	84.9	83.5	83.5	83.5	80.7	80.7	80.7	**85.8**	**85.8**	**85.8**
IBK	82.8	82.5	82.5	82.4	82.1	82	**83**	**83**	**83**	82.6	*82.5*	*82.5*
SMO	80.7	80.7	80.6	79.8	79.8	79.7	80.4	80.2	80.2	79.8	79.7	79.7
NaiveBayes	76	75.9	75.9	**78.4**	**77.8**	**77.7**	**79.1**	**78.3**	**78.1**	75.6	73.1	72.4
MLP	79	78.8	78.7	77	76.9	76.9	77.4	77.4	77.4	78.8	78.8	78.8

Table 4

Performance evaluation of different FS techniques with various classifiers using dataset Elastic search

Classifier	All features			Correlation + Top 70% features			Infogain + Top 70% features			ReliefF + Top 70% features		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Decision Tree	70	69.7	69.6	**73.5**	**72.7**	**72.5**	**75**	**72.7**	**72.1**	**74.8**	**74.2**	**74.1**
RandomForest	68.6	68.2	68	66.9	66.7	66.5	**72.2**	**71.2**	**70.9**	**71.4**	**71.2**	**71.2**
RandomTree	59.3	59.1	58.9	**74.8**	**74.2**	**74.1**	59.2	59.1	**59**	59.1	59.1	**59.1**
DecisionTable	54.5	53	48.8	54.5	53	48.8	54.5	53	48.8	54.5	53	48.8
AdaBoost	71.7	71.2	71	**72.7**	**72.7**	**72.7**	**73.1**	**72.7**	**72.6**	**74.4**	**74.2**	**74.2**
Bagging	63.8	63.6	63.5	63.8	63.6	63.5	59.1	59.1	59.1	60.8	60.6	60.5
LogitBoost	65.2	65.2	65.1	**74.4**	**74.2**	**74.2**	**74.3**	**74.2**	**74.2**	**69.8**	**69.7**	**69.7**
MulticlassClassifier	63.7	63.6	63.6	63.7	63.6	63.6	60.6	60.6	60.6	**65.2**	**65.2**	**65.1**
RandomCommitte	72.2	71.2	70.9	**73.1**	**72.7**	**72.6**	**73.5**	**72.7**	**72.5**	70	69.7	69.6
RandomSubspace	63.7	63.6	63.6	**69.8**	**69.7**	**69.7**	**69.6**	**68.2**	**67.6**	**73.1**	**72.7**	**72.6**
IBK	61	60.6	60.3	59.3	59.1	58.9	59.1	59.1	59.1	**65.5**	**65.2**	**65**
SMO	74.3	74.2	74.2	71.2	71.2	71.2	59.2	59.1	59	74.3	74.2	74.2
NaiveBayes	60.8	59.1	57.4	**69.1**	**65.2**	**63.3**	57.3	56.1	54.3	**66.6**	**63.6**	**61.9**
MLP	66.9	66.7	66.5	**72.8**	**72.7**	**72.7**	62.1	62.1	62.1	**70**	**69.7**	**69.6**

Table 5

Performance evaluation of different FS techniques with various classifiers using dataset JFreechart

Classifier	All features			Correlation + Top 70% features			Infogain + Top 70% features			ReliefF + Top 70% features		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Decision Tree	93.4	93.2	93.2	90.8	90.7	90.7	91.7	91.5	91.5	**94.2**	**94.1**	**94.1**
RandomForest	91.5	91.5	91.5	90.7	90.7	90.7	91.5	91.5	91.5	**92.4**	**92.4**	**92.4**
RandomTree	92.3	91.5	91.5	90.7	90.7	90.7	91.5	91.5	91.5	87.8	87.3	87.2
DecisionTable	87.4	85.6	85.4	**91**	**89**	**88.8**	87.4	85.6	85.4	87.4	85.6	85.4
AdaBoost	85.9	85.6	85.6	**87.8**	**87.3**	**87.2**	85.4	84.7	84.7	**86.6**	**86.4**	**86.4**
Bagging	87.8	87.3	87.2	**88.3**	**88.1**	**88.1**	**90**	**89.8**	**89.8**	87.8	87.3	87.2
LogitBoost	90.8	90.7	90.7	**94.4**	**94.1**	**94.1**	**91**	90.7	90.7	89.3	89	89
MulticlassClassifier	84.1	83.9	83.9	83.1	83.1	83	**85.9**	**85.6**	**85.6**	77.5	77.1	77
RandomCommitte	92.5	92.4	92.4	92.5	92.4	92.4	**94.1**	**94.1**	**94.1**	92.5	92.4	92.4
RandomSubspace	88.2	88.1	88.1	**90.7**	**90.7**	**90.7**	**89**	**89**	**89**	**90.7**	**90.7**	**90.7**
IBK	81.9	81.4	81.3	**85.1**	**84.7**	**84.7**	**86.1**	**85.6**	**85.5**	**84.9**	**84.7**	**84.7**
SMO	83.9	83.9	83.9	83.9	83.9	83.9	83.1	83.1	83	**84**	83.9	83.9
NaiveBayes	74.6	74.6	74.6	73.1	72.9	72.8	**76.3**	**76.3**	**76.3**	**77.1**	**77.1**	**77.1**
MLP	89.8	89.8	89.8	84.9	84.7	84.7	88.2	88.1	88.1	85.7	85.6	85.6

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 6

Performance evaluation of different FS techniques with various classifiers using dataset JRuby

