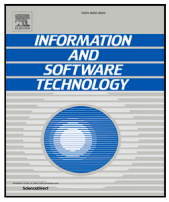
Information and Software Technology 143 (2022) 106736

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



A comparison of machine learning algorithms on design smell detection using balanced and imbalanced dataset: A study of God class

Khalid Alkharabsheh [a], Sadi Alawadi [e,*], Victor R. Kebande [d], Yania Crespo [c], Manuel Fernández-Delgado [b], José A. Taboada [b]

[a] Department of Software Engineering, Prince Abdullah bin Ghazi Faculty of Information and Communication Technology, Al-Balqa Applied University (BAU), Jordan
[b] CiTIUS, Centro Singular de Investigación en Tecnoloxías Intelixentes, Universidad de Santiago de Compostela, Santiago de Compostela 15782, Spain
[c] Departamento de Informática. Escuela de Ingeniería Informática, Campus Miguel Delibes, Universidad de Valladolid, Paseo de Belén 15, Valladolid 47011, Spain
[d] Department of Computer Science (DIDA), Blekinge Institute of Technology, 37179, Karlskrona, Sweden
[e] Department of Information Technology, Uppsala University, Box 337, 75105, Uppsala, Sweden

ARTICLE INFO

Keywords:
Software quality
Design smell detection
Machine learning
God class
Balanced data

ABSTRACT

Context: Design smell detection has proven to be a significant activity that has an aim of not only enhancing the software quality but also increasing its life cycle.
Objective: This work investigates whether machine learning approaches can effectively be leveraged for software design smell detection. Additionally, this paper provides a comparatively study, focused on using balanced datasets, where it checks if avoiding dataset balancing can be of any influence on the accuracy and behavior during design smell detection.
Method: A set of experiments have been conducted-using 28 Machine Learning classifiers aimed at detecting God classes. This experiment was conducted using a dataset formed from 12,587 classes of 24 software systems, in which 1,958 classes were manually validated.
Results: Ultimately, most classifiers obtained high performances,-with Cat Boost showing a higher performance. Also, it is evident from the experiments conducted that data balancing does not have any significant influence on the accuracy of detection. This reinforces the application of machine learning in real scenarios where the data is usually imbalanced by the inherent nature of design smells.
Conclusions: Machine learning approaches can effectively be used as a leverage for God class detection. While in this paper we have employed SMOTE technique for data balancing, it is worth noting that there exist other methods of data balancing and with other design smells. Furthermore, it is also important to note that application of those other methods may improve the results, in our experiments SMOTE did not improve God class detection.

The results are not fully generalizable because only one design smell is studied with projects developed in a single programming language, and only one balancing technique is used to compare with the imbalanced case. But these results are promising for the application in real design smells detection scenarios as mentioned above and the focus on other measures, such as Kappa, ROC, and MCC, have been used in the assessment of the classifier behavior.

1. Introduction

While the quality of software is one of the most important concern that attracts the attention of the field/community of software engineering, there is a constant need for maintaining this quality even though software complexity is being experienced quite often. Precisely, maintaining the quality of software involves exercising continuous actions, that help in the identification and detection of poor programming practices or bad designs in software systems, where these poor programming pieces are referred as "Design Smells" [1,2]. On the same note, detecting design smells is an activity that plays a significant role of supporting software developers in order to come up with better software design solutions. Design smells do not produce compilation

* Corresponding author.
E-mail addresses: khalidkh@bau.edu.jo (K. Alkharabsheh), sadi.alawadi@it.uu.se (S. Alawadi), victor.kebande@bth.se (V.R. Kebande), yania@infor.uva.es (Y. Crespo), manuel.fernandez.delgado@usc.es (M. Fernández-Delgado), joseangel.taboada@usc.es (J.A. Taboada).

https://doi.org/10.1016/j.infsof.2021.106736
Received 22 January 2021; Received in revised form 23 August 2021; Accepted 23 September 2021
Available online 8 October 2021

or run-time errors [2], however, they portray negative impacts on different software quality features, such as reliability, usability, changeability, and maintainability [3]. More often, the existence of design smells threaten the software life cycle, which leads to poor design outputs. Notably, the growing software complexity, the rapid advancements of software systems require more novel approaches, techniques, practices, and tools that can help to detect and identify the true positive design smells.

Existing works have shown that quite a number of approaches have been proposed that have had a focus on smell detection. For example, metric-based, rule-based approaches, [4–10] and machine learning based approaches [11–17].

