A Novel Approach for Code Smell Detection: An Empirical Study

Seema Dewangan¹ Rajwant Singh Rao¹, Alok Mishra² AND Manjari Gupta³

¹Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya, Bilaspur, India
²Faculty of Logistics, Molde University, College (Specialized University in Logistics), 6410 Molde, Norway
³(Computer Science), DST - Centre for Interdisciplinary Mathematical Sciences, Institute of Science, Banaras Hindu University, Varanasi, India

Corresponding author: Alok Mishra (e-mail: alok.mishra@himolde.no).
This research received no external funding.

ABSTRACT Code smells detection helps in improving understandability and maintainability of software while reducing the chances of system failure. In this study, six machine learning algorithms have been applied to predict code smells. For this purpose, four code smell datasets (God-class, Data-class, Feature-envy, and Long-method) are considered which are generated from 74 open-source systems. To evaluate the performance of machine learning algorithms on these code smell datasets, 10-fold cross validation technique is applied that predicts the model by partitioning the original dataset into a training set to train the model and test set to evaluate it. Two feature selection techniques are applied to enhance our prediction accuracy. The Chi-squared and Wrapper-based feature selection techniques are used to improve the accuracy of total six machine learning methods by choosing the top metrics in each dataset. Results obtained by applying these two feature selection techniques are compared. To improve the accuracy of these algorithms, grid search-based parameter optimization technique is applied. In this study, 100% accuracy was obtained for the Long-method dataset by using the Logistic Regression algorithm with all features while the worst performance 95.20% was obtained by Naive Bayes algorithm for the Long-method dataset using the chi-square feature selection technique.

INDEX TERMS Code smell, Code smell detection, Machine learning techniques, feature selection, parameter optimization

I. INTRODUCTION

Now days the complexity of the software is increasing continuously due to complex requirements, increased number and size of modules, and code smells in the developed software etc. Complex requirements are difficult to analyze and understand and thus development becomes difficult. Understanding complex software is also difficult and thus maintainability of complex software becomes low. Complex requirements are not in developers hand but code smells can be detected and refactored to make the software more simple, understandable, easy to develop and maintain [1]. In the software development process, functional and nonfunctional values both are essential for designers to follow the guaranteed software quality [2]. Generally functional requirements are only emphasized by developers whereas nonfunctional requirements, for example comprehensibility, verifiability, evolution, maintainability and reusability are neglected [3]. The lack of nonfunctional quality leads to decline in the quality of the software, so that the complexity and maintenance work of software increases. Fowler et al. [4] explained the refactory technique by which the loosely implemented code can be converted into a good implementation. They proposed definitions of 22 code smells.

The effects of code smells on software have been examined by various studies and revealed their undesirable effect on the quality of the software [19]–[23]. The effects of removing code smells in improving the possibility of software system failures and faults are also examined by them. They explored the challenges that code smells have negative effects on the software development process and recommended refactoring the software to remove them.

The influences of code smells on software are analyzed by different researchers; Olbrich et al. [24]–[25], Khomh et
al. [26], Deligiannis et al. [27] by inspecting the frequency and size of changes in the software system. It is also examined by them that the classes, infected by code smells have a higher rate of changes and needs more maintenance work. Li et al. [29] studied the role of bad smells and class error probability in the object-oriented software system. Their study results showed that the infected software components using code smells have higher changes of class errors than other components. Castillo et al. [28] studied the harmful effect of god class on power consumption and that eliminating God Class smells reduce the cyclomatic complication of the source code. Thus code smell detection will indirectly help in reducing all these problems of costly maintenance and chances of failure of software systems etc.

There are lot of challenges in code smell detection for software developers. Different types of code smells make this process difficult. Not a formal definition of code smells is another challenge.

In this study, a framework is built for code smell prediction using machine learning algorithms and software metrics. In a code smell prediction, metrics play an important role in measuring the functional and nonfunctional qualities of software and understanding the features of the source code. Static information of the software is given by metrics, such as number of methods, classes, and parameters along with the measure of coupling and cohesion among objects in the system.

This paper proposes code smell prediction and its analysis using six machine learning algorithms. For this purpose, four code smell dataset (God Class, Data-Class, Feature-Envy, and Long-Method) from Fontana et al. [18] are used that are generated from 74 open source systems. The first two datasets belong to class-level and the remaining two datasets belong to method-level. Six machine learning algorithms (Naive Bayes, KNN, Multilayer Perceptron, Decision Tree, Logistic Regression and Random Forest) are applied on the datasets and achieved the highest accuracy in the Logistic Regression algorithm (100%).

Various experimental studies, tools, and techniques have been developed by researchers to detect code smells and they found different results. Various reviews and comparative studies [30-32] represent that there are several reasons behind it including the problem to discover the formal definitions for code smells. Additionally, given tools and techniques are not identifying all code smells since they focus on some code smells.

The main contribution of this paper is two-fold: In the first step the six machine learning methods for detecting the code smell are proposed. In the second step, performance measurement (Accuracy, Precision, Recall, and F-1 Score), achieved by applying machine learning methods and their evaluation with grid search cross-validation technique are shown.

The following research questions are answered and discussed in this research study.

RQ1. What is the effectiveness of applying machine learning techniques to explain the code smell detection problem and which software metrics play important role in predicting code smells?

Motivation. To study and examine the impact of machine learning methods to build a code smell prediction framework that can predict code smells in object-oriented software using software metrics.

RQ2. Does a feature selection method affect the performance of the prediction?

Motivation. To study the impact of feature selection technique to improve the model’s accuracy and to identify which software metrics contribute an important role in the code smell prediction process.

RQ3. Whether methods used for separating dataset into training set and test set affect accuracy or not?

Motivation. To study the impact of 10-fold cross validation, to set the training set and test in each dataset, on the performance accuracy.

RQ4. Does the parameter optimization technique affect the accuracy of the prediction?

Motivation. To study the impact of tuning the machine learning techniques’ parameters on performance accuracy.

The outline of the paper is as follows; the section 2 explains related works, which briefly describes code smell detection using machine learning techniques. Section 3 reports the datasets related information, approaches and research framework used in this work. Section 4 illustrates the experimental results. Threats to validity is discussed in section 5 and the final section 6, provides the conclusion of our work.

II. RELATED WORK

Many researchers have studied machine learning techniques applications to detect the code smells. This section represents different types of tools and techniques, and different types of machine learning techniques applied on code smell datasets to detect the code smells. In the literature, different tools and techniques are proposed to detect code smells. Each tool and technique produces different results. Kessentini et al.[17] classified the code smells detection techniques into seven categories such as manual approaches, symptom-based approaches, metrics-based approaches [13]-[15], probabilistic approaches [12], visualization-based approaches [8], search-based approaches [9]-[11], and cooperative-based approaches [7]. Inspection methods [5], process identification, and manufacturing process methods [6] are used to improve software quality by manual approach. The specification algorithm [16] is used to detect code smell by symptom-based approach.

Travassos et al. [5] proposed Inspection methods and Ciupke [6] presented process identification, and manufacturing process methods to improve software quality by manual approach. The specification algorithm [16] used
to detect code smell by symptom-based approach. Marinescu [13] proposed a technique called "detection strategy" for devising metrics-based rules that capture deviations from good design principles and heuristics. Moha et al. [14] and Tsantalis et al. [15] also introduced metrics-based approaches. Rao et al. [12] proposed a probabilistic approach to detect bad smells in object-oriented design.

