Ebtehal Alnaihom. et. al. International Journal of Engineering Research and Applications
www.ijera.com
ISSN: 2248-9622, Vol. 13, Issue 9, September 2023, pp 166-178

RESEARCH ARTICLE OPEN ACCESS

Detecting Design Smells Using Machine Learning: A Case Study

Ebtehal Alnaihom, Huda Alzureedi, Mrwan BenIdris
Computer Science Department, Faculty of Information Technology, University of Benghazi, LIBYA
itstd.4304@uob.edu.ly, itstd.4160@uob.edu.ly, mobenidris@uob.edu.ly

ABSTRACT
Continuous development in software results in complexity and this confuses the design and programming stages, which makes the maintenance of the software difficult and thus affects the quality of software. Bad smells refer to weak solutions that can lead to issues with software maintainability. These smells are common problems that arise in implementation, design, and architecture, and can be identified by using a set of metrics and their threshold values. This paper conducted multiple case studies on 9 Apache projects in order to (1) determine the most effective tool for detecting bad smells, (2) learn how to detect bad smells using the most effective tools, and (3) identify the detection strategies used by those tools. Additionally, machine learning techniques were used to identify Design Smells. The aim was to demonstrate that ML techniques can be used to identify design smells, with the created dataset being made available once our work is accepted and published.
Keywords – design smell, machine learning, detection, refactoring.

Date of Submission: 06-09-2023 Date of acceptance: 18-09-2023

I. INTRODUCTION

Code smells are indicators of quality issues that can have a negative impact on many aspects of software quality. Fowler [1] was the first to use the metaphor of "code smells" to refer to signs of a weak solution that can lead to issues with code maintainability. Too many code smells in a system can make it difficult to maintain and develop further. Code smells are also known as bad smells, code anomalies, design flaws, and anti-patternsm [2] .

Bad smells can be used to detect Technical Debt (TD), and it is the most commonly used indicator for TD, as reported by Alves et al. [3] and Ben Idris et al. [4]. TD was first introduced by Ward Cunningham, who famously said, “Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite” [5]. Seventeen types of TD have been mentioned by [4], many of them, but not all, can be identified by using bad smells and each type of smell can be recognized by using a set of metrics and their threshold values. After detecting the bad smells, refactoring can be performed to reduce technical debt. However, many software developers opt for adding new features or fixing existing bugs rather than refactoring, as refactoring does not offer any immediate rewards. God Class (GC) and Data Class (DC) are two types of design smells. These two design smells have been characterized as follows: Data Class consists of fields with getters and setters and nothing else, is often manipulated too much by other classes. God Class is a class that does too much work on its own, delegating only minor tasks to other classes and using data from other classes [6]. We focus our attention on those two types of design smells because they are very common.

Normally, tools are used to detect bad smells during maintenance phase. However, the problem appears if different tools are used to detect GC and DC due to the fact that different tools use different thresholds, which affect the accuracy of the detector results. The question is how we should detect the GC and DC with high accuracy. For that reason, we will use machine learning techniques to detect GC and DC. Different machine learning models will be applied to achieve the best possible classification accuracy. The objective of this research paper was as follows: Carry out empirical studies on open-source projects, use machine learning techniques to detect design smells, and provide datasets on design smells. The main contributions of this paper are the following:

- We conducted a case study to explore the use of bad smell tools, analyze open source projects, and extract metrics values for the purpose of detecting GC and DC.
- We applied machine learning techniques to identify GC and DC in open-source projects.

- We compiled a dataset that will be made available to other researchers upon the publication of our findings.

The rest of this paper is organized as follows. Section 2 highlights the related works. Our methodology is discussed in Section 3. Section 4 presents a case study will be conducted and machine learning will be applied to detect design smells. The results, and discussion are provided in Section 5. Section 6 presents the threats that may be affecting the findings and Section 7concludes our work.

II. RELATED WORKS

Khoma et al. [7] used a Bayesian approach to detect code and design smells. The goal of the paper was to develop a method for automatically detecting code and design smells in software systems. The developed approach was able to detect code and design smells with an accuracy of up to 97%. Later, Khoma et al. [8] proposed a new approach to detecting anti-patterns, called BDTEX. BDTEX is a GQM-based Bayesian approach that uses a probabilistic model to detect anti-patterns in software systems. The goal of this research paper is to develop an effective and efficient method for detecting anti-patterns in software systems. The authors evaluated the performance of BDTEX on two real-world software systems and found that it was able to detect anti-patterns with high accuracy and low false positive rates.

Maneerat and P. Muenchaisri [9] used Machine Learning Techniques to predict bad smells from software design models. The results showed that the proposed approach was able to detect bad smells with an accuracy of up to 90%. With an accuracy of up to 97%. Maiga et al. [10] used a Support Vector Machine (SVM) to detect anti-patterns in software systems. Their goal was to develop a method for automatically detecting anti-patterns in software systems using SVM. They showed that SVM was more accurate than other methods such as decision trees and neural networks. Kaur et al. [2] used SVM approach to detect code smells. The goal of the paper was to develop an automated system for detecting code smells in software systems. The authors found that their SVM-based approach was able to accurately detect code smells with an accuracy of up to 97%. Additionally, they found that their approach was able to detect code smells more quickly than other existing methods.

Fontana et al. [11] proposed an approach based on machine learning technique to detect code smells. The results showed that the proposed approach was able to detect code smells with an accuracy of up to 80%. In 2016, Fontana et al. [12] used a dataset of Java projects to compare the performance of different machine learning techniques for code smell detection. The goal of this paper was to evaluate the effectiveness of different machine learning techniques for detecting code smells in software projects. The results showed that SVMs and Random Forests (RFs) were the most effective machine learning techniques for detecting code smells in software projects. In 2017, Fontana et al. [13] classified code smell severity using Machine Learning Techniques. The results showed that the proposed approach was able to classify code smell severity with an accuracy of up to 90%.

In 2018, Di Nucci et al [14] replicated Fontana's work to compare the performance of different machine learning techniques for code smell detection. Their aims were to evaluate the effectiveness of different machine learning techniques for detecting code smells in software projects and to compare their performance with existing approaches such as static analysis tools and rule-based systems. Their results showed that SVMs were the most effective machine learning technique for detecting all four types of code smells in software projects, outperforming existing approaches such as static analysis tools and rule-based systems. Based on relational association rule mining, Czibula et al. [15] detected software design defects. The results showed that the proposed approach was able to detect software design flaws with an accuracy of up to 95%.