Classifier	All features			Correlation + Top 70% features			Infogain + Top 70% features			ReliefF + Top 70% features		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Decision Tree	82.4	82.3	82.3	80.3	80	80	77.7	77.7	77.7	**85.6**	**85.4**	**85.4**
RandomForest	85.8	85.4	85.3	81.4	80.8	80.7	84.5	83.8	83.8	81.1	80.8	80.7
RandomTree	73.9	73.8	73.8	**76.2**	**76.2**	**76.1**	**75.4**	**75.4**	**75.4**	**85.4**	**85.4**	**85.4**
DecisionTable	81.5	80	79.8	79.2	76.2	75.5	81.5	80	79.8	**84.7**	**82.3**	**82**
AdaBoost	79.3	77.7	77.4	78.5	77.7	77.5	**81.4**	**78.5**	**77.9**	**85.2**	**80.8**	**80.1**
Bagging	82.1	80.8	80.6	**83.1**	**81.5**	**81.3**	**83.7**	**82.3**	**82.1**	82	79.2	78.8
LogitBoost	81.6	81.5	81.5	79.6	79.2	79.2	77.7	77.7	77.7	78.9	78.5	78.4
MulticlassClassifier	64	63.8	63.7	**66**	**65.4**	**65**	62.8	62.3	61.9	61.8	61.5	61.3
RandomCommitte	85.5	84.6	84.5	85.3	83.8	83.7	81.6	81.5	81.5	83.4	83.1	83
RandomSubspace	81.1	80	79.8	**81.5**	80	79.8	**84.2**	**83.1**	**82.9**	**83**	**80.8**	**80.4**
IBK	65.1	64.6	64.3	**65.6**	64.6	64.1	**67.6**	**66.9**	**66.6**	62.8	62.3	61.9
SMO	72.1	70.8	70.3	**76.4**	**74.6**	**74.2**	**76.4**	**74.6**	**74.2**	71.9	70	69.3
NaiveBayes	69.5	68.5	68	68.8	67.7	67.2	**75.8**	**73.1**	**72.4**	62.8	62.3	61.9
MLP	69	68.5	68.2	67.8	67.7	67.6	61.8	61.5	61.3	69	68.5	68.2

Table 7

Performance evaluation of different FS techniques with various classifiers using dataset Lucene

Classifier	All features			Correlation+ Top 70% features			Infogain + Top 70% features			ReliefF + Top 70% features		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Decision Tree	94.5	94.4	94.4	89.1	88.9	88.9	89.1	88.9	88.9	94.5	94.4	94.4
RandomForest	89.1	88.9	88.9	89.1	88.9	88.9	87.1	87	87	88.9	88.9	88.9
RandomTree	88.9	88.9	88.9	82.2	81.5	81.4	**90.8**	**90.7**	**90.7**	**92.8**	**92.6**	**92.6**
DecisionTable	87.1	87	87	87.1	87	87	87.1	87	87	87.1	87	87
AdaBoost	90.8	90.7	90.7	**91.3**	**90.7**	**90.7**	**92.2**	**90.7**	**90.7**	87.5	87	87
Bagging	89.8	88.9	88.8	89.8	88.9	88.8	89.8	88.9	88.8	89.8	88.9	88.8
LogitBoost	94.5	94.4	94.4	87.5	87	87	90.8	90.7	90.7	92.8	92.6	92.6
MulticlassClassifier	72.5	72.2	72.1	72.5	72.2	72.1	**77.8**	**75.9**	**75.5**	72.3	72.2	72.2
RandomCommitte	90.8	90.7	90.7	89.1	88.9	88.9	89.8	88.9	88.8	89.1	88.9	88.9
RandomSubspace	92.2	90.7	90.7	89.8	88.9	88.8	92.2	90.7	90.7	92.2	90.7	90.7
IBK	77.8	77.8	77.8	**77.9**	77.8	77.7	**81.7**	**81.5**	**81.5**	**81.7**	**81.5**	**81.5**
SMO	72.3	72.2	72.2	**74.2**	**74.1**	**74**	70.5	70.4	70.3	70.5	70.4	70.3
NaiveBayes	65.3	64.8	64.5	**71.4**	**70.4**	**70**	**69.2**	**68.5**	**68.2**	**71.4**	**70.4**	**70**
MLP	81.5	81.5	81.5	78.4	77.8	77.7	81.5	81.5	81.5	76	75.9	75.9

4.2. Accuracy of different Feature Selection techniques with various classifiers

Tables 8 and 9 report the accuracy of different feature selection techniques with various classifiers. On the dataset, Axis2, Random Forest, Decision Table, and MLP give the highest accuracy of 86.05. Random forest gives the highest accuracy of 89.62 on the Eclipse. Jdt.core dataset with the ReliefF feature selection algorithm. For the dataset, Elastic Search, Decision Tree, Random Tree, AdaBoost, LogitBoost and SMO give the highest accuracy of 74.24. On the dataset Jfreechart, Decision Tree, LogitBoost, and Random Committee achieve the highest accuracy of 94.07 using ReliefF, Correlation, and