A novel smell indicator known as Stench Blossom is proposed by Hill et al. [8], it represents a cooperative ambient conception designed to first give programmers a fast, high-level overview of the smells in their code, and then, if they wish, to help in understanding the sources of those code smells. Abdelmoez et al. [7] recommended risk-based approach to detect code smells. And search-based approach presented by [9]-[11] for code smell detection. A comparing and experimenting machine learning algorithm for code smell detection is suggested by Fontana et al. [18]. They experimented 16 machine learning algorithms on four code smells dataset and 74 Java systems that are manually validated instances on the training dataset. Additionally, boosting techniques are applied on four code smell datasets. The code smell severity classification using machine learning algorithm is proposed by Fontana et al. [33]. This technique can help software designers to prioritize the classes or methods. Multinomial classification and regression method is used to classify the code smell severity.

Assessment of Code Smell for Predicting Class Change Proneness Using Machine Learning techniques is presented by Pritam et al. [42]. They approve the effect of code smells on the change inclination of a specific class in a product framework. They applied six machine learning algorithms to predict the change proneness using code smell from a set of 8200 java classes spanning 14 software systems. Kacem et al. [43] suggested an advanced method to detect code smells by using deep-learning techniques. They applied hybrid detection approach based on deep Auto-encoder and artificial neural network algorithm on four code smell datasets that is extracted from 74 open-source systems.

Pushpalatha et al. [34] recommended ensemble methods approach to predict the severity of closed source bug reports. They applied two preprocessing techniques, i.e., Information gain and Chi-square for data reduction and they observed information gain gives slightly good accuracies over chi-square. They found bagging gives better accuracy than other ensemble algorithms. They obtained the accuracies of various ensemble approaches after reducing dimensionality using Information gain. The accuracy for, PitsA varies between 58.09 and 74.07, for PitsB varies between 54.38 and 80.72, for PitsC varies between 79.85 and 89.80, for PitsD varies between 93.42 and 96.20, for PitsE varies between 69.21 and 72.36, for PitsF varies between 64.10 and 75.70 using different ensemble methods.

Mhawish et al. [35] proposed Machine learning approach to detect code smells from software and they observed the metrics that play critical roles in the detection process. They applied genetic algorithm based two feature selection techniques and parameter optimization technique based on a grid search. They obtained best accuracy in predicting the Data Class, God Class, and Long Method smells by 98.05%, 97.56%, and 94.31% respectively using GA_CFS method, and in the Long Method scored the best accuracy of 98.38% using GA-Naive Bayes feature selection method.

Mhawish et al. [36] suggested code smells analysis of Predictions using machine learning techniques and software metrics. They also applied genetic algorithm-based feature selection methods to enhance the accuracy of these machine learning algorithms by selecting the best features in each dataset. Moreover, they applied parameter optimization techniques based on the grid search algorithm which is enhance the accuracy of all these algorithms. They noted in Random Forest model achieves the best accuracy of 99.71% and 99.70% in predicting the Data Class in the ORI_D and REFD _D datasets respectively.

Guggulothu et al. [37–38] considered code smell detection using multi-label classification approach. They used multi-label classification method to detect whether the given code elements are affected in multiple smells. They applied an unsupervised classification technique for finding good accuracy. Guggulothu et al.[37] obtained 99.10% highest accuracy using B-J48 pruned algorithm for Feature-envy dataset, and 95.90% highest accuracy using Random forest algorithm for Long-method dataset.

Gupta et al. [39] recommended prediction of code smells using feature extraction from source code on eight types of code smells. They present the application of data sampling technique to handle the class imbalance problem and uses feature selection technique to find most relevant features sets. They applied deep learning technique and improved 88.47 to 96.84% AUC accuracy.

Kaur et al. [40] presented Ensemble learning technique and correlation feature selection technique on three open-source java datasets for detection of the code smell. They applied Bagging and Random forest classifier to analyze each approach with four performance measurements like accuracy (P1), G-mean 1(P2), G-mean2 (P3), and F-measure (P4).

Draz et. al. [41] suggested search-based technique to improve the code smell prediction using Whale optimization algorithm as a classifier. They perform their experiment on five open-source software projects and found the nine types of code smells. They obtained average of 94.24% precision and 93.4% recall.

Azeem et.al.[63] described a Systematic Literature Review (SLR) on the utilization of machine learning techniques for code smell detection. They targeted four aspects related to previous research work on code smell detection techniques. These are (a) which code smells have been detected (b)
what machine learning setup has been adopted (c) what kind of evaluation strategies have been exploited, and (d) what are the claimed performances of proposed ML techniques. They have found that the decision tree and support vector machine are the most common machine learning techniques used for code smell detection. On the other hand, JRip and Random forest algorithms are the most effective classification techniques.

F. Pecorelli et al. [64] proposed a significant study to compare the performance of heuristic-based and machine-learning-based techniques for metric-based code smell detection. They considered five types of code smells (God Class, Spaghetti Code, Class Data Should Be Private, Complex Class, and Long Method) and compare ML techniques with DECOR, a state-of-the-art heuristic-based approach. They found that the DECOR normally obtained better performance than ML baseline.

III. RESEARCH FRAMEWORK

The list of steps followed to build the code smell prediction framework is shown in figure 1. In the first step, code smell datasets are taken from Fontana et al. [18]. Then, preprocessing (Normalization) step is applied to the dataset to cover the different ranges of the datasets and to obtain the best algorithms’ parameters. Then machine learning algorithms on the dataset is trained and their performance are computed. Finally, 10-Fold cross-validation technique is applied to evaluate each experiment performance during the training process and Grid search algorithm applied to enhance the accuracy.

A. REFERENCE DATASETS

In this study four code smell datasets (God class, Data-class, Feature-envy, and Long-method) from Fontana et al. [18] are used to build the code smell detection framework. The data preparation methodologies of Fontana et al. [18] are briefly explained in following subsections. These datasets are available at http://essere.disco.unimib.it/reverse/MLCSD.html.

B. DATASET SELECTION AND REPRESENTATION

Qualitus Corpus Software System is compiled by Tempero et al. [44] and analyzed by Fontana et al. [18]. In the corpus software out of 111 systems, 74 systems are considered and remaining 37 systems could not be used to detect code smells since these did not comply correctly. For the available 74 software systems, Fontana et al. [18] computed 61 software metrics for the class level code smells- Data Class and God Class and 82 software metrics for the method level code smells- Feature Envy and Long Method.

Fontana et al. [18] used several detection tools and techniques to detect code smells called advisors: iPlasma (God Class, Brain Class), Anti-pattern Scanner [45], PMD [46], iPlasma, Fluid Tool [47], and Marinescu detection rules [48]. Table 1 shows the automatic detection tools. They filtered and relabeled results manually with the help of 3 students of Master’s degree. Each dataset contains 140 smells and 280 no-smell.

| Code Smell       | Reference, Tool/Detection rules |
|------------------|---------------------------------|
| God Class        | iPlasma (God Class, Brain Class) [46], PMD |
| Data Class       | iPlasma, Fluid Tool [47], Anti-Pattern Scanner [45] |
| Feature Envy     | iPlasma, Fluid Tool |
| Long Method      | iPlasma (Brain Method), PMD, Marinescu detection rule [48] |

As shown in Table 2, 61 software metrics are calculated for Data class and God class code smells at class level. 82 software metrics are calculated for long method and feature envy code smells at method level. Detailed descriptions of these features are available in Fontana et al. [18].

| Code Smell | Samples | Features |
|------------|---------|----------|
| Class Level | Data Class | 420 | 61 |
|            | God Class  | 420 | 61 |
| Method Level | Feature Envy | 420 | 82 |
|            | Long Method | 420 | 82 |

Definitions of the code smells on which study is based as following:

- **God class** describes a large class that have many lines of code, functions, or fields. The God Class is considered the most complicated code smells for many reasons, operations and functions that occur there. It causes issues associated with size, coupling, and complexity [49].