These approaches (metric, rule) are constructed based on the design smell concepts that are mapped into a set of metric rules in order to identify software either through manual, semi-automatic, or automatic detection. A number of approaches that have been used in design smell research have been evaluated empirically. They have then ended up achieving higher accuracy, however, there exist some challenges that limit their adoption in industries, such as false positive and false negative ratios and the low agreements between them. In spite of that, some of these approaches have found ways of being implemented as commercial or open source detection tools [6,7,18–20].

In quest of overcoming the aforementioned challenges, the existing studies [11–14,21–23], have shown that machine learning techniques can be exploited effectively towards detecting design smells in software systems. Machine learning is a partial field of artificial intelligence that uses mathematical algorithms to give systems the capacity to learn without being programmed [24].

It is based on the aforementioned premise that the authors of this paper see the need to leverage a balanced dataset to as a way of testing whether this objective of design smell detection can be achieved. Consequently, the aspect of using a balanced dataset and different design smell detection tools presents a significant approach of design smell detection. In this context, the notion of utilizing a balanced dataset in this study, is owing to the fact that studies in [17] have shown that, models become more accurate when the best balancing technique is used. Studies such as [17] have shown that models become more accurate when the best balancing technique is used. However, by the inherent nature of design smells and the data are often highly imbalanced. Therefore, an empirical study comparing the behavior of the classifiers in both cases, when using an imbalanced dataset, gives insights on how much the improvement really is. In addition, the behavior of the classifiers should be observed from other indicators, such as Kappa, ROC, and MCC (see Section 3.3). These indicators are more appropriate for problem scenarios with imbalanced data, without losing track of the other indicators such as accuracy and F-measure that are usually better when using balanced data. Despite that, the improvement may not be substantial, which would be good news for the application of machine learning techniques in real scenarios of design smell detection. On the same note, despite taking a step of validating the detected smells, the uniqueness of this study is juxtaposed to also leverage 28 distinct classifiers, which uniquely based on the expected results, may depict a design smell detection behavior worth exploring.

On the one hand, this work aims empirically to investigate whether machine learning classifiers can effectively be used for God Class design smell detection, while on the other hand, the study does a comparison on the behavior of the classifiers using a balanced and imbalanced dataset from other indicators, such as Kappa, ROC, and MCC which are more appropriate for problem scenarios with imbalanced data. This, also is done without losing track of the other indicators such as accuracy and F-measure that are usually better when using balanced data. We highlight that despite the conclusions from our experiments, there exist other methods of data balancing besides SMOTE that we employed in this paper, also with the existence of other design smells. The application of those other methods may improve the results; however, in our experiments, SMOTE did not improve God class detection.

Therefore, the main contributions of this paper can be summarized as follows:

- We conduct an experiment using a balanced and imbalanced dataset to show how machine learning classifiers could be leveraged during God Class design smell detection.
- We investigate the influence of data balancing on God Class design smell detection.
- We provide a large dataset formed from 12,587 classes of 24 software systems that include 1958 God Classes that are manually validated. Also there is a Reduction of false negative with the original automatic labeling system based on five detection tools and reducing the false positive by manual validation on the results on the first automatic labeling stage.
- We provide a replication package that involves raw data, scripts, and all related material for the replication of the study. [25].

Section 2 presents related works on machine learning for design smell detection and the influence of dataset balancing on the accuracy of the machine learning models. Section 3 describes the study goals, research questions, and main hypotheses, as well as the study design that includes context selection, data collection, and data analysis. Next, Section 4 provides an analysis of the obtained results while Section 5 presents a replication of experiments using a new dataset. Section 6 discusses the main threats to the validity of the study. Finally, the paper concludes with Section 7 by providing a conclusion and a mention of future work.

2. Related work

In this section, related work that is focused on machine learning techniques for design smell detection and the influence of data balancing on the learning techniques behavior concerning smell detection activity is discussed.