Table 8

Accuracy of various classifiers on different datasets

Classifier	Axis2				Eclipse.Jdt.Core				Elastic Search			
	All features	Correlation (Top 70%)	Infoain (Top 70%)	ReliefF (Top 70%)	All features	Correlation (Top 70%)	Infogain (Top 70%)	ReliefF (Top 70%)	All features	Correlation (Top 70%)	Infogain (Top 70%)	ReliefF (Top 70%)
Decision Tree	75.58	80.23	79.07	83.72	81.6	81.13	79.25	81.13	69.7	72.73	72.73	**74.24**
RandomForest	**86.05**	82.56	**86.05**	82.56	85.85	87.26	87.26	**89.62**	68.18	66.67	71.21	71.21
RandomTree	79.07	76.74	81.4	76.74	78.77	77.83	80.19	79.72	59.09	**74.24**	59.09	59.09
DecisionTable	82.56	**86.05**	**86.05**	**86.05**	78.3	74.53	78.77	80.19	53.03	53.03	53.03	53.03
AdaBoost	75.58	81.4	77.91	75.58	77.36	76.89	78.77	77.36	71.21	72.73	72.73	**74.24**
Bagging	80.23	81.4	80.23	81.4	80.66	82.08	83.02	82.08	63.64	63.64	59.09	60.61
LogitBoost	67.44	81.4	83.72	76.74	82.08	81.6	83.02	84.43	65.15	**74.24**	**74.24**	69.7
MulticlassClassifier	61.63	69.77	72.09	70.93	79.25	79.25	80.66	77.83	63.64	63.64	60.61	65.15
RandomCommitte	83.72	83.72	82.56	81.4	85.85	86.32	86.79	87.74	71.21	72.73	72.73	69.7
RandomSubspace	79.07	82.56	81.4	82.56	83.96	83.49	80.66	85.85	63.64	63.64	68.18	72.73
IBK	75.58	76.74	77.91	79.07	82.55	82.08	83.02	82.55	60.61	59.09	59.09	65.15
SMO	73.26	75.58	69.77	72.09	80.66	79.72	80.19	79.72	**74.24**	71.21	59.09	**74.24**
NaiveBayes	70.93	79.07	67.44	72.09	75.94	77.83	78.3	73.11	59.09	65.15	56.06	63.64
MLP	73.26	82.56	**86.05**	80.23	79.72	76.88	77.36	78.77	66.67	72.73	62.12	69.7

Table 9

Accuracy of various classifiers on different datasets

Classifier	JFreeChart				JRuby				Lucene			
	All features	Correlation (Top 70%)	Infogain (Top 70%)	ReliefF (Top 70%)	All features	Correlation (Top 70%)	Infogain (Top 70%)	ReliefF (Top 70%)	All features	Correlation (Top 70%)	Infogain (Top70%)	ReliefF (Top 70%)
Decision Tree	93.22	90.68	91.53	**94.07**	82.31	80	77.69	**85.38**	**94.44**	88.89	88.89	**94.44**
RandomForest	91.53	90.68	91.53	92.37	**85.38**	80.77	83.85	80.77	88.89	88.89	87.04	88.89
RandomTree	91.53	90.68	91.53	87.29	73.85	76.15	75.38	**85.38**	88.89	81.48	90.74	92.59
DecisionTable	85.59	88.98	85.59	85.59	80	76.15	80	82.31	87.04	87.04	87.04	87.04
AdaBoost	85.59	87.29	84.75	86.44	77.69	77.69	78.46	80.77	90.74	90.74	90.74	87.04
Bagging	87.29	88.14	89.83	87.29	80.77	81.54	82.31	79.23	88.89	88.89	88.89	88.89
LogitBoost	90.68	**94.07**	90.68	88.98	81.54	79.23	77.69	78.46	**94.44**	87.04	90.74	92.59
MulticlassClassifier	83.9	83.05	85.59	77.12	63.85	65.38	62.31	61.54	72.22	72.22	75.93	72.22
RandomCommitte	92.37	92.37	**94.07**	92.37	84.62	83.85	81.54	83.08	90.74	88.89	88.89	88.89
RandomSubspace	88.14	90.68	88.98	90.68	80	80	83.08	80.77	90.74	88.89	90.74	90.74
IBK	81.36	84.75	85.59	84.75	64.62	64.62	66.92	62.31	77.78	77.78	81.48	81.48
SMO	83.9	83.9	83.05	83.9	70.77	74.62	74.62	70	72.22	74.07	70.37	70.37
NaiveBayes	74.58	72.88	76.27	77.12	68.46	67.69	73.08	62.31	64.81	70.37	68.52	70.37
MLP	89.83	84.75	88.14	85.59	68.46	67.69	61.54	68.46	81.48	77.78	81.48	75.93

Infogain feature selection methods respectively (see Table 9). On dataset, JRuby, Decision Tree, and Random Tree achieve the highest accuracy of 85.38 using the ReliefF selection method whereas Random Forest achieves the same accuracy with all features. On dataset Lucene, Decision Tree gives the highest accuracy of 94.44 with all features and ReliefF feature selection method whereas LogitBoost gives the same accuracy with all features.

4.3. Comparison with existing approaches

Table 10 compares the results of applying feature selection methods to recommending clones for refactoring with the existing machine learning-based approaches. The performance of AdaBoost and Naïve Bayes is improved with all selection methods. Random Forest achieves the same performance as the previous approach with Infogain. The Decision Tree's performance increased with ReliefF compared with the existing approach.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 10

Comparison with the existing approaches in terms of F-score

	Wang et. al [20]	Yue et. al [23]	Feature selection and Machine learning (Current Work)		
			Correlation	Infogain	ReliefF
AdaBoost	--	79.6	**81.06**	**80.41**	**80.06**
Random Forest	--	84.4	82.75	**84.4**	84.21
SMO	--	75.71	**76.4**	72.71	74.9
Naive Bayes	--	68.46	**71.66**	**69.35**	**69**
Decision Tree	72.8	82.75	82.22	81.4	**85.46**

5. Discussion

This study shows how the integration of feature selection methods enhances the performance of machine learning algorithms in recommending clones for refactoring. Experiments were conducted on datasets derived from six open-source Java projects, applying a systematic evaluation of three widely used feature selection methods: Correlation, Infogain, and ReliefF.