Finally, in 2020, Ben Idris et al. [16] used machine learning-based approach to prioritize software components' risk. The goal of the paper was to develop a method for prioritizing software components' risk. The results showed that the proposed approach was able to accurately prioritize software components' risk with extraordinary performance. We have distinguished our study from the previous literature by noting the following four points:

- Unlike [2] [8] [10] [13] [14], we created our own dataset based on Apache projects, which we then analyzed.

- We focused on detecting Design smell specifically God Class and Data Class smells, while [9] concentrated on other types of design smells and [11] focused on code smells, and [7] [8] [10] did not detect Data Class. While [13] classified the severity of the God and Data Class.
- Our study employed the PMD tool for extracting the internal structure and rules used to detect GC and DC smells, whereas literary [17] used a different tool related to code smell detection.
- We compared our study to previous literature by using three metrics to detect GC class and four metrics to detect Data Class while [12] utilized one and three metrics to detect the God Class and Data Class smells respectively. Additionally, we employed more than 30 machine learning models, which are detailed in Section 3, whereas the prior studies used only one or no more than 30 models. This comparison provides a clear understanding of how our research differs from what has been done before. By presenting this information in an organized way, we aim to show the originality of our work.

III. METHODOLOGY

The purpose of this work is to build a machine learning model that can detect design smells in open-source projects. To do this, we must create our own dataset. We will use a design smell tool to analyze various open-source projects in a case study. This case study will help us building our own dataset by answering these three research questions:

- **RQ1**: What are the most effective tool used by researchers to identify bad smells?
- **RQ2**: Which type of bad smells can be detected by the most effective tools in RQ1?
- **RQ3**: What are the detection strategies used by the most effective tools in RQ1?

RQ1 helps us find the best tool that can be used to generate our dataset, while RQ2 assists us in focusing on the type of smells that are deemed more hazardous based on the effective tools. Finally, RQ3 aids us in discovering the most reliable methods used by researchers to detect design smells.

3.1 Select a Detection Tool

In order to answer RQ1, we conducted a search on Google Scholar for papers related to the design smell detection tool. We discovered 10 papers, and after reading them, it was concluded that JDeodorant, PMD, iPlasma, InFusion and DÉCOR were the most effective tools according to the researchers.

JDeodorant is a software tool that operates within the Eclipse environment and aims to improve software design. It accomplishes this by detecting common design issues, referred to as "code smells", and offering recommended solutions through appropriate refactoring techniques. The tool uses unique and innovative methods to identify these code smells and suggest the right course of action. PMD analyzes Java source code to identify potential issues, including potential bugs like dead code, empty control structures, unused variables, and duplicated code. It also detects code smells and allows for customization of metric threshold values. iPlasma is a comprehensive platform for evaluating the quality of object-oriented systems, covering all phases of analysis, from model extraction to high-level metric-based analysis and duplication detection. It can detect various code smells known as "disharmonies," including identity disharmonies, collaboration disharmonies, and classification disharmonies. Further information on these disharmonies can be found. InFusion is a comprehensive solution for assessing and improving the quality of systems at both the architectural and code levels. It covers all phases of the analysis process and is capable of detecting over 20 design weaknesses and coding inefficiencies, including code duplication, breaches in encapsulation, excessive coupling, and suboptimal class hierarchy design. InFusion is a product of the expansion of iPlasma, featuring additional functionalities. DÉCOR is a method for specifying and automatically detecting code and design weaknesses, commonly referred to as anti-patterns. This approach specifically defines six code smells and creates detection algorithms using templates. The precision and recall of these algorithms were then evaluated. The term DECOR refers to the component created for detecting these weaknesses [18].

WE found that PMD, iPlasma, and JDeodorant are the most prevalent tools in this area. As reported in [18] and [19], a comparison of these tools was conducted. JDeodorant was dismissed from the comparison due to its utilization of a custom specification language. The comparison was narrowed down to PMD and iPlasma. However,

iPlasma was eventually excluded from the comparison because the threshold values that use it. It was concluded that PMD was the preferred tool for smells detection, given its superior performance in terms of threshold value utilization

3.2 Select Type of Design Smells

The answer of RQ1 helps us address RQ2. And based on that, we conducted a literature review to gain a thorough understanding of the predominant design smells. This will assist us in concentrating our attention on the most critical design smell. Table 1 categorizes the bad smells detected by the tool [18].

Table 1: Information about bad smells detection tools

Tool	Type	Supp. Lang.	Code smell
JDeodorant	Eclipse Plug-in	Java	Feature Envy, God Class, Long Method, Type Checking, and Duplicate code
Infusion	Standalone application	C, C++, Java	Cyclic Dependencies, Brain Method Data Class, Feature Envy, God Class, Intensive Coupling, Missing Template, Method, Refused Parent, Bequest, Significant, Duplication, and Shotgun Surgery
iPlasma	Standalone application	C++, Java	Brain Class, Brain Method, Data Class, Dispersed Coupling, Feature Envy, God Class, Intensive Coupling, Shotgun Surgery, Refused Parent, Bequest, Tradition Breaker By custom rules, Long Method, Long Parameter List, Speculative, and Generality
PMD	Eclipse Plugin or Standalone app.	Java, JavaScript, Apex and Visualforce	Data Class, God Class. Long Method, and Long Parameter List
DÉCOR (Black Box)	Standalone application.	Java	Large Class, Lazy Class, Long Method, Long Parameter List, Refused Parent, Bequest, and Speculative Generality

After examining the researchers' recommendations, we found that God Class and Data Class are the most prominent design smells that have a negative impact on open source software [14]. A God Class is a software design anti-pattern that occurs when a single class takes on too many responsibilities. This type of class is often characterized by having an excessive number of methods, attributes, and dependencies. It can also be identified by its lack of cohesion, as the methods and attributes are unrelated to each other. A God Class can lead to code that is difficult to maintain and debug, as it is often difficult to understand the purpose of the class and how its various components interact with each other. On the other hand, Data Class Smell is an anti-pattern in software design where a class contains only data fields and no methods or behaviour. This type of class can lead to code that is difficult to maintain and debug, as it does not provide any context for understanding how the data fields are used or related to each other. Additionally, Data Class Smell can lead to code duplication if multiple classes contain similar data fields but no behaviour [20].