- **Data Class** points to the class used to store the data which are used by other classes. Data Class covers only
fields and accessor methods (getters/setters) without any behavior methods or complex functionalities. Because of it create problems related to data abstraction and encapsulation [49].

Feature Envy is the method level smell that accesses data or use operations that belong to different classes as compared to its own class i.e., it admits additional foreign data in comparison the local one. and therefore, it creates problems associated to the strength of coupling [49].

Long Method is method level smell that points to big-sized method, because of the heavy size of code lines and the functionalities, it is applied within the method. It enhances the functional complication of the method and it will be difficult to understand and therefore it creates problems related to the strength of understanding the operations in methods [49].

C. FEATURE SCALING
Sometimes datasets have different ranges of features, so machine learning or classification algorithms cannot be directly applied to the dataset. Therefore, Feature scaling is necessary to cover the different range of the dataset. Sometimes, several machine learning algorithms can converge faster on the feature scaled dataset and when the model is sensitive to size, it will have a greater impact. For example, before applying the support vector machine algorithm, avoiding standardization id is essential for supremacy of advanced numerical ranges on slight numerical ranges where the range of possible high values cause mathematical problems [50].

In this paper, Min-Max normalization technique is applied to convert feature values of datasets between 0 − 1. This process is used in the data preprocessing stage in which the data are prepared to be used later by one of the machine learning technique like Support vector machine, Neural network etc. [51]. Equation 1 shifts and rescales the values of a feature (X) so that they end up ranging between 0 and 1.

\[ X' = \frac{X - X_{min}}{X_{max} - X_{min}} \]  

(1)

In the equation 1, X is an original value and X' is the normalized value. X' will be between 0 and 1. Min-Max normalization is applied to all the datasets and the new reformed data received in the form of 0 to 1 have been taken as input into all machine learning techniques.

D. FEATURE SELECTION TECHNIQUE
Feature selection is a method designed to obtain the maximum impact features in the dataset so it can advance performance by improving awareness of the software metrics that play a significant role in distinguishing between similar roles in design patterns [55]. In this study, two feature selection techniques are applied to find most important features from each dataset: Chi-square and Wrapper based feature selection technique.

The chi-square based feature selection technique is used for categorical features in a dataset. Chi-square is calculated between each metrics and select the desired number of metrics with best chi-square scores. Best metrics are selected in each code smell datasets using chi-square based feature selection technique.

The Wrapper based feature selection technique is used to select a subset of features or variables in the dataset so that these features are most relevant to the predicted target value [60]. Fig. 2 shows the working of wrapper-based feature selection method.

FIGURE 2. Wrapper based feature selection method [61].

E. PARAMETER OPTIMIZATION
Every learning approach in machine learning, has one set of parameters that affect the learning procedure and algorithm performance. In addition, each parameter is different in type and domains. The best parameter for each algorithm is different, depending on training data set. To determine the correct parameters’ values, different combination of parameter values for each algorithm should be tested; so that the prediction model can correctly predict the test data set [36].

Grid search algorithm is applied in this work to find the optimum values of the parameters of each algorithm. The grid search approach is an optimization algorithm which is used to obtain the best hyper parameter values from the parameter values set provided [52]. Grid search is essential to find the optimal hyper parameters of a model, which results in the most accurate predictions. It is based on comprehensive search for the combination of parameters that yields the best performance value in the prediction model [53]. 10-fold cross-validation is used to prevent it from overfitting of the model on training data. It is used to measure the algorithm performance with each possible combination of the parameters.

In parameter optimization based on a grid search algorithm, a set of values for each parameter are recognized. For nominal parameters, the nominal values are assigned as shown in Table 3. The value range as well as number of steps to the numeric parameters are also assigned. The values that must be tested are assigned within upper and lower bounds of the range based on the specified number of steps that are assigned for each parameter, as shown in Table 4.


F. VALIDATION METHODOLOGY
In this study, validation techniques are applied to evaluate each experiment performance. During the training process 10-fold cross-validation technique is applied. Machine learning models are evaluated by using 10-fold cross-validation that split the dataset into 10 portions with 10 times of repetition of training the model. In each repetition, one part of the dataset is considered as a test dataset, and other parts of the dataset are taken for training purpose. Then, finally the trained models are tested with unseen test dataset (20% split from the dataset before training). The following fig. 3 shows the general k-fold cross-validation evaluation technique.

G. PERFORMANCE MEASURES
In this study, a set of experiments have been performed. To measure the performance of machine learning algorithms, four performance parameters: Precision, F-measure, Recall and Accuracy are considered. To calculate them, true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are used. TP shows the detected instances where the model correctly predicted positive class. False positive (FP) shows the detected instances where the model wrongly predicts the positive class. True negative (TN) shows the detected instances where the model correctly predicts the negative class. False negative (FN) shows the detected instances where the model wrongly predicts the negative class. These parameters are calculated using a confusion matrix that contains the actual and predicted information recognized by design pattern detection classifiers [54]. A brief definition and equations of the performance parameters which are used to measure the performance of design pattern prediction model are given below:

- **Precision**: The precision measures the percentage of correctly identified code smell instances by the machine learning model. The following equation 2 is used to calculate precision value. Precision is measured as the number of true positives divided by the total number of true positives and false positives. The precision result value is 0.0 for no Precision and 1.0 for perfect Precision.

  \[
  \text{Precision} = \frac{TP}{TP+FP} \quad \text{(2)}
  \]

- **Recall**: The recall measures the code smell instances identified by machine learning model. The formula of recall value is shown in equation 3. Recall is measured as the number of true positives divided by the total number of true positives and false negatives. The recall result value is 0.0 for no recall and 1.0 for perfect recall.

  \[
  \text{Recall} = \frac{TP}{TP+FN} \quad \text{(3)}
  \]

- **F1-Score**: F1-score is a combination of precision (P) and recall (R). It is a harmonic mean of precision and recall metrics, and it represents a balance between their values. The formula of F1-score is shown in equation 4. The F1-score result value is 0.0 for no F1-score and 1.0 for best F1-score.

  \[
  \text{F1-score} = 2 \times \frac{P \times R}{P + R} \quad \text{(4)}
  \]
• **Accuracy**: in the machine learning technique, accuracy is an important performance measurement. It represents the percentage of correctly classified instances in the positive and negative classes. The accuracy is defined to the relationship between precision and recall. Ideally, a rational approach should be taken in decent precision and recall rate, i.e., while recall values improve the precision values should remain high. Thus, it is concluded that a suitable method should have a high rate of true positives with a low rate of false positives and false negatives [34]. Equation 5 shows the formula of accuracy. The accuracy is 0.0 for worst performance and 1.0 for best performance.

\[
\text{Accuracy (A)} = \frac{TP+TN}{TP+TN+FP+FN}
\]  

(5)

### IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

The six machine learning techniques and the results obtained from the experiments are shown in following sub sections. The experimental results obtained from each machine learning technique are shown in the form of table and bar chart.