Initially, all classifiers were evaluated on the full feature set (i.e., without feature selection). Decision **Tree** and **LogitBoost** achieved the highest F-Measure of 94.4% on the Lucene dataset. **Random Forest** performed best on the Axis2, Eclispse.jdt.core, and Jruby datasets with F-measures of 86%, 86.8%, and 85.3%, respectively. Subsequently, the three feature selection methods were applied to assess their effect on classifier performance. The Infogain method consistently improved the performance of multiple classifiers across all datasets except Axis2, where correlation-based selection led to better results for eleven classifiers. We observed that the **ReliefF** method enables the **Decision Tree** classifier to achieve the highest average performance i.e., precision (85.71%), recall (85.48%), and F-score (85.46%).

Table 11 presents a detailed summary of the best-performing classifiers for each dataset and feature selection method. Figures 2-5 show performance of different classifiers for all features, correlation, Infogain, and ReliefF feature selection methods, respectively.

Our results indicate that the optimal choice of classifier and feature selection method varies depending on the dataset characteristics. For instance:

– **Random Committee** achieved the highest average precision, recall and F-measure using all features and correlation (see Figure 2 and Figure 3);

– **Random Forest and Decision Tree** were most effective when paired with InfoGain and ReliefF respectively (see Figure 4 and Figure 5);

– **AdaBoost, Bagging, RandomCommittee, SMO, Naïve Bayes, and MLP** achieved the highest average precision, recall, and F-measure using correlation feature selection.

– **Random Forest, LogitBoost, and MulticlassClassifier** achieved the highest average precision, recall, and F-Measure using Infogain feature selection;

– **Decision Tree, Random Tree, Decision Table, Random Subspace, and IBK** achieved the highest average precision, recall and F-Measure using ReliefF feature selection.

The superiority of the Decision Tree Classifier with ReliefF suggests that tree-based models benefit from attribute selection that emphasizes local instance-level relevance. This combination can be useful for clone-related tasks, where attribute behaviour varies across different projects. Additionally, Random Forest and Random Committee displayed robust performance across multiple datasets and feature selection attributes, indicating their resilience to noisy and redundant features.

These findings suggest that applying appropriate feature selection methods can substantially improve the performance of machine learning models for recommending clones for refactoring. The results reveal that feature selection is not merely a preprocessing step but a decisive step in improving classifier performance for clone recommendation systems. This has high practical implications as it reduces manual effort and improves the accuracy of identifying suitable clones for refactoring, especially in large software. Furthermore, our comparative evaluation offers a reproducible framework for future studies in this area. The methodology, which consists of clone data modeling, feature selection and classifier evaluation can be generalized to other domains such as bug-proneness prediction in cloned code, Test-case prioritization for clone-heavy modules and cross-language clone recommendation systems in Python and C++.

Table 11

ML algorithms achieved the highest F-Measure on various datasets

Dataset	All features	Correlation	Infogain	ReliefF
Axis2	Random Forest (86%)	Decision Table (85.9%)	Random Forest (86%)	Decision Table (85.9)
Eclispe.jdt.core	Random Forest (86.8%)	Random Forest (87.3%)	Random Forest (87.3%)	Random Forest (89.6%)
Elastic Search	SMO (74.2%)	LogitBoost (74.2%)	LogitBoost (74.2%)	AdaBoost, SMO (74.2%)
JFreeChart	Decision Tree (93.2%)	LogitBoost (94.1%)	Random Committee (94.1%)	Decision Tree (94.1%)
JRuby	Random Forest (85.3%)	Random Committee (83.7%)	Random Forest (83.8%)	Decision Tree and Random tree (85.4%)
Lucene	Decision Tree, LogitBoost (94.4%)	AdaBoost (90.7%)	Random Tree, LogitBoost, AdaBoost (90.7%)	Decision Tree (94.4%)