3.3 Select Design Smells Detection Strategies

After answering RQ1, RQ2, and RQ3, we decided to delve into the God and scent detection strategies used by the selector. We ran the tool and examined its results, as well as browsed through the tool's documentation and help. The metrics proposed by Lanza and Marinescu [20] for detection strategies were applied, as indicated in Table 2. The rules for identifying God Classes and Data Classes were formulated as presented in Equations1 and 2, respectively.

$$\text{IF}(\text{WMC} \geq \text{High} \cap \text{TCC} < ONETHIRD \cap \text{ATFD} > \text{FEW}) \ldots. (1)$$

$$IF(WMC < High \cap [NOAM + NOPA] > FEW) \cup (WMC < VeryHigh \cap [NOAM + NOPA] > MANY]) \cap (WOC < ONETHIRD) \ldots (2)$$

Table 2: Metrics used for design smells detection

Short Name	Long Name
ATFD	Access To Foreign Data
WMC	Weighted Methods Count
TCC	Tight Class Cohesion
NOAM	Number Of Accessor Methods
NOPA	Number Of Public Attributes
WOC	Weight Of Class

IV. CONDUCT A CASE STUDY

4.1 Collect the Software Systems

We decided to conduct a case study using the PMD tool on open source software projects that have been studied by researchers in literature. At the beginning of our research, we intended to analyze the projects of graduate students at our college. However, when we started to look into them, we noticed that some of these projects did not have source code or were written in different languages that makes hard to find tools analyzing some of that programming languages. This led us to modify our plan and focus on the most popular open source software projects that had been examined by Deeb et al. [21] and Lenarduzzi et al. [22], even though their objectives were different from ours. We randomly selected one release from each Apache project studied by [21] and [22], as shown in Table 3. Apache Projects [23] are a group of open source software projects managed by the Apache Software Foundation. These projects span a wide range of topics, from web servers and databases to big data and machine learning. The Apache Projects are developed in an open and collaborative environment, with contributions from individuals and organizations around the world. The projects are released under the Apache License, which allows for free use, modification, and redistribution of the software.

Table 3: Projects and releases used in this case study

Apache project	Release	Description	Link
Aurora	0.12.0	a service scheduler that runs on top of Apache Mesos, enabling you to run long-running services, cron jobs, and ad-hoc jobs that take advantage of Apache Mesos' scalability, fault-tolerance, and resource isolation	https://aurora.apache.org/ notice: This project has retired. For details please refer to https://attic.apache.org/projects/aurora.html
Beam	2.7.0	Apache Beam is an open-source, unified programming model for defining and executing batch and streaming data processing pipelines, enabling parallel processing of large scale data processing across a variety of runners including Apache Flink, Apache Spark, Google Cloud	http://beam.apache.org/
Cocoon	2-1-8	Apache Cocoon is an open-source web development framework that provides a component-based model for building and deploying dynamic, XML-based web	http://cocoon.apache.org/
commons-collections	3.3	Apache Commons Collections is an open-source Java library that provides a set of collection classes and utilities that can be used to enhance the Java Collections Framework	http://www.apache.org/licenses/LICENSE-2.0
commons-configuration	1.3	Apache Commons Configuration is an open-source Java library for reading configuration data from a variety of sources, including property files, XML, and other	http://www.apache.org/licenses/
HTTP components-client	4.5.11	Apache HTTP Components Client is a library in Java that provides low-level client-side HTTP services. It enables developers to send HTTP/HTTPS requests to a web server and process the responses. HttpComponents Client is part of the Apache HttpComponents project and provides a flexible, efficient, and easy-to-use set of components for implementing HTTP-based client applications	http://hc.apache.org/
httpcomponents-core	4.4.12	Apache HttpComponents is a collection of Java components that provide low-level and high-level APIs for HTTP-based communication.	http://hc.apache.org/
XML Graphics	1-6	Apache XML Graphics is an open-source project that provides a Java library for rendering and generating graphics in a portable and flexible manner. It provides a unified API for working with different graphics formats, such as SVG, PDF, and AWT. The library is designed to be extensible and can be easily integrated into Java applications	http://xml.apache.org/batik/
zookeeper	3.4.5	Apache ZooKeeper is a distributed coordination service for distributed systems. It provides a simple API for maintaining configuration information, naming, and providing distributed synchronization across a large number of hosts. ZooKeeper is used to coordinate the activities of a large number of hosts in a dynamic, distributed environment.	http://xml.apache.org/batik/

4.2 Analysis the Software Systems

Figure 1 presents the smells distribution among God Class (GC) and Data Class (DC). The data consists of a total of 651 smell instances, with 426 instances belonging to GC and 225 instances belonging to DC.

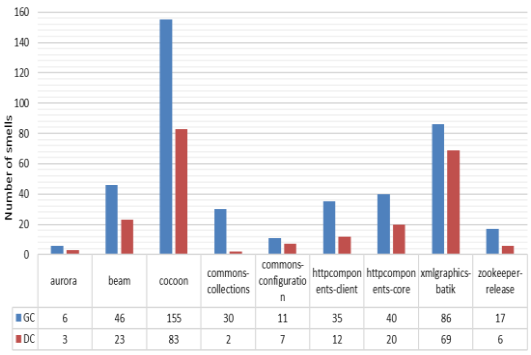

Figure 1. The distribution of God Class and Data Class on the selected Apache project

In the Aurora Release 0.12.0 project, there were 6 God Class (GC) smell and Data class (DC) smell occurrences, making up 0.92% and 0.46% of the total GC and DC smell respectively, which was the lowest percentage of smells. The Beam Release 2.7.0 project had 46 GC and 23 DC smell, with percentages of 7% and 3.5%. The Cocoon Release 2-1-8 project had the highest number of GC and DC smells with 155 GC and 83 DC smells, accounting for 23% and 12%. The Commons-Collections Release 3.3 project had 30 GC and 2 DC smell, with rates of 4% and 0.3%. Lastly, the Commons-Configuration Release 1.3 project had 11 GC and 7 DC smell, with rates of 1.6% and 1%. The Httpcomponents-Client Release 4.5.11 project had 35 GC smells and 12 DC smells, with a rate of 5.3% for GC and 1.8% for DC. The Httpcomponents-Core Release 4.4.12 project had 40 GC smells and 20 DC smells, with a rate of 7.1% for GC and 3% for DC. The XML Graphics Release 1-6 project had 86 GC smells and 69 DC smells, with a rate of 13% for GC and 10.5% for DC. Lastly, the Zookeeper Release 3.4.5 project had 17 GC smells and 6 DC smells, with a rate of 2.6% for GC and 0.92% for DC respectively.