#### A. NAIVE BAYES

Naive Bayes is a simple machine learning algorithm, it uses Bayes rules, and strongly assumes that given classes, the conditions are conditionally independent. Although this independence assumption is often violated in practice. Naive Bayes often provides competitive classification accuracy. Together with its computational efficiency and many other desirable features, this has led to the wide application of Naive Bayes in practice [56].

Experimental results obtained by applying the Naive Bayes algorithm to the four code smell datasets is shown in Table 5. The comparative performance among all the code smell datasets are shown in Fig. 4 in the form of bar chart. In this experiment, it is found that Naive Bayes algorithm is obtained highest accuracy (96.80%) for God Class dataset.

| Data Class | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) |
|------------|--------------|-----------|-------------|--------------|
| God Class  | 97.00        | 93.00     | 96.00       | 96.80        |
| Feature-Envy | 95.00    | 95.00     | 95.00       | 92.30        |
| Long-Method | 97.00       | 99.00     | 98.00       | 95.20        |

**TABLE 5**

PREDICTION RESULTS OF NAIVE-BAYES ALGORITHM.

#### B. K-NEAREST NEIGHBOUR (KNN)

The K Nearest Neighbor (KNN) algorithm is a supervised ML procedure that can be applied for classification and regression prediction problems. However, it is mostly used for classification prediction problems in industry. KNN algorithm uses 'feature similarity' to predict the value of a new data point, which also means that the new data point will be assigned a value based on how closely the new data point corresponding to training point [57].

Experimental results obtained by applying the KNN algorithm to the four code smell datasets are shown in Table 6. The comparative performance among all the code smell datasets is shown in Fig. 5 in the form of bar chart. In this experiment, it is found that KNN algorithm obtained highest prediction accuracy (97.89%) for Long-Method dataset.

| Data Class   | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) |
|--------------|--------------|-----------|-------------|--------------|
| God Class    | 98.00        | 96.00     | 91.00       | 97.04        |
| Feature-Envy | 90.00        | 1.00      | 95.00       | 91.27        |
| Long-Method  | 95.00        | 1.00      | 98.00       | 97.89        |

**TABLE 6**

PREDICTION RESULTS OF KNN

**FIGURE 4.** Comparative performance bar chart of all datasets with Naive-Bayes technique.

**FIGURE 5.** Comparative performance bar chart of all datasets with KNN technique.
C. MULTILAYER PERCEPTRON (MLP)

Multilayer perceptron (MLP) is flexible machine learning technique that can adapt to complex nonlinear assignments. MLP is the most popular type of neural network, consisting on a feed forward network of processing neurons that are grouped into layers and connected by weighted links [58].

Experimental results obtained by applying the Multilayer perceptron algorithm to the four code smell datasets are shown in Table 7. The comparative performance among all the code smell datasets is shown in Fig. 6 in the form of bar chart. In this experiment, it is found that MLP algorithm obtained highest prediction accuracy (97.62%) for Data Class dataset.

| Method       | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) |
|--------------|---------------|------------|--------------|--------------|
| Data Class   | 95.00         | 95.00      | 95.00        | 97.62        |
| God Class    | 95.00         | 95.00      | 95.00        | 97.58        |
| Feature-Envy | 97.00         | 97.00      | 97.00        | 95.58        |
| Long-Method  | 96.00         | 94.00      | 95.00        | 96.97        |

D. DECISION TREE (DT)

A decision tree is a tree in which each internal or non-leaf node is associated with a decision, and the leaf node is usually associated with an outcome or class label. Each internal node tests one or more attribute values that lead to 2 or more links or branches. Each link in turn is associated with a possible decision value. These links are separate and exhaustive [59].

Experimental results obtained by applying the Decision tree algorithm to the four code smell datasets are shown in Table 8. The comparison performance among all the code smell datasets is shown in Fig. 7 in the form of bar chart. In this experiment, Decision Tree algorithm obtained highest prediction accuracy (98.59%) for Long-Method dataset.

| Method       | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) |
|--------------|---------------|------------|--------------|--------------|
| Data Class   | 99.00         | 99.00      | 99.00        | 98.21        |
| God Class    | 97.00         | 95.00      | 97.00        | 97.09        |
| Feature-Envy | 1.00          | 1.00       | 1.00         | 97.58        |
| Long-Method  | 1.00          | 1.00       | 1.00         | 99.52        |

E. LOGISTIC REGRESSION

Logistic regression is a statistical analysis technique that is applied to predict data values based on past observations of a dataset. This is an important tool in machine learning which predicts the dependent data variables by analyzing the relationship between one or more existing independent variables. It can be used to predict whether software metrics will be found, or not.

Experimental results obtained by applying the logistic regression to the four code smell datasets are shown in Table 9. The comparison performance among all the code smell datasets is shown in Fig. 8 in the form of bar chart. In this experiment, it is found that Logistic Regression algorithm obtained highest accuracy 99.52% and F1 score 100.00% for Long-Method dataset.

| Method       | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) |
|--------------|---------------|------------|--------------|--------------|
| Data Class   | 98.00         | 98.00      | 98.00        | 97.02        |
| God Class    | 98.00         | 99.00      | 98.21        | 98.00        |
| Feature-Envy | 95.00         | 1.00       | 98.00        | 98.59        |

![Figure 6](image6.png) Comparative performance bar chart of all datasets with MLP technique.

![Figure 7](image7.png) Comparison performance bar chart of all datasets with DT technique.

![Figure 8](image8.png) Comparison performance bar chart of all datasets with Logistic regression.
**F. RANDOM FOREST**

Random forest is a machine learning technique used to solve regression and classification problems. It uses ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems. Random forest algorithm builds the result based on the prediction performance among the trees. It can reduce the over fitting of the dataset and improve accuracy. Experimental results obtained by applying the Random forest algorithm to the four code smell datasets are shown in Table 10. The comparison performance among all the code smell datasets is shown in Fig. 9 in the form of bar chart. In this experiment, it is observed that Random Forest algorithm is obtained highest prediction accuracy 99.52% and F1 score 100% for Long-Method dataset.

**TABLE 10**

| Prediction Results for Random Forest |
|-------------------------------------|
| Data Class  | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) |
| God Class   | 1.00          | 1.00       | 1.00         | 98.94        |
| Feature-Envy| 1.00          | 95.00      | 98.00        | 97.58        |
| Long-Method | 1.00          | 1.00       | 1.00         | 99.52        |

**G. PERFORMANCE COMPARISON**

In the earlier subsections, the performance measurements of six machine learning techniques are shown in different tables. In this section, the performance of all these six machine learning techniques (Naive Bayes, KNN, MLP, Decision Tree, Logistic Regression and Random Forest) are compared. The comparison of performances of these six machine learning techniques on the four code smell datasets are shown in Table 11 and their comparison chart are shown in Figure 10. After comparing it is observed that the Random Forest algorithm (Data class- 98.94%, God class- 97.88%, Feature-envy- 97.58%, and Long-method- 99.52%) has got better accuracy then the rest of five algorithms, while the overall worst performance is achieved by Naive-Bayes.