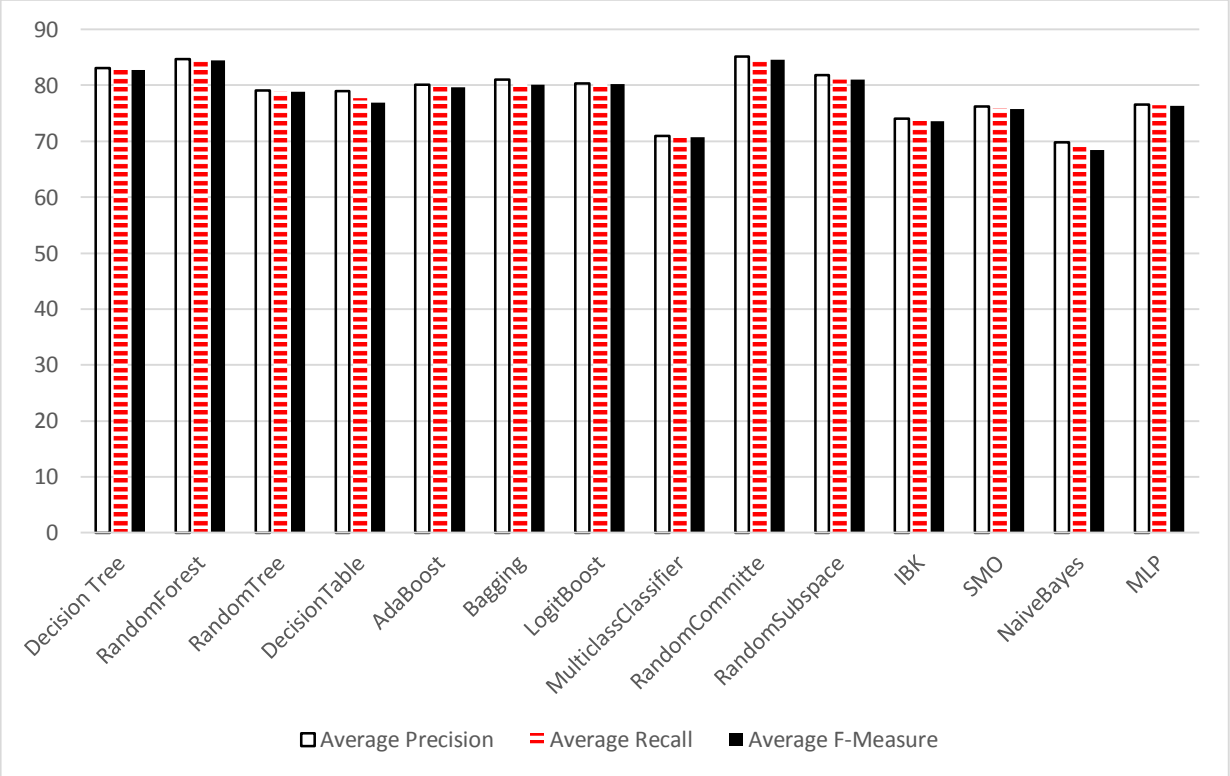

Fig. 2. Performance of different classifiers using all features

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

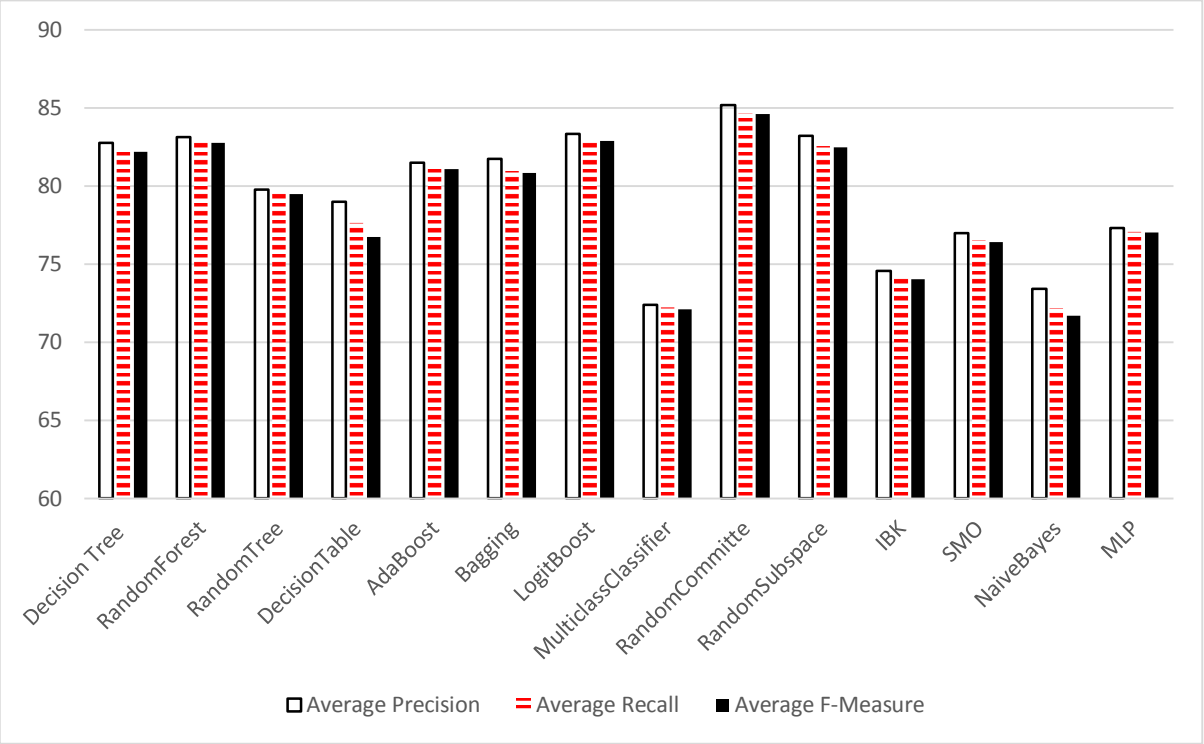

Fig. 3. Performance of different classifiers using correlation feature selection

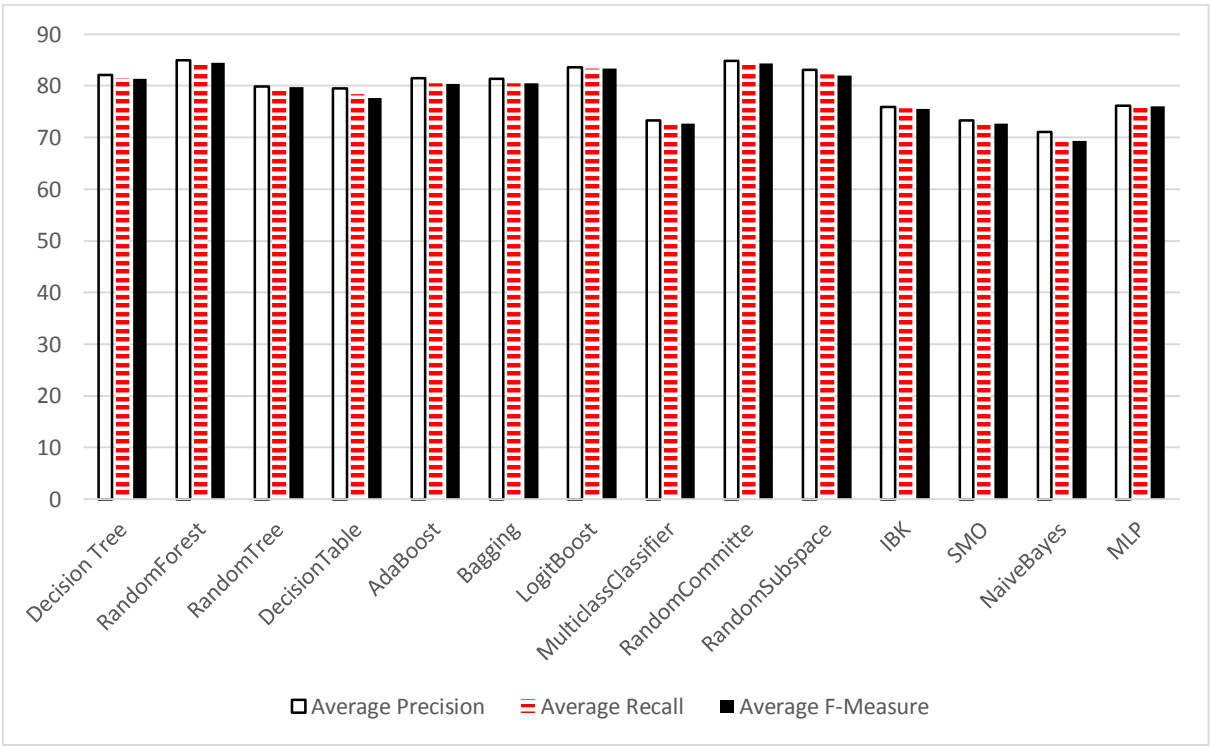

Fig. 4. Performance of different classifiers using Infogain feature selection

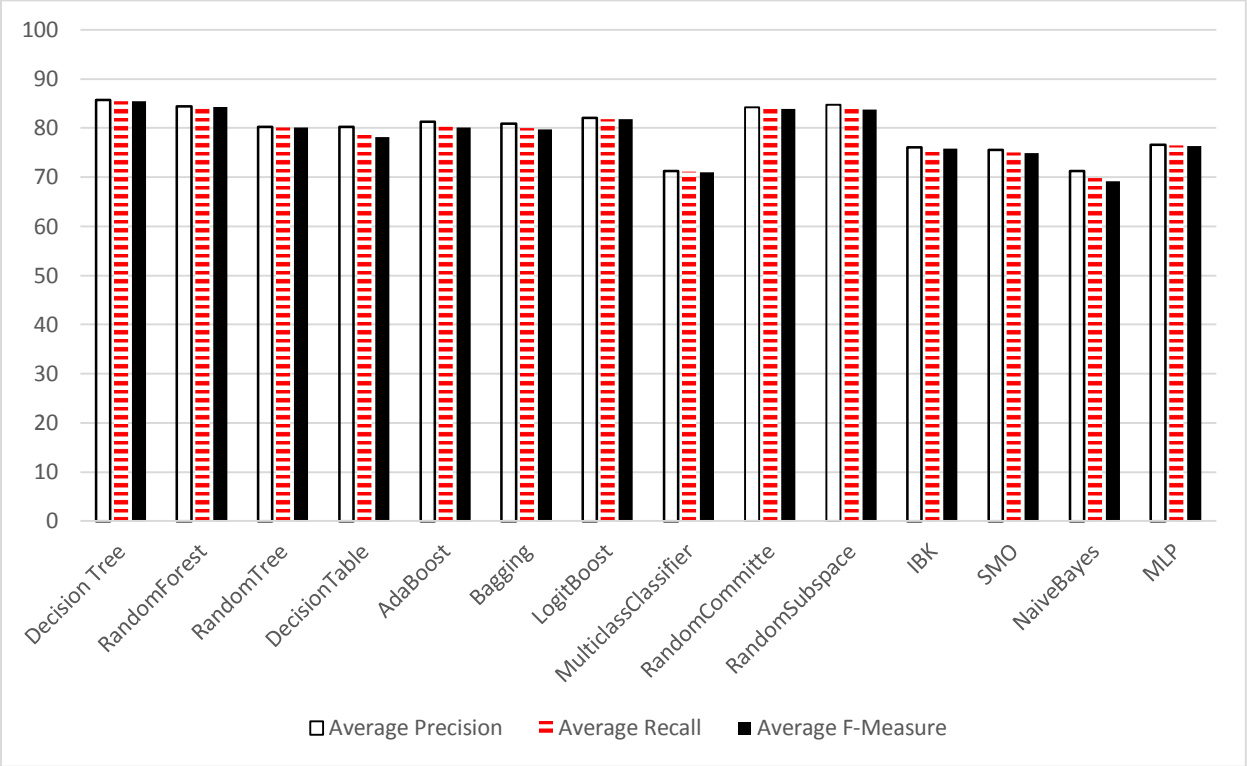

Fig. 5. Performance of different classifiers using ReliefF feature selection

6. Conclusion

After detecting clones from software, clones suitable for removal through refactoring must be filtered. The manual selection of suitable clones can be tiring and time-intensive. This study addresses this challenge by systematically evaluating the effectiveness of various machine learning algorithms in combination with three feature selection methods- Correlation, Infogain and ReliefF for identifying suitable clones for refactoring.

A total of fourteen machine-learning classifiers were analyzed across six open-source Java projects. The results demonstrate that **Decision Tree** and **LogitBoost** classifiers achieved the highest accuracy of 94.44% on Lucene dataset with feature selection. Furthermore, Decision Tree when used with ReliefF, gives the highest average precision, recall, and F-measure across datasets, underscoring the significant impact of applying suitable feature selection techniques.

The findings not only validate the use of automated machine learning approaches in clone management but also offer a scalable framework for real-world software maintenance. This contributes both scientifically by introducing a comparative evaluation methodology and practically by reducing manual effort in clone filtering.

The results lead to the following general recommendations:

ReliefF is highly effective when used with Decision Tree models in clone recommendations for refactoring. Infogain tends to benefit classifiers with structured feature dependencies whereas Correlation proves useful when datasets are less complex but require basic relevance filtering. Classifiers such as Random Forest, Random Committee and Decision Tree are consistently reliable across various datasets making them suitable for clone refactoring scenarios.