4.3 Apply Machine Learning

Weka 3.8.6 will be used to develop our machine learning models. Weka is a powerful open source machine learning tool developed by the University of Waikato in New Zealand. It is widely used for data mining, predictive analytics, and other machine learning tasks. WEKA provides a graphical user interface (GUI) that allows users to interact with the software and perform various tasks such as data pre-processing, classification, clustering, regression, association rule mining, and visualization. WEKA also supports a wide range of algorithms for each task. WEKA is written in Java and can be run on any platform that supports Java. It can be used to analyze large datasets from various sources such as CSV files, databases, or even Excel spreadsheets. WEKA also provides an API which allows users to develop their own algorithms or integrate existing ones into their applications. Additionally, WEKA includes a variety of tools for evaluating the performance of different machine-learning algorithms on datasets. These tools include cross-validation techniques and statistical tests for comparing different models [24] [25].

The main focus of this research paper was to apply supervised machine learning techniques for design smell detection. We used three internal structure metrics to identify God Classes and we used four internal structure metrics to identify Data Classes. In addition, nine Apache open-source system projects were chosen from the Apache website. Equation 1 and 2 were used to detect GC and DC respectively, while Table 3 provides information about the nine projects.

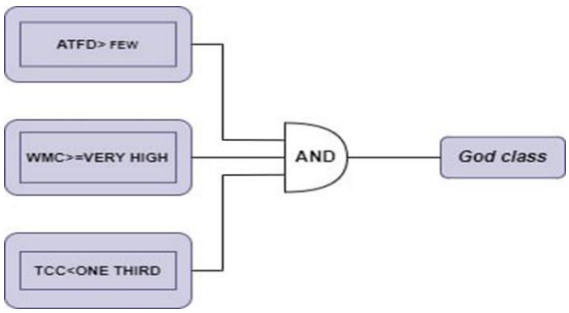

Figure 2. Rules to detect God Class

As we mention above, we used three internal structure metrics to identify God Classes (Figure 2). If Access To Foreign Data (ATFD) metric is greater than FEW, it implies that the classes heavily access data of other simpler classes, either directly or by using accessor methods. The higher the ATFD value, the more likely it is that the class is a God Class. If Weighted Methods Count (WMC) metric is equal to or higher than VERY HIGH, it indicates that the classes are large and

complex. Lastly, if Tight Class Cohesion (TCC) metric is less than one-third, it suggests that the class has a lot of non-communicative behaviour (low cohesion between the methods belonging to that class). This means that the Class carries out multiple distinct tasks with separate sets of attributes, which has a detrimental effect on its cohesion. The lower the TCC value, the more likely it is that the class is a God Class.

As shown in Figure 3, a specific strategy is used to identify Data Classes based on four metrics which are Weight Of Class, (WOC), Number of Accessor Methods (NOAM), Number of Public Attributes (NOPA), and Weighted Method Count (WMC). The class is a Data Class if (1) WOC is less than one-third that means it suggests that interface of the class reveals data rather than offering services AND (2) the class reveals many attributes and is not complex. To find if (2) the class reveals many attributes and is not complex by checking if:

1. It has more than a few public data members (NOAP + NOAM > FEW) AND its complexity is not HIGH. This would indicate that the class is relatively small, has minimal functionality, and only provides some data and accessors to that data.
2. It has many public data (NOAP + NOAM > MANY) AND class complexity is not VERY HIGH. That means this class provides MANY public data but the complexity of the class (WMC) to be considerably not VERY HIGH. We just look at the class less that very high complexity because it does not conceptually fit the Data Class term [20].

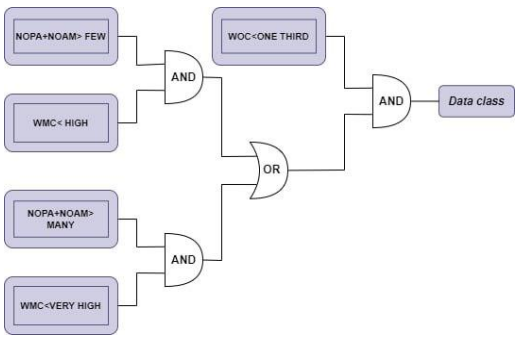

Figure 3. Rules to detect Data Class

A 10-fold cross-validation techniques was used to estimate the performance of each predictive. According to Kohavi [26] it has less bias because it divides the data into 10 segments (folds), where and the first segment was tested while the other nine were used as training data. This process was repeated for all segments. We measured the accuracy and performance of each machine learning models to evaluate them. By looking at the Accuracy, Receiver Operating Characteristics Curve (ROC), F-measure, and Matthews Correlation Coefficient (MCC) values we can determine whether the ML model is a good model. However, before that, we will give a short explanation of each measurement.

Accuracy is a measure of how accurately a machine learning model can predict the correct outcome. It is calculated by dividing the number of correct predictions by the total number of predictions. A higher accuracy indicates that the model is better at predicting the correct outcome. To interpret the results, a higher accuracy indicates that the model is more accurate in its predictions.

Receiver Operating Characteristic (ROC) is a measure of how well a machine learning model can distinguish between two classes. It is calculated by plotting the true positive rate against the false positive rate for different thresholds. A higher ROC indicates that the model is better at distinguishing between two classes. To interpret the results, a higher ROC indicates that the model has better discrimination power between two classes.

F-measure is a measure of how well a machine learning model can classify data points into different classes. It combines precision and recall into one metric and is calculated by taking the harmonic mean of precision and recall scores. A higher F-measure indicates that the model has better classification performance. To interpret the results, a higher F-measure indicates that the model has better classification performance across different classes.

Matthews Correlation Coefficient (MCC) is a measure of how well a machine learning model can classify data points into different classes while taking into account true positives, false positives, true negatives, and false negatives. It ranges from -1 to 1 where 1 indicates perfect prediction and -1 indicates perfect misclassification. To interpret the results, a higher MCC score indicates that the model has better classification performance across different classes while taking into account all four types of outcomes (true positives, false positives, true negatives, and false negatives) [27].