**TABLE 11**

| Data class | God class | Feature-envy | Long-method |
|------------|-----------|--------------|-------------|
| P (%)      | R(%)      | F-1 (%)      | A(%)        | P (%)      | R(%)      | F-1 (%)      | A(%)        | P (%)      | R(%)      | F-1 (%)      | A(%)        |
| Naive Bayes| 85.00     | 92.00        | 87.00       | 84.70      | 97.00     | 93.00       | 95.00       | 96.80      | 95.00     | 95.00       | 95.00       | 92.30     | 97.00       | 99.00       | 98.00       | 98.00       | 95.20       |
| KNN        | 95.00     | 98.00        | 96.00       | 97.38      | 98.00     | 96.00       | 91.00       | 97.04      | 90.00     | 1.00        | 95.00       | 91.27      | 95.00       | 1.00        | 98.00       | 97.89       |
| MLP        | 95.00     | 95.00        | 95.00       | 97.62      | 95.00     | 95.00       | 95.00       | 97.59      | 97.00     | 97.00       | 97.00       | 95.58      | 96.00       | 94.00       | 95.00       | 96.97       |
| DT         | 99.00     | 99.00        | 99.00       | 98.21      | 98.00     | 98.00       | 98.00       | 97.02      | 98.00     | 98.00       | 99.00       | 98.21      | 95.00       | 1.00        | 98.00       | 98.59       |
| LR         | 97.00     | 94.00        | 95.00       | 97.09      | 1.00      | 95.00       | 95.00       | 97.35      | 97.00     | 1.00        | 97.00       | 97.58      | 1.00        | 1.00        | 1.00        | 99.52       |
| RF         | 1.00      | 1.00         | 1.00        | 98.94      | 1.00      | 93.00       | 96.00       | 97.88      | 1.00      | 95.00       | 98.00       | 97.58      | 1.00        | 1.00        | 1.00        | 99.52       |

P- Precision, R- Recall, F1-score, A- Grid search Accuracy, MLP- Multilayer Perceptron, DT- Decision Tree, LR- Logistic Regression, RF- Random Forest
H. IMPACT OF FEATURE SELECTION

This experiment is focused on the study and influences of feature selection methods to improve the model accuracy and recognizing the software metrics that contribute significant role in predicting the code smells. To answer the RQ2, Chi-square and Wrapper-based feature selection techniques are applied. In Table 12, the percentage accuracy and F1-score of all algorithms before and after applying the feature selection technique are compared. The results indicate that Random forest and Logistic regression algorithm performs better when all features are used. On the other hand, the feature selection technique significantly improves the accuracy in some models such as KNN, Naive Bayes and Multilayer perceptron model.

Table 13 shows the selected set of features that are detected by Chi-square feature selection technique for code smell datasets. The best results are obtained by using 10 metrics for the God class and 12 metrics for the Data class, Feature-envy and Long-method are selected. Likewise, Table 14 shows the selected metrics from each datasets that are detected by Wrapper based feature selection technique. Using this method 12 metrics in data class, 9 metrics in the God class, 11 metrics in Feature envy and 7 metrics in Long method are selected.

TABLE 12
EFFECT OF FEATURE SELECTION METHODS ON PREDICTION MODELS

| Code Smell Dataset | Model          | Chi-square Feature Selection | Wrapper based Feature Selection | All Features |
|--------------------|----------------|------------------------------|--------------------------------|--------------|
|                    |                | Accuracy (%) | F1-Score (%) | Accuracy (%) | F1-Score (%) | Accuracy (%) | F1-Score (%) |
| Data Class         | Naive Bayes    | 84.70         | 87.00         | 86.20         | 98.00         | 83.10         | 93.00         |
|                    | KNN            | 97.38         | 96.00         | 97.38         | 96.00         | 97.38         | 91.00         |
|                    | MLP            | 97.62         | 95.00         | 97.35         | 1.00          | 97.28         | 97.00         |
|                    | Decision Tree  | 98.21         | 99.00         | 98.90         | 99.00         | 98.22         | 96.00         |
|                    | Logistic Regression | 97.88     | 96.00         | 97.09         | 95.00         | 98.41         | 98.00         |
|                    | Random Forest  | 98.94         | 96.00         | 98.94         | 1.00          | 99.74         | 91.00         |
| God Class          | Naive Bayes    | 96.80         | 95.00         | 97.40         | 92.00         | 96.60         | 1.00          |
|                    | KNN            | 97.04         | 91.00         | 93.81         | 94.00         | 93.81         | 96.00         |
|                    | MLP            | 97.59         | 95.00         | 96.82         | 90.00         | 96.26         | 95.00         |
|                    | Decision Tree  | 97.02         | 98.00         | 97.02         | 98.00         | 97.02         | 98.00         |
|                    | Logistic Regression | 97.08     | 1.00          | 97.35         | 95.00         | 97.35         | 1.00          |
|                    | Random Forest  | 98.21         | 97.00         | 97.88         | 96.00         | 97.88         | 95.00         |
| Feature Envy       | Naive Bayes    | 92.30         | 95.00         | 95.10         | 93.00         | 92.80         | 95.00         |
|                    | KNN            | 91.27         | 95.00         | 84.85         | 95.00         | 92.50         | 93.00         |
|                    | MLP            | 95.58         | 95.00         | 96.82         | 98.00         | 98.30         | 96.00         |
|                    | Decision Tree  | 98.21         | 99.00         | 98.07         | 1.00          | 98.60         | 98.00         |
|                    | Logistic regression | 97.82     | 97.00         | 97.58         | 97.00         | 98.37         | 95.00         |
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3133810, IEEE Access

| Dataset | No. of metrics (Chi-square) | Metrics selected by Chi-square feature selection technique |
|---------|----------------------------|----------------------------------------------------------|
| Data class | 10 | WOC_type, TCC_type, FANOUT_type, DIT_type, CFNANNM_type, WMCNANNM_type, WMNMC_type, LOCNANNM_type, LOC_type, ATFD_type, RFC_type, AMW_type |
| God class | 12 | LOC_method, NOAV_method, CYCLO_method, ATFD_method, NOAV_method, FANOUT_method, NOAV_method, FANOUT_method, ATFD_method, NOAV_method, ATLD_method, MAXNESTING_method |
| Feature envy | 12 | LOC_method, CYCLO_method, ATFD_method, RFC_type, ATFD_type, CINT_method, NOAV_method, FANOUT_method, NOAV_method, FANOUT_method, ATLD_method, MAXNESTING_method |

I. IMPACT OF 10-CROSS VALIDATION AND GRID SEARCH

To answer the RQ3, the accuracy of 10-Fold cross validation and Grid search algorithm are compared. Table 15 illustrates the accuracy of 10-fold cross validation and Grid search. In this experiment it is found that the grid search algorithm gives better results rather than cross validation.