In the future, we aim to explore additional feature selection methods and a broader set of machine learning algorithms on a clone refactoring dataset that incorporates a cloned fragment's bug-proneness and developer's effort. Further, the applicability of deep learning techniques for semantic feature extraction to build clone recommendation systems for refactoring can be explored in future work.

Contribution of authors: Introduction, methodology – **Manpreet Kaur**; related work– **Dhavleesh Rattan**; performance evaluation – **Manpreet Kaur**; Tables, Figures, Discussion – **Madan Lal**; original draft preparation and editing – **Manpreet Kaur**; review- **Dhavleesh Rattan** and **Madan Lal**

Conflict of Interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Financing

This study was conducted without any financial support.

Data Availability

The manuscript contains no associated data.

Use of Artificial Intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

All the authors have read and agreed to the published version of this manuscript.

References

1. Rattan, D., Bhatia, R., & Singh, M. Software clone detection: A systematic review. *Information and Software Technology*, 2013, vol. 55, no. 7, pp. 1165–1199. DOI: 10.1016/j.infsof.2013.01.008.

2. Roy, C. K., Cordy, J. R., & Koschke, R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 2009, vol. 74, no. 7, pp. 470–495. DOI: 10.1016/j.scico.2009.02.007.

3. Roy, C. K., Zibran, M. F., & Koschke, R. The Vision of Software Clone Management: Past, Present, and Future. *Proceedings of the IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering*, 2014, pp. 18–33. DOI: 10.1109/CSMR-WCRE.2014.6747168.

4. Sheneamer, A., & Kalita, J. A Survey of Software Clone Detection Techniques. *International Journal of Computer Applications*, 2016, vol. 137, no. 10, pp. 1–21. DOI: 10.5120/ijca2016908896.

5. Zibran, M. F. Analysis and visualization for clone refactoring. *Proceedings of the 2015 IEEE 9th International Workshop on Software Clones (IWSC)*, 2015, pp. 47–48. DOI: 10.1109/IWSC.2015.7069889.

6. Tairas, R., & Gray, J. Clone maintenance through analysis and refactoring. *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2008, pp. 29–32. DOI: 10.1145/1496653.1496661.

7. Mondal, M., Roy, C. K., & Schneider, K. A. A survey on clone refactoring and tracking. *Journal of Systems and Software*, 2020, vol. 159, article no. 110429. DOI: 10.1016/j.jss.2019.110429.

8. Duala-Ekoko, E., & Robillard, M. P. Clone tracker: Tool support for code clone management. *Proceedings of the International Conference on Software Engineering*, 2008, pp. 843–846. DOI: 10.1145/1368088.1368218.

9. Kaur, M., Rattan, D., & Lal, M. An Approach To Recommend Clones For Refactoring Using Machine Learning And Feature Selection. *IOSR Journal of Computer Engineering*, 2023, vol. 25, no. 6, pp. 62–64. DOI: 10.9790/0661-2506016264.

10. Chen, Z., Kwon, Y. W., & Song, M. Clone refactoring inspection by summarizing clone refactorings and detecting inconsistent changes during software evolution. *Journal of Software: Evolution and Process*, 2018, vol. 30, no. 1, pp. 1–24. DOI: 10.1002/smr.1951.

11. Alharbi, M. A comparative study of automated refactoring tools. *IEEE Access*, 2024, vol. 12, pp. 18764–18781. DOI: 10.1109/ACCESS.2024.3361314.

12. Alomar, E. A., & Mkaouer, M. W. Behind the intent of extract method refactoring. *IEEE Transactions on Software Engineering*, 2024, vol. 50, no. 1, pp. 668–694. DOI: 10.1109/TSE.2023.3345800.

13. Kalhor, S., Keyvanpour, M. R., & Salajegheh, A. A systematic review of refactoring opportunities by software antipattern detection. *Automated Software Engineering*, 2024, vol. 31, no. 1, article no. 42. DOI: 10.1007/s10515-024-00443-y.

14. Higo, Y., Kamiya, T., Kusumoto, S., & Inoue, K. Refactoring Support Based on Code Clone Analysis. *Proceedings of the Product Focused Software Process Improvement, 5th International Conference (PROFES 2004), Kansai Science City, Japan*, 2004, pp. 220–233. DOI: 10.1007/978-3-540-24659-6_16.

15. Higo, Y., Kamiya, T., Kusumoto, S., & Inoue, K. ARIES: Refactoring support tool for code clone. *Proceedings of the International Conference on Software Engineering*, 2005, pp. 53–56. DOI: 10.1145/1083292.1083306.

16. Higo, Y., Kusumoto, S., & Inoue, K. A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system. *Journal of Software Maintenance and Evolution: Research and Practice*, 2008, vol. 20, no. 6, pp. 435–461. DOI: 10.1002/smr.394.

17. Schulze, S., Kuhlemann, M., & Rosenmüller, M. Towards a refactoring guideline using code clone classification. *Proceedings of the ACM International Conference*, 2009, pp. 1–4. DOI: 10.1145/1636642.1636648.

18. Choi, E., Yoshida, N., Ishio, T., Inoue, K., & Sano, T. Extracting code clones for refactoring using combinations of clone metrics. *Proceedings of the International Workshop on Software Clones (IWCS)*, 2011, pp. 7–13. DOI: 10.1145/1985404.1985407.

19. Mondal, M., Roy, C. K., & Schneider, K. A. Automatic identification of important clones for refactoring and tracking. *Proceedings of the 2014 IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)*, 2014, pp. 11–20. DOI: 10.1109/SCAM.2014.11.

20. Wang, W., & Godfrey, M. W. Recommending clones for refactoring using design, context, and history. *Proceedings of the 4th IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 331–340. DOI: 10.1109/ICSME.2014.55.