V. RESULTS

This paper utilized thirty different machine learning algorithms to create models that can identify two types of design smells, specifically God and Data Class. The machine learning inputs were obtained from the internal metrics extracted from the source code of Apache projects. The ML algorithm accuracies for the God Class ranged from 99% to 92%, while for the Data Class they ranged from 99% to 86%. However, it is important to note that our observations may be biased due to the imbalance in our dataset, which can affect our machine learning models. To address this, we decided to evaluate our models using MCC and F-measure, as they are better suited for imbalanced datasets. MCC considers all values in the Confusion Matrix, providing a more accurate evaluation of the dataset.

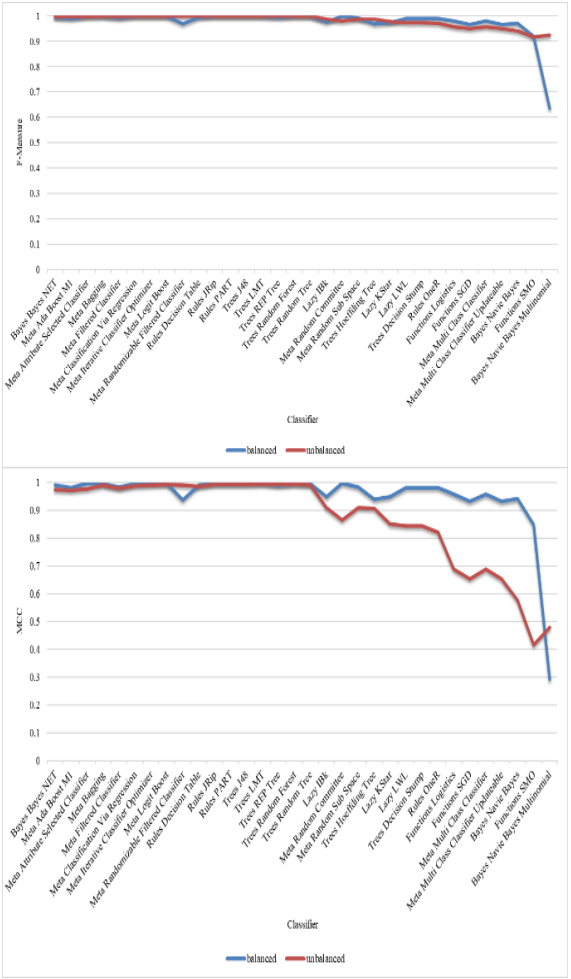

Figure 4. God Class: The MCC and F-measure values for balanced and imbalanced dataset

Figure 4 illustrates the performance of different machine learning models in detecting God Class using F-measure and MCC. When working with an imbalanced dataset, Trees LMT, Trees Random Forest, and Meta Logit Boost showed the highest MCC values, while Bayes Navie Bayes, Bayes Navie Bayes Multinomial, and Functions SMO had the lowest MCC values. After balancing the dataset, Trees LMT, Meta Logit Boost, Rules PART, Rules JRip, and Trees J48 exhibited the highest MCC values, while Bayes Navie Bayes Multinomial had the lowest. In terms of F-measure when working with an imbalanced dataset, Rules PART, Meta Bagging, Meta Randomizable Filtered Classifier, Meta Classification Via Regression, Rules Decision Table, Meta Filtered Classifier, and Meta Attribute Selected Classifier demonstrated the highest values. Conversely, Bayes Navie Bayes Multinomial had the lowest F-measure value. After balancing the dataset, Trees J48, Meta Bagging, Lazy IBk, Meta Filtered Classifier, and Meta Attribute Selected Classifier showed the highest F-measure values while Functions SMO had the lowest.

Figure 5 displays the results of various machine learning models in detecting Data Class, measured by F-measure and MCC. When using an imbalanced dataset, Trees LMT, Trees J48, and Rules PART exhibit the highest MCC values, while Functions SGD, Meta Multi Class Classifier Updateable, and Functions SMO show the lowest MCC values. However, after balancing the dataset, Meta Random Committee, Trees Random Forest, and Trees Random Tree demonstrate the highest MCC values, while Meta Multi Class Classifier, Functions Voted Perceptron, and Bayes Navie Bayes Multinomial exhibit the lowest MCC values. In terms of F-measure with an imbalanced dataset, Trees LMT, Trees J48, and Meta Random Committee have the highest values. Conversely, Functions SGD, Meta Multi Class Classifier Updateable, and Functions SMO have the lowest F-measure values. After balancing the dataset, Meta Random Committee, Trees Random Forest, and Trees Random Tree display the highest F-measure values. On the other hand,Trees Hoeffding Tree , Functions Voted Perceptron ,and Bayes Navie Bayes Multinomial have the lowest F-measure values.

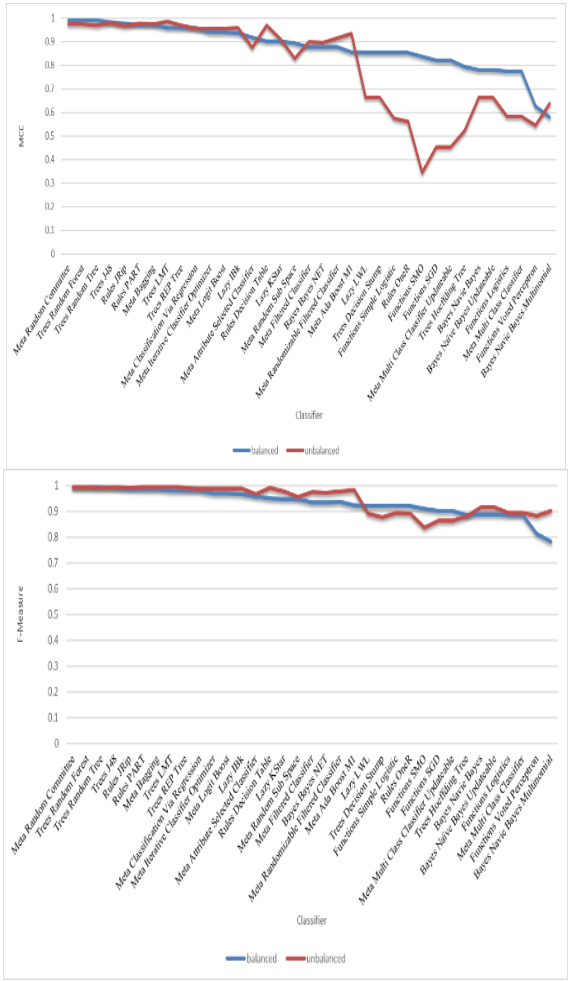

Figure 5. Data Class: The MCC and F-measure values for balanced and imbalanced dataset