### Table 15

| Code Smell Dataset | Model | 10-Fold cross validation Accuracy (%) | Grid Search Accuracy (%) |
|--------------------|-------|--------------------------------------|--------------------------|
| Data Class | Naive Bayes | 86.20 | 86.20 |
| KNN | 97.58 | 97.88 |
| MLP | 97.00 | 97.35 |
| Decision Tree | 96.97 | 97.88 |
| Logistic Regression | 96.05 | 97.08 |
| Random Forest | 98.34 | 98.94 |
| God Class | Naive Bayes | 97.40 | 97.40 |
| KNN | 96.57 | 93.81 |
| MLP | 95.52 | 96.82 |
| Decision Tree | 92.85 | 97.02 |
| Logistic Regression | 97.35 | 97.35 |
| Random Forest | 96.43 | 97.88 |
| Feature Envy | Naive Bayes | 95.10 | 95.10 |
| KNN | 91.98 | 84.85 |
| MLP | 95.58 | 96.82 |
| Decision Tree | 96.21 | 98.07 |
| Logistic Regression | 94.71 | 97.58 |
| Random Forest | 97.01 | 97.58 |
| Long Method | Naive Bayes | 97.00 | 97.00 |
| KNN | 96.67 | 97.88 |
| MLP | 92.80 | 96.22 |
| Decision Tree | 94.59 | 95.28 |
| Logistic Regression | 97.59 | 99.52 |
| Random Forest | 99.14 | 99.52 |

J. IMPACT OF TUNING MACHINE LEARNING PARAMETERS

### Table 14

| Dataset | No. of metrics (Wrapper-based) | Metrics selected by Wrapper based feature selection technique |
|---------|-------------------------------|----------------------------------------------------------|
| Data class | 12 | LOC_project, AMWNANNM_type, LOCNANNM_type, NOM_project, WMCNANNM_type, WMNMC_type, NOMANNM_type, CFNANNM_type, NOMNANNM_type, LOCNANNM_type, LOC_type, AMW_type, LOCNANNM_type, LOC_project, LOC_type |
| God class | 9 | FANOUT_type, CFNANNM_type, WMCNANNM_type, WMNMC_type, LOCNANNM_type, ATFD_type, NOAV_type, CFNANNM_type, NOMNANNM_type |
| Feature envy | 11 | FANOUT_method, MeMCL_method, MaMCL_method, FDP_method, LOC_method, CM_method, CYCLO_method, CINT_method, NOAV_method, CFNANNM_method, NOMNANNM_type, NOAV_method |
To answer the RQ4, the impact of tuning machine learning algorithm parameters on performance is checked. Decision tree model achieved highest accuracy of 98.22% when the number of trees are 12, maximum depth 10, and criteria is "Gain ratio", "Gini index" or "Information gain", "Accuracy" and "Entropy". The Table 16 and 17 shows the different combination of parameters for decision tree algorithm. The Multilayer Perceptron model achieved best accuracy 97.62% when Learning Rate is Constant, and Momentum Rate is set to 0.9. In the same case, when the Momentum Rate is set to 0, the accuracy is decreased from 97% to 93%. The Naive Bayes model achieved best accuracy 96.80% when the best parameter Alpha set is 10. In the same case, when the Alpha value is set to 0, the accuracy is decreased 96% to 92%.

The Random Forest model obtained best accuracy 99.74% when the numbers of trees are 35, and the maximal depth is 20. Table 18 shows that when the number of trees grows and the maximal depth remains constant, the accuracy decreases. The logistic regression model achieved highest accuracy 99.52% when parameter 'C' is 0.1 and penalty is 11. A table 19 shows, in the same case, when 'C' value is set to 1 to 100 and penalty is 12 or 11, the accuracy is decreased 97% from 98%.

In this experiment, it is observed that the parameter optimization technique provides positive effects for improving the accuracy of the machine learning algorithms.

### TABLE 16
**APPLYING TUNING ALGORITHMS PARAMETERS ON DECISION TREE MODEL**

| Parameters          | Values       |
|---------------------|--------------|
| Maximum depth       | 10           |
| Criteria            | Information gain |
| Apply pruning       | False        |
| Minimal size of split | 10          |
| Minimal gain        | 0.0          |
| Apply Pre-pruning   | True         |
| Bootstrap           | False        |
| N_estimators        | 300          |

### V. DISCUSSION

#### A. COMPARISON OF OUR RESULTS WITH OTHER RELATED WORKS

Table 20 represents a brief comparison of our method with other related works. These methods applied machine learning techniques on code smell dataset given by Fontana et al. [18]. In the Data class dataset, in our approach's accuracy is 99.74% using Random forest algorithm with all features whereas the Fontana et al. [18] approach's accuracy score is 99.02% in the B-J48 Pruned algorithm. Nucci et al. [62] approach's shows its accuracy in graphical form not in number form. According to this, they obtained approximately 83.00% accuracy using Random forest and J48 algorithm. Mhawish et al. [36] obtained highest accuracy 99.70% using Random forest model.

### TABLE 20
**COMPARISON OF OUR APPROACH WITH OTHER RELATED WORKS.**

| Year   | Author Name          | Dataset                  | Best Algorithm | Accuracy (%) | Best Algorithm | Accuracy (%) | Feature envy | Long method     |
|--------|----------------------|--------------------------|----------------|--------------|----------------|--------------|--------------|-----------------|
| 2016   | Fontana et al. [18]  | B-J48                    | 99.02          | Naive Bayes  | 97.55          | B-J48        | 96.64        | B-J48 Pruned    |
| 2018   | Nucci et al. [62]    | Random Forest and J48    | ~83.00         | Random Forest and J48 | ~83.00 | Random Forest and J48 | ~84.00 | Random Forest and J48 | ~82.00 |
| 2020   | Mhawish et al. [36]  | Random Forest            | 99.70          | GBT          | 98.48          | Decision tree | 97.97         | Random Forest   | 95.97 |
| 2020   | Guggulothu et al. [37]| N/A                      | N/A            | N/A          | N/A            | B-J48        | 99.10        | Random Forest   | 95.90 |
| 2021   | Our Approach         | Random Forest            | 99.74          | Random Forest | 98.21 | Decision tree               | 98.60 | Logistic Regression | 100.00 |
For the God class dataset, in our approach Random forest algorithm using Chi-square feature selection technique, scores the best accuracy of 98.21% and F1-score of 97.00% while the Fontana et al. [18] approach scores maximum accuracy of 97.55% and 98.14% of F1-score in the case of Naive Bayes algorithm. Nucci et al. [62] obtained accuracy approximately 83.00% using Random forest and J48 algorithm. Mhawish et al. [36] obtained highest accuracy 98.48% using GBT model.

For the Feature envy dataset, Decision tree algorithm, scores the best accuracy of 98.60% and F1-score of 97.00% while the Fontana et al. [18] approach scores maximum accuracy of 96.64% and F1- score of 97.44% using the B-JRip algorithm. Nucci et al. [62] obtained approximately 84.00% using Random forest and J48 algorithm. Mhawish et al. [36] obtained highest accuracy 97.97 % using Decision tree model. Guggulothu et al. [37] obtained best accuracy 99.10% using B-J48 Pruned algorithm.

For the Long method dataset, our approach scores the best accuracy of 100% and F1-score 100% using Logistic regression algorithm with all features. When the Logistic regression model is applied with all features for the Long method dataset and take 10% for the test set and 90% for the training set then 100% accuracy is obtained, while when 20% for the test set and 80% for the training set are taken then 99.52% accuracy is achieved. While the Fontana et al. [18] approach scores maximum accuracy of 99.43% and 99.49% of F1-score using the B-J48 Pruned algorithm. Nucci et al. [62] obtained approximately 82.00% using Random forest and J48 algorithm. Mhawish et al. [36] obtained highest accuracy 95.97% using Random forest model. Guggulothu et al. [37] obtained best accuracy 95.90% using Random forest algorithm

B. ANALYSIS OF RESULTS

RQ1: To answer our first research question six machine learning algorithms are applied on four code smell datasets. It has been seen that machine learning algorithms have good capability of predicting code smells. Software metrics that play important role in predicting code smells are identified and shown in Table 13 and Table 14.

RQ2: Chi-square and Wrapper-based feature selection techniques are applied. The results indicate that Random forest and Logistic regression algorithm perform better when all features are used while KNN, Naive Bayes and Multilayer perceptron model’s accuracy is significantly improved using feature selection techniques.