21. Rongrong, S., Liping, Z., & Fengrong, Z. A Method for Identifying and Recommending Reconstructed Clones. *Proceedings of the 2019 International Conference on Management Engineering, Software Engineering and Service Sciences (ICMESS)*, 2019, pp. 39–44.

22. Sheneamer, A. M. An Automatic Advisor for Refactoring Software Clones Based on Machine Learning. *IEEE Access*, 2020, vol. 8, pp. 124978–124988. DOI: 10.1109/ACCESS.2020.3006178.

23. Yue, R., Gao, Z., Meng, N., Xiong, Y., Wang, X., & Morgenthaler, J. D. Automatic clone recommendation for refactoring based on the present and the past. *Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 115–126. DOI: 10.1109/ICSME.2018.00021.

24. Fanqi, M. Using self-organized mapping to seek refactorable code clone. *Proceedings of the 2014 International Conference on Communication Systems and Network Technologies (CSNT)*, 2014, pp. 851–855. DOI: 10.1109/CSNT.2014.177.

25. Kaur, M., & Rattan, D. A systematic literature review on the use of machine learning in code clone research. *Computer Science Review*, 2023, vol. 47. DOI: 10.1016/j.cosrev.2022.100528.

26. Quradaa, F. H., Shahzad, S., & Almoqbily, R. S. A systematic literature review on the applications of recurrent neural networks in code clone research. *Plos One*, 2024, vol. 19, no. 2, article no. e0296858. DOI: 10.1371/journal.pone.0296858.

27. Idouglid, L., Tkatek, S., Elfayq, K., & Guezzaz, A. A novel anomaly detection model for the industrial internet of things using machine learning techniques. *Radioelectronics and Computer Systems*, 2024, vol. 2024, no. 1, pp. 143–151. DOI: 10.32620/reks.2024.1.12.

Received 27.09.2024, Accepted 25.08.2025

ЕМПІРИЧНА ОЦІНКА ВИБОРУ ФУНКЦІЙ ТА ТЕХНІК МАШИННОГО НАВЧАННЯ, ЩОБ РЕКОМЕНДУВАТИ КЛОНИ ДЛЯ РЕФАКТОРИНГУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Манпріт Каур, Дхавліш Ратан, Мадан Лал

Предметом статті є управлінння програмними клонованими фрагментами. Програмні клони – це дублікати кодових фрагментів, які можуть існувати в одному або різних файлах програмного забезпечення. Виявлення та управління програмними клонами стало добре встановленою областю досліджень. Програмні клони повинні бути керовані для мінімізації їх негативних впливів, оскільки наявність клонів може збільшити витрати на обслуговування та вимоги до ресурсів програмного забезпечення. Рефакторинг є поширеною технікою управління клонами. Інструмент для виявлення програмних клонів може виявити багато клонів у програмному забезпеченні, однак не всі виявлені клони можуть бути придатними для рефакторингу. Розробнику потрібен підмножина виявлених клонів, які можна легко відрефакторити.Метою цього дослідження є пропозиція програмних клонів для рефакторингу з використанням методів машинного навчання. У статті оцінюється продуктивність чотирнадцяти алгоритмів машинного навчання та досліджується вплив трьох методів відбору ознак на точність рекомендацій клонів. Завдання, які потрібно вирішити, це: вибрати відповідні ознаки з наборів даних, розробити моделі на основі машинного навчання, які можуть пропонувати підходящі клони для рефакторингу, обрати ефективний алгоритм машинного навчання та відбору ознак для рекомендації клонів для рефакторингу. Методи, які використовуються для відбору ознак, включають кореляцію, InfoGain та ReliefF. Дослідження проводиться на наборах даних з шести відкритих програмних продуктів, написаних на Java. Експериментальні результати показують, що класифікатори Дерево Рішень та LogitBoost досягають найвищої точності 94,44% на датасеті Lucene. Серед методів відбору ознак, ReliefF забезпечує найкращу продуктивність, особливо коли використовується з алгоритмом Дерево Рішень. Це дослідження робить висновок, що Випадковий Комітет, Випадковий Ліс та Дерево Рішень показують найкращі результати у поєднанні з кореляцією, InfoGain та ReliefF відповідно. Загалом, класифікатор Дерево Рішень, комбінований із методом відбору ознак ReliefF, забезпечує найвищу середню точність, відгук та F-міру на різних датасетах.

Ключові слова: клони програмного забезпечення; управління клонами; рекомендація щодо клонування, рефакторинг клонування, вибір функцій, машинне навчання.

Манпрет Каур – доктор філософії, доцент кафедри комп’ютерних наук та інженерії, Інженерний коледж Баба Банда Сінгх Бахадур, Фатегарх Сахіб, Пенджаб, Індія.

Дхавліш Раттан – доктор філософії, доцент кафедри комп’ютерних наук та інженерії Пенджабського університету, Патіала, Пенджаб, Індія.

Мадан Лал – доктор філософії, доцент кафедри комп’ютерних наук та інженерії Пенджабського університету, Патіала, Пенджаб, Індія.

Manpreet Kaur – PhD, Assistant Professor, Department of Computer Science and Engineering, Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, Punjab, India,
e-mail: manpreet.kaur09@gmail.com; ORCID: 0000-0002-6880-0480.

Dhavleesh Rattan – PhD, Assistant Professor, Department of Computer Science and Engineering, Punjabi University, Patiala, Punjab, India, e-mail: dhavleesh.ce@pbi.ac.in, ORCID: 0000-0002-6295-5078.

Madan Lal – PhD, Assistant Professor, Department of Computer Science and Engineering, Punjabi University, Patiala, Punjab, India, e-mail: madanlal@pbi.ac.in, ORCID: 0000-0001-6202-4399.