Figure 6, 7, 8 and 9 compare our ML models with the related work based on their type of algorithms that have been used to detect bad smells in the related work. The comparison was based on three evaluation metrics: Accuracy, F-Measure, and ROC Area. Figure 6 compares the results of the imbalanced GC dataset with [11] and [12]. In terms of accuracy, study [11] achieved a score of 0.973 for Random Forest and Rule JRip, while study [12] achieved 0.9755 for Bayes Naive Bayes. Our Random Forest and Rule JRip models achieved an accuracy of 0.99. For F-Measure, study [11] scored 0.974 for Random Forest and Rule JRip, while study [12] achieved 0.9927 for Random Forest. Our Random Forest and Rule JRip models had a value of 0.99 as well. In terms of ROC measures, Study [11] and [12] achieved the highest scores of 0.989 and 0.981 for Random and Bayes Naive Bayes respectively. Our work saw ROC equal one for the Random Forest model.

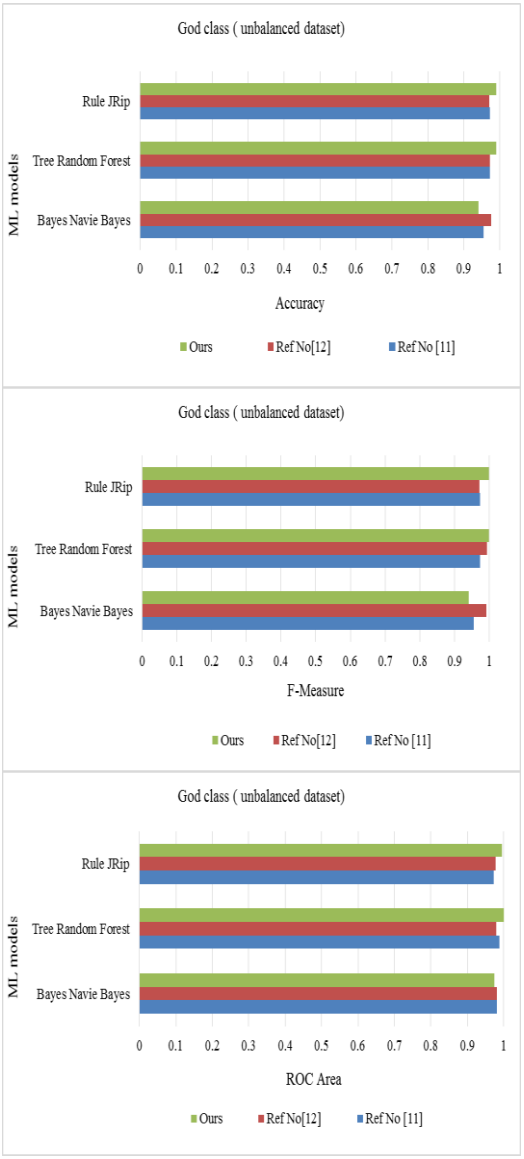

Figure 6. Compare our results with related works – God Class (imbalanced dataset)

We compared the results of our balanced GC dataset to those shown in Figure 7. Paper [11] had an accuracy of 0.973 for Random Forest and Rule JRip, while paper [12] achieved the highest accuracy (0.9755) for Bayes Naive Bayes. Our Rule JRip model had the highest accuracy at 0.999. In terms of F-Measure, paper [11] had the highest value (0.974) for Random Forest and Rule JRip, while paper [12] achieved a value of 0.9927 for Random Forest. Our models Bayes Naive Bayes and Random Forest both achieved 1 in this measurement. Lastly,

paper [11]'s ROC value was 0.989 for Random Forest and 0.981 for Bayes Naive Bayes in paper [12], while our Rule JRip model's ROC value was equal to 0.998.

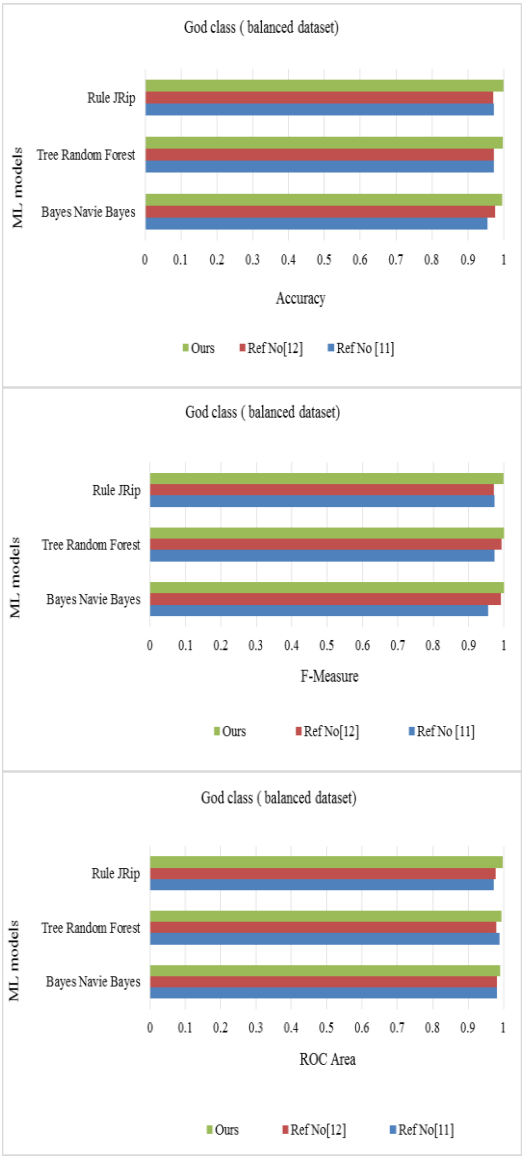

Figure 7. Compare our results with related works – God Class (balanced dataset)

In Figure 8, the results of the imbalanced DC dataset are shown. In terms of accuracy, study [11] and [12] achieved the highest accuracy of 0.99 for Random Forest while our models Random Forest and Rule JRip model achieved an accuracy of 0.999. For F-Measure, study [11] achieved the highest value of 1 for Random Forest while study [12] achieved a value of 0.992 for Random Forest and our model Random Forest achieved a value of 0.994. Lastly, in terms of ROC measures, study [11] achieved a highest value of 0.994 for Random Forest and study [12] achieved a highest value of 0.999 for Random Forest with our model achieving the highest value at 1.