RQ3: To answer our third question the accuracy of 10-Fold cross validation and Grid search algorithm are compared. Table 15 illustrates the accuracy of 10-fold cross validation and Grid search. It is found that the grid search algorithm gives better results rather than cross validation.

RQ4: The impact of tuning machine learning algorithm parameters on performance is checked to answer fourth question. It is observed that the parameter optimization technique provides positive effects for improving the accuracy of all machine learning algorithms used in this work.

C. THREATS TO VALIDITY

Here possible threats will be discussed that might have affect our experiment and how we tried to mitigate them.

1) THREATS TO INTERNAL VALIDITY

The main internal threat in our study is the dataset. The dataset used in this study is taken directly from Fontana et al. [18]. They developed the dataset using code smell advisors to select candidates from large repository of 74 heterogeneous software systems (Qualitas Corpus) and validated manually the 420 examples for each code smell. Different metrics are considered to generate dataset. All of them might not have impact on the performance of models implemented. To manage this threat, two feature selection techniques are used to find metrics that are more impactful and compared results found using both techniques. As for the experimentation prediction models, the model is implemented in Python which is now widely accepted as a better programming language with large set of the libraries in most of the domains

2) THREATS TO EXTERNAL VALIDITY

In our experiment, threats to external validity are as follows. The first threat is that the dataset used has only two types of code smells, namely class-level and Method-level smells. The second threat is related to application software from which dataset is generated are all Java source code. Thus, our approach might not be appropriate for C/C++ source codes.

3) THREATS TO CONCLUSION VALIDITY

This threat focuses on evaluating the performance of prediction models. 10-fold cross-validation is used to evaluate predictive models using multiple evaluation metrics, including accuracy, F1-score, precision, and recall. Although these evaluation metrics are not sufficient, to compare our results with existing techniques, the same metrics are used to evaluate the performance of models as taken from Fontana et al. [18]. To manage this threat, the confidence value of each prediction is calculated, and using feature selection, the important metrics are identified that have higher impact on prediction which helps the prediction model to take the correct decisions.

VI. CONCLUSION AND FUTURE WORK

In this paper, a novel approach is proposed to predict the code smells from the software and detect the metrics that contribute a significant role in the detection process using machine learning techniques. Four code smell datasets (God class, Data class, Feature-envy and Long-method) generated from 74 open-source system (Fontana et al. [18]) are used. The Chi-Square and Wrapper-based feature selection technique is applied to detect the best metrics that can be used to improve the accuracy. The Grid search algorithm is applied for the parameter optimization
technique that significantly improves the accuracy of all algorithms. The six different machine learning algorithms (Naive Bayes, KNN, Multilayer Perceptron, Decision Tree, Random Forest and Logistic Regression) are used to detect metrics from code smell datasets that are generated from 74 open-source systems (Fontana et al. [18]). The main contribution of this paper is two-fold: In first step machine learning algorithms are used for detecting the code smells. In the second step the performance measurement (Accuracy, Precision, Recall, and F1-Score), are calculated. The performance was improved by applying chi-square and wrapper-based feature selection techniques along with grid search algorithm with 10-fold cross-validation technique. In this paper, it is observed that for Data class dataset Random Forest algorithm achieved highest accuracy 99.74% when all features were considered while worst performance was achieved by Naive Bayes 83.10% when all features were considered. In case of God class dataset Random Forest algorithm achieved highest accuracy 98.21% when chi-square feature selection technique was used while worst performance was achieved by KNN 93.81% when chi-square feature selection technique was used. In case of Feature-envy dataset Decision tree algorithm achieved highest accuracy 98.60% when all features were considered while worst performance was achieved by KNN 84.85% using wrapper-based feature selection technique. In case of Long-method dataset Logistic Regression achieved highest accuracy 100% when all features were considered while worst performance was achieved by Naive Bayes 95.20% using chi-square feature selection technique.

REFERENCES

[1] M. M. Lehman, “Programs, life cycles, and laws of software evolution,” Proceedings of the IEEE, vol. 68, no. 9, pp. 1060–1076, 1980.
[2] Wiegers K, Beatty J. Software Requirements. Pearson Education, 2013.
[3] Chung L, do Prado Leite J C S. On non-functional requirements in software engineering. In Conceptual Modeling: Foundations and Applications-Essays in Honor of John Mylopoulos, Borgida AT, Chaudhri V, Giorgini P, YuE (eds.), Springer, 2009, pp. 363-379.
[4] Fowler M, Beck K, Brant J, Opdyke W, Roberts D. Refactoring: Improving the Design of Existing Code (1st edition). Addison-Wesley Professional, 1999.
[5] G. Travassos, F. Shull, M. Fredericks, V. R. Basili, Detecting defects in object-oriented designs: using reading techniques to increase software quality, in: ACM Sigplan Notices, Vol. 34, ACM, 1999, pp. 47-56.
[6] O. Ciupke, Automatic detection of design problems in object-oriented reengineering, in: Technology of Object-Oriented Languages and Systems, 1999. TOOLS 30 Proceedings, IEEE, 1999, pp. 18-32.
[7] W. Abdelmoez, E. Kosha, A. F. Iesa, Risk-based code smells detection tool, in: The International Conference on Computing Technology and Information Management (ICCTIM2014), The Society of Digital Information and Wireless Communication, 2014, pp. 148-159.
[8] E. Murphy-Hill, A. P. Black, An interactive ambient visualization for code smells, in: Proceedings of the 5th international symposium on Software visualization, ACM, 2010, pp. 5-14.
[9] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshiyvanyk, A. De Lucia, Mining version histories for detecting code smells, IEEE Transactions on Software Engineering 41 (5) (2015) 462-489.
[10] H. Liu, X. Guo, W. Shao, Monitor-based instant software refactoring, IEEE Transactions on Software Engineering (2013) 1.
[11] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, D. Poshiyvanyk, Detecting bad smells in source code using change history information, in: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, IEEE Press, 2013, pp. 268-278.
[12] A. A. Rao, K. N. Reddy, Detecting bad smells in object oriented design using design change propagation probability matrix 1 (2007).
[13] R. Marinescu, Detection strategies: Metrics-based rules for detecting design aws, in: Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on, IEEE, 2004, pp. 350-359.
[14] N. Moha, Y.-G. Gueleneuc, A.-F. Duchien, et al., Decor: A method for the specification and detection of code and design smells, IEEE Transactions on Software Engineering (TSE) 36 (1) (2010) 20-36.
[15] N. Tsantalis, A. Chatzipiergiou, Identification of move method refactoring opportunities, IEEE Transactions on Software Engineering 35 (3) (2009) 347-367.
[16] N. Moha, Y.-G. Gu_el_eune, A.-F. Le Meur, L. Duchien, A. Tiberghien, From a domain analysis to the specification and detection of code and design smells, Formal Aspects of Computing 22 (3-4) (2010) 345-361.
[17] W. Kessentini, M. Kessentini, H. Sahraoui, S. Bechikh, A. Ouni, A cooperative parallel search-based software engineering approach for code-smell detection, IEEE Transactions on Software Engineering 40 (9) (2014) 841-861.
[18] F. A. Fontana, M. V. Mäntylä, M. Zanoni, A. Marino, Comparing and experimenting machine learning techniques for code smell detection, Empirical Software Engineering 21 (3) (2016) 1143-1191.
[19] Yamashita A, Moonen L. Exploring the impact of inter smell relations on software maintainability: An empirical study. In Proc. the 35th Int. Conf. Softw. Eng., May 2013, pp. 682-691.
[20] Yamashita A, Counsell S. Code smells as system-level indicators of maintainability: An empirical study. J. Syst. Softw., 2013, 86(10): 2659-2673.
[21] Yamashita A, Moonen L. Do code smells reflect important maintainability aspects? In Proc. the 28th IEEE Int. Conf. Softw. Maintenance, September 2012, pp.306-315.
[22] Sjöberg D I K, Yamashita A, Anda B C D, Mockus A, DybøåA. Quantifying the effect of code smells on maintenance effort. IEEE Trans. Softw. Eng., 2013, 39(8): 1144-1156.
[23] Sahin D, Kessentini M, Bechikh S, Ded K. Code-smells detection as a bi-level problem. ACM Trans. Softw. Eng. Methodol., 2014, 24(1): Article No. 6.
[24] Olbrich S, Cruzes D S, Basili V, Zazworth N. The evolution and impact of code smells: A case study of two open source systems. In Proc. the 3rd International Symposium on Empirical Software Engineering and Measurement, October 2009, pp.390-400.
In the evolution of Data Normalization, Ph.D. thesis, University of Wisconsin Milwaukee (Aug. 2009). A Hybrid Approach To Detect Code Smells using Deep Learning. In Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE, ISBN 978-989-758-300-1, pages 137-146. DOI: 10.5220/0006709801370146.