2.1. Machine learning approaches for design smell detection

From a cursory view, [13] proposed a concept that addressed an amalgamation of a machine learning approaches with Object-Oriented (OO) metrics. The dataset that was used included two software systems that were analyzed using a prototype. The outcome showed that learning decision trees were utilized in the detection of two types of design flaws (Large Class & Long Method) on two software systems. Also, [14] investigate the prediction of bad smells (Lazy Class, Feature Envy, Middle Man, Message Chains, Long Method, Long Parameter List, Switch Statement) using seven machine learning techniques. In their study, the dataset involved 27 design metrics and seven bad smells mentioned that were collected from a set of software that have design smells detected from previous studies. Furthermore, their prediction model was validated by statistical significance testing. In another research, the Bayesian belief networks have been used in the named BDTEX strategy to detect three types of anti-patterns (Blob, Functional Decomposition & Spaghetti Code) on two Java projects [12] and compare the results with the obtained by the detection tool based in DECOR method, used as the reference of the state of the art in detection. Next, [26,27] introduce an approach to detect four different types of anti-patterns (Blob, Functional Decomposition, Spaghetti Code & Swiss Army Knife) based on Support Vector Machine (SVM), in three software systems and compare the results with the detection tool DETEX which instantiate the DECOR method, used as the reference of the state of the art in detection. In [26], the authors focused on SVM and object-oriented metrics, while [27] they have taken into account the feedback of practitioners in the detection strategies and compare the results with BDTEX and DETEX. More recent research by [28] presented a non-intrusive machine learning approach (NiPAD) that is based on system performance metrics for detecting and classifying anti-patterns using five machine learning techniques. They experiment with one application and determines SVMlinear as the algorithm with the best behavior. The obtained results showed that the proposed approach

has the ability to identify the One-lane Bridge performance anti-pattern with high accuracy detection. Another relevant work [21] conducts a large-scale experiment that compared sixteen machine learning techniques to detect four types of code smells (God Class, Feature Envy, Long Method and Data Class) using five different detection tools (two tools for God Class) as "advisors" to perform an automate smell detection as initial labeling technique. The designed experiment was able to give a validation of 74 software systems, in addition to a sample of 1986 code smells validated by human experts as second step in labeling which include 493 God Class, 430 Data class, 517 Feature Envy, 546 Long Method. The obtained result showed that most of the chosen techniques have more than 95% accuracy and F-measure. From this basis, [29] conducted a replication study of the exploration by [21] in order to identify the current state of machine learning approaches in design smells detection. The results of that study indicated the need for more investigation in the learning methodologies, construction of dataset, and the set of metrics used to present the smelly and not smelly classes.

Finally, in [17], the authors performed an empirical study that compares the performance of machine learning-based (Naive Bayes) and heuristic-based approaches, specifically, metric-based approaches in terms of design smell detection. In their study, the dataset involved 17 metrics and 11 code smells gathered from 13 software systems. The obtained results show that heuristic-based approaches achieve slightly better performance compared with machine learning techniques. However, there is a need for further investigation and researches to improve the accuracy and effectiveness of both approaches on design smell detection. One of the aspects that deserve more study in this type of problem, i.e design smell detection, are the indicators of good classifiers performance. Accuracy and F-measure should not be the only ones. This involves working with a set of elements where only a few must be identified (as smelly in this case). It can be quite common to obtain high accuracy values. Accuracy alone is not a good indicator when working with imbalanced categories. However, Cohen's Kappa can be a good complementary indicator when accuracy is not good because of the large imbalance. Nevertheless, Cohen's Kappa, by definition, is always higher with balanced data. Hence, it is also necessary to combine the behavior analysis with ROC and MCC indicators (see Section 3.3), which can be used even if the categories are of very different sizes. These indicators are widely used in medicine to assess the goodness of a test that detects the disease, which is a similar case to that of detection design smells.

2.2. Influence of data balancing on machine learning model concerning design smell detection

The issue of data balancing in machine learning has attracted the attention of the research community in the recent past. Several studies in the existing literature have tackled this topic [30–33]. In this section, our focus is mainly on the studies that concern data balancing in machine learning based design smell detection.

Research in [34], examines the impact of balancing and unbalancing datasets on the behavior of machine learning techniques regarding code smell detection. Three types of machine learning techniques SVM, J48, and Random Forest have been used in the experiment to detect seven types of design smells, in four versions of the Eclipse software system. The dataset is balanced based on a popular strategy used in [35] while one-third of classes are positive, the remaining is negative. The results show that the performance of the constructed models using J48 and SVM techniques does not improve after data balancing, while a slight improvement on the RandomForest model is observed. Their conclusion is that data balancing does not have a dramatic influence on the behavior of machine learning algorithms. Next, a study by [16,17] investigates the role of data balancing techniques in machine learning on design smell detection. The authors conducted a large scale study that compares the performance of five data balancing techniques, that involves the SMOTE technique against unbalancing data. In this study, the experiment was conducted using a set of five machine learning algorithms on a dataset of 13 open source systems, that were analyzed and manually validated to detect 11 types of design smells. The obtained results show that the behavior of machine learning techniques has not significantly improved using data balancing despite the accuracy of learned models for design smell detection that used existing metrics which slightly changed, especially while SMOTE technique was used. Eventually, they conclude that it is necessary to include new metrics related to the software systems in order to increase the accuracy of models.