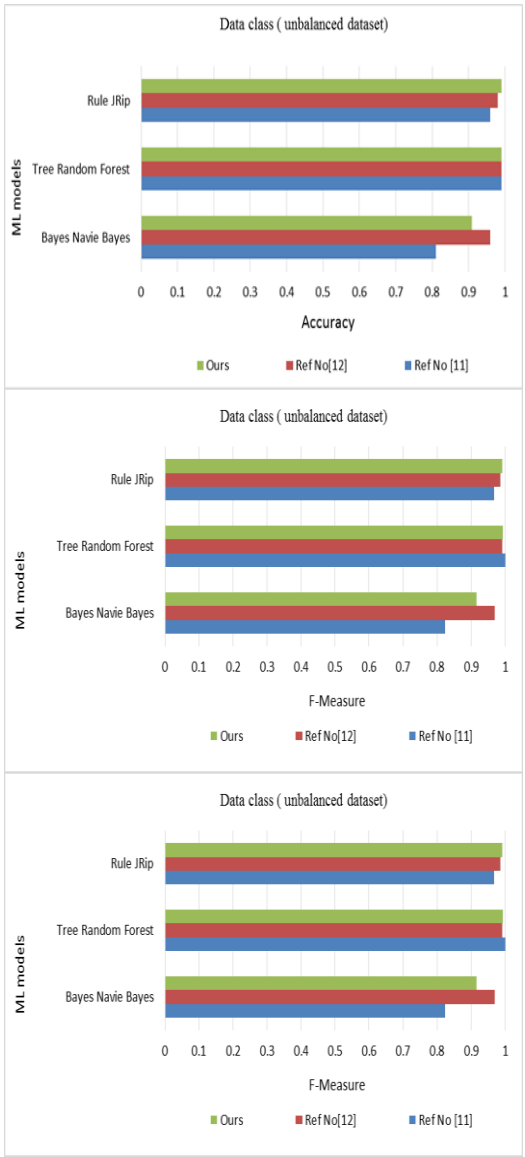

Figure 8. Compare our results with related works – Data class (imbalanced dataset)

In Figures 9, the findings of the balanced DC dataset are presented. The accuracy of study [11] and [12] was highest for Random Forest at 0.99 and 0.999 respectively. For F-Measure, study [11] achieved a highest value of one for Random Forest while study [12] achieved a highest value of 0.992 for Random Forest; our Random Forest model had the highest value at 0.999. Lastly, in ROC measures,

study [11] achieved a highest value of 0994 for Random Forest while study [12] achieved a highest value of 0.999; our Random Forest model had the highest value at 0.995

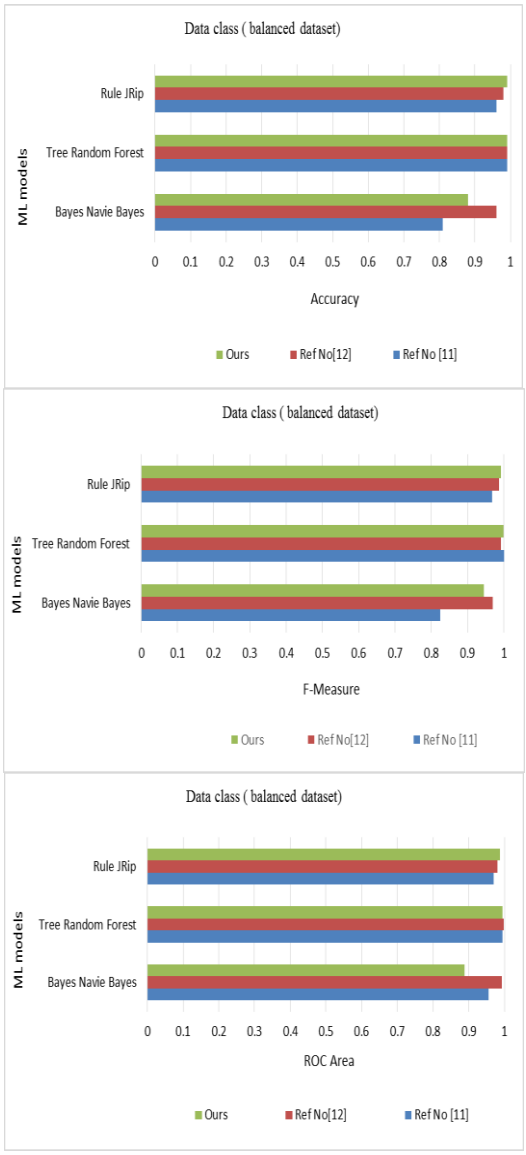

Figure 9. Compare our results with related works – Data class (balanced dataset)

VI. THREATS TO VALIDITY

Threats to construct validity concern the relationship between theory and observation. In our study, we used PMD to detect design smells, as it was the best option for the smells we were looking for. However, this could be biased by our imbalanced dataset, so we tested our models using both balanced and imbalanced datasets. To create a ground truth for machine learning, we developed criteria based on related work and PMD. We then used three metrics to detect God Classes and four metrics to detect Data Classes. PMD has a good performance, thus allowing us to effectively extract the metrics we used to detect design code smell.

The potential for our results to be impacted by external factors is a concern when it comes to internal validity. The rules used to detect God Class and Data Class smells may not be seen as such by a human expert or other tools. However, the rules we used were based on a rule-based PMD tool which has been proven to have good performance. To ensure accuracy, we employed 10-fold cross-validation which is known to have less bias in its estimation.

Threats to conclusion validity is linked to our ability to draw the right ones. One risk to the accuracy of these conclusions is connected to the datasets. The datasets were imbalanced, so F-measure and MCC values were used to evaluate our machine learning model. F-measure and MCC are known for being used with asymmetric datasets. To compare the results of the two models created from balanced and imbalanced datasets, we balanced our datasets and then compared the machine learning results.