[44] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, J. Noble, The qualitas corpus: A curated collection of java code for empirical studies, in: Software Engineering Conference (APSEC), 2010 17th Asia Pacific, IEEE, 2010, pp. 336-345.

[45] S. Olbrich, D. Cruzes, D. I. K. Sjoberg, Are all code smells harmful? A study of god classes and brain classes in the evolution of three open source systems, in: Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2010), Timisoara, Romania, 2010, pp. 1–10. doi:10.1109/ICSM.2010.5609564.

[46] C. Marinescu, R. Marinescu, P. Mihancea, D. Ratiu, R. Wettel, iPlasma: An integrated platform for quality assessment of object-oriented design, in: Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM 2005), IEEE, Budapest, Hungary, 2005, pp. 77–80, (Industrial & Tool Proceedings), Tool Demonstration Track.

[47] K. Nongpong, Integrating "code smells" detection with refactoring tool support, Ph.D. thesis, University of Wisconsin Milwaukee (Aug. 2012).

[48] R. Marinescu, Measurement and quality in object oriented design, Ph.D. thesis, Department of Computer Science, “Polytechnic” University of Timisoara (2002).

[49] Riel A J. Object-Oriented Design Heuristics (1st edition). Addison-Wesley Professional, 1996.

[50] Forman G, Scholz M, Rajaram S. Feature shaping for linear SVM classifiers. In Proc. the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, June 2009, pp.299-308.

[51] Peshawa J. Muhammad Ali, Rezinha H. Faraj, “Data Normalization and Standardization : A Technical Report”, Machine Learning Technical Reports, 2014, 1(1), pp 1-6.

[52] Chicco D. Ten quick tips for machine learning in computational biology. BioData Mining, 2017, 10(1): 35.

[53] Hsu C W, Chang C C, Lin C J. A practical guide to support vector classification. Technical Report, Taiwan University, 2008. https://www.csie.ntu.edu.tw/~cjlin/papers/guide.pdf, May 2020.

[54] Catal C (2012) Performance evaluation metrics for software fault prediction studies. Acta Politeh Hung 9(4):193–206.

[55] Romero E, Sopena JM (2008) Performing feature selection with multilayer perceptrons. IEEE Trans Neural Netw 19(3):431–441.

[56] Webb G.I. (2011) Naïve Bayes. In: Sammut C., Webb G.I. (eds) In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_576.

[57] https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm.

[58] Cortez P. (2012) Data Mining with Multilayer Perceptrons and Support Vector Machines. In: Holmes D.E., Jain L.C. (eds) Data Mining: Foundations and Intelligent Paradigms. Intelligent Systems
Seema Dewangan received the MCA degree from Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya (A Central University) Bilaspur, Chhattisgarh in 2012. She is currently Research Scholar at Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya (A Central University) Bilaspur, Chhattisgarh, India. Her research interest includes Design Patterns mining, Code smell detection.

Rajwant Singh Rao is currently working as Assistant Professor in Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya (A Central University) Bilaspur, Chhattisgarh, India. His research interest includes Design Patterns mining, Code smell detection and machine learning. He received his Ph.D. degree from Department of Institute of Science, Banaras Hindu University Varanasi.

Alok Mishra is Professor in Informatics and Digitalization at Molde University College (A Specialized University in Logistics) Norway. He is also associated as Professor in Software Engineering at Atılım University, Turkey. His areas of research interests are in software engineering, information system, information technology and artificial intelligence. Prof. Mishra is an editorial board member of many reputed journals including Computer Standards and Interfaces (Elsevier), Journal of Universal Computer Science, Computing & Informatics, Data Technologies and Applications Journal etc. He is actively involved in editing special issues of reputed journals in his areas of research interest. Prof. Mishra had also extensive experience in online education related to Computing and Management disciplines. In teaching, he has received excellence in online education award by U21Global Singapore while in research he has been awarded by Scientific and Research Council of Turkey and Board of Management of University for outstanding publications in Science and Social Science Citation Indexed (Thomson Reuters) journals. He is a recipient of many scholarships, international awards and research projects.

Manjari Gupta is currently working as an associate professor (computer science) in the DST-Centre for Interdisciplinary Mathematical Sciences, Banaras Hindu University, Varanasi. Her area of interest is software engineering and artificial intelligence. More specially, her work is on design patterns detection, code smell detection, Object-oriented frameworks and application of artificial intelligence in this area. She also works on video processing, in particular action recognition using deep learning techniques. Dr. Gupta received her Ph.D. from Indian Institute of Technology (Banaras Hindu University) in the year 2008.

Reference Library, vol 24. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-23241-1_2.

[59] Murty M.N., Devi V.S. (2011) Decision Trees. In: Pattern Recognition. Undergraduate Topics in Computer Science, vol 0. Springer, London. https://doi.org/10.1007/978-0-85729-495-1_6.

[60] Soto A.J., Cecchini R.L., Vazquez G.E., Ponsoni I. (2008) A Wrapper-Based Feature Selection Method for ADMET Prediction Using Evolutionary Computing. In: Marchiori E., Moore J.H. (eds) Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. EvoBIO 2008. Lecture Notes in Computer Science, vol 4973. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-78757-0_17.

[61] https://images.app.goo.gl/1PAv9VM1Rm7xBWV97

[62] Di Nucci, Dario & Palomba, Fabio & Tamburri, Damian & Serebrenik, Alexander & Lucia, Andrea. (2018). Detecting Code Smells using Machine Learning Techniques: Are We There Yet?, 25th IEEE International Conference on Software Analysis, Evolution, and Reengineering At: Campobasso, Ita10.1109/SANER.2018.8330266.

[63] Muhammad Ilyas Azeem, Fabio Palombad, Lin Shi, Qing Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis." Information and Software Technology 108 (2019): 115-138.

[64] F. Pecorelli, F. Palomba, D. Di Nucci and A. De Lucia, "Comparing Heuristic and Machine Learning Approaches for Metric-Based Code Smell Detection," 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 2019, pp. 93-104, doi: 10.1109/ICPC.2019.00023.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3133810, IEEE Access