Recently, in [15,36], the authors explore the influence of including new software metrics, such as size categories and domain on the God class detection. The experiments conducted using eight machine learning algorithms on a large dataset formed by 24 software systems. Also, they examine the impact of using SMOTE data balancing technique on the performance of algorithms. The results show that the performance of machine learning algorithms improved using size and domain and can be exploited effectively in improving design smell detection. On the other hand, after data balancing, the behavior of algorithms slightly improved, however, they did not find conclusive empirical evidence on the usefulness of data balancing.

To overcome the limitations of previous works and their findings, in this paper, from the one hand, we confirm the results concerning exploiting machine learning approaches on design smell detection, and from the other perspective, we generalize the conclusions related to the influence of data balancing on design smell detection. For this purpose, the proposed work differs from the previous works in the following aspects:

- A large set of different machine learning techniques (28 classifiers) have been used to conduct the experiment.
- A set of five different design smell detection tools have been used initially as advisors to automatically label the original dataset to construct the God Class dataset.
- A manual labeling of design smells has been applied by experts to the whole dataset.
- The results have been compared by replicating the experiment on two different datasets of the previous studies.

3. Empirical study definition and design

As mentioned before, machine learning techniques are widely exploited to detect design smells. However, despite a concentration in this of research field, further investigation are still needed, especially in the comparison between using or not using a balanced dataset. For studying this aspect, the aim of this work is to (i) investigate whether machine learning based approaches can be effectively leveraged for God Class design smell detection, and (ii) how avoiding dataset balancing can be of influence to the accuracy/performance of design smell detection. The objective of this study is defined by the GQM method, widely used in software engineering research for goal setting [37,38] as follows:

Analyze Set of classes.

With the purpose of Evaluation.

With respect to the efficiency of machine learning techniques to detect God Class design smell using a balanced in front of imbalanced dataset.

From the point of view of researchers.

In the context of Software hosted in open source repositories.

According to this objective, we derive the following research questions and a working hypothesis.

RQ1 *To what extent can machine learning techniques be effectively leveraged for God Class design smell detection?*

RQ2 *To what extent can avoiding data balancing have an influence to the accuracy of God Class design smell detection?*

To this end, the following null hypotheses have been formulated:

Hypothesis1 *Machine learning techniques cannot be effectively leveraged for God Class design smell detection.*

Hypothesis2 *Avoiding data balancing will not influence the accuracy of God Class design smell detection.*

3.1. Context selection

The context of this study consists of the target software systems, God Class design smell, and the set of machine learning techniques used for God Class detection.

3.1.1. Target systems in the first set of experiments

We used a dataset that was previously constructed by our team, that has also been used in other research publications with different formats [15,39]. The dataset consists of 24 open-source software systems having different domains, sizes, and categories. The selected open-source systems have been obtained from the Github[1] and SourceForge[2] software repositories, which are well-known in the context of open-source systems. All systems were written in Java language which is one of the most recognized object-oriented languages in the context of design smell detection, where a total of 12,587 classes were analyzed.

Table 1 reports the main characteristics of the chosen software systems: software name and version, Number of Classes (NOC), Number of Methods (NOM), and the Total Lines of Code (TLOC). In order to better understand the machine learning process, we increased the number of selected software projects in the experiment to get a high number of classes so that we can be able to generalize the study results. The whole dataset was included in the replication package.

3.1.2. God class design smell

There exist several types of design smells that have a possibility of being detected in software systems. These smells vary in scope, influence, and frequency of occurrence. Important to note is that the focus of this study is inclined towards God Class detection. God Class has in different situations attracted great attention of the research community based on the existing literature [40–46]. Moreover, previous work on systematic mapping on design smell detection [1], has shown that God Class is one of the most detected design smells in the software system and it has often been detected using a wide set of detection tools. Also, God class smell has a negative influence on a wide set of software quality features such as maintainability, understandability, complexity, readability, flexibility, evolvability, performance, reusability, and stability.

God Class is a class-level smell [47] that describe the case of a class tends to do several functions and has many responsibilities, i.e. most of the tasks are centralized in this class. It tends to be very large in terms of Lines of Code (LOC) and it is also very complex. Based on the existing literature that has extensively been explored, God Class has been referred as the Blob anti-pattern [3], and the large class bad smell [48]. According to [49] and [50], the presence of the God Class in a particular class is an indicator to finding other types of design smells such as Feature Envy, Data Class, God Method, and Duplicate Code. In order to detect God Class, it is important to explore the inside of the class. According to [51] in their classification of design smells God Class belongs to “Bloaters” group that describe parts of code that have a large size and are hard to maintain.

Table 1
Characterization of the selected software systems that includes Number of packages (NOP), Number of classes (NOC), Number of