External validity refers to the generalization of the results. We conducted a case study to analyze nine Apache projects written in Java, and the machine learning input was based on these projects. However, due to the limited scope of the study, we cannot extrapolate the results to other contexts.

VII. CONCLUSIONS AND FUTURE WORK

In this study, we conducted a case study to create a dataset which was then used to train our machine-learning models to detect design smells. The accuracy, Receiver Operating Characteristics Curve (ROC), F-measure, and Matthews Correlation Coefficient (MCC) values of the models were reported in order to demonstrate their performance. We conduct a case study using a balanced and imbalanced dataset to show how machine learning

classifiers could be affected while detecting God and Data Class smell. Ultimately, the level of prediction was comparable with other related studies, making many of our models suitable for prediction. Our future work will center on the development of a machine-learning model for the identification of additional design smells in software systems. To this end, we will be collecting a larger set of design smells by analyzing at least one hundred Apache projects (10 releases for each). Our objectives are to detect more type of design smells and make the extensive dataset available to the research community as a valuable resource.

REFERENCES

[1]. M. Fowler, "Refactoring: Improving the Design of Existing Code," 2000, DOI= http://www. martinfowler. com/books. html/refactoring, 2003.

[2]. A. Kaur, S. Jain, and S. Goel, "A support vector machine based approach for code smell detection," in 2017 International Conference on Machine Learning and Data Science (MLDS), IEEE, 2017, pp. 9–14.

[3]. N. S. Alves, T. S. Mendes, M. G. De Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," Inf. Softw. Technol., vol. 70, pp. 100–121, 2016.

[4]. M. BenIdris, H. Ammar, and D. Dzielski, "Investigate, identify and estimate the technical debt: a systematic mapping study," Available SSRN 3606172, 2020.

[5]. W. Cunningham, "The WyCash portfolio management system," ACM Sigplan Oops Messenger, vol. 4, no. 2, pp. 29–30, 1992.

[6]. V. Ferme, A. Marino, and F. A. Fontana, "Is it a Real Code Smell to be Removed or not," in International Workshop on Refactoring & Testing (RefTest), co-located event with XP 2013 Conference, sn, 2013.

[7]. F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui, "A bayesian approach for the detection of code and design smells," in 2009 Ninth International Conference on Quality Software, IEEE, 2009, pp. 305–314.

[8]. F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui, "BDTEX: A GQM-based Bayesian approach for the detection of antipatterns," J. Syst. Softw., vol. 84, no. 4, pp. 559–572, 2011.

[9]. N. Maneerat and P. Muenchaisri, "Bad-smell prediction from software design model using machine learning techniques," in 2011 Eighth international joint conference on computer science and software engineering (JCSSE), IEEE, 2011, pp. 331–336.

[10]. A. Maiga , N. Ali, N. Bhattacharya, A. Sabané, Y.-G. Guéhéneuc, G. Antoniol ,and E.Aïmeur , "Support vector machines for anti-pattern detection," in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 278–281.

[11]. F. A. Fontana, M. Zanoni, A. Marino, and M. V. Mäntylä, "Code smell detection: Towards a machine learning-based approach," in 2013 IEEE international conference on software maintenance, IEEE, 2013, pp. 396–399.

[12]. F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," Empir. Softw. Eng., vol. 21, pp. 1143–1191, 2016.

[13]. F. A. Fontana and M. Zanoni, "Code smell severity classification using machine learning techniques," Knowl.-Based Syst., vol. 128, pp. 43–58, 2017.

[14]. D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia, "Detecting code smells using machine learning techniques: are we there yet?," in 2018 ieee 25th international conference on software analysis, evolution and reengineering (saner), IEEE, 2018, pp. 612–621.

[15]. G. Czibula, Z. Marian, and I. G. Czibula, "Detecting software design defects using relational association rule mining," Knowl. Inf. Syst., vol. 42, pp. 545–577, 2015.

[16]. M. BenIdris, H. Ammar, D. Dzielski, and W. H. Benamer, "Prioritizing software components risk: Towards a machine learning-based approach," in Proceedings of the 6th International Conference on Engineering & MIS 2020, 2020, pp. 1–11.

[17]. F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia, and R. Oliveto, "Smells like teen spirit: Improving bug prediction performance using the intensity of code smells," in 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2016, pp. 244–255.

[18]. F. A. Fontana, E. Mariani, A. Mornioli, R. Sormani, and A. Tonello, "An experience report on using code smells detection tools," in 2011 IEEE fourth international conference on software testing, verification and validation workshops, IEEE, 2011, pp. 450–457.

[19]. S. M. Al Khatib, K. Alkharabsheh, and S. Alawadi, "Selection of human evaluators for design smell detection using dragonfly optimization algorithm: An empirical study," Inf. Softw. Technol., vol. 155, p. 107120, 2023.

[20]. M. Lanza and R. Marinescu, Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media, 2007.

[21]. S. Deeb, M. BenIdris, H. Ammar, and D. Dzielski, "Refactoring cost estimation for architectural technical debt," Int. J. Softw. Eng. Knowl. Eng., vol. 31, no. 02, pp. 269–288, 2021.

[22]. V. Lenarduzzi, A. Martini, D. Taibi, and D. A. Tamburri, "Towards surgically-precise technical debt estimation: Early results and research roadmap," in Proceedings of the 3rd ACM SIGSOFT International workshop on machine learning techniques for software quality evaluation, 2019, pp. 37–42.

[23]. "Apache Projects Directory." https://projects.apache.org/ (accessed Nov. 10, 2022).
[24]. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," ACM SIGKDD Explor. Newsl., vol. 11, no. 1, pp. 10–18, 2009.
[25]. I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, and M. Data, "Practical machine learning tools and techniques," in Data Mining, Elsevier Amsterdam, The Netherlands, 2005, pp. 403–413.
[26]. R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in Ijcai, Montreal, Canada, 1995, pp. 1137–1145.
[27]. Y. Liu, J. Cheng, C. Yan, X. Wu, and F. Chen, "Research on the Matthews correlation coefficients metrics of personalized recommendation algorithm evaluation," Int. J. Hybrid Inf. Technol., vol. 8, no. 1, pp. 163–172, 2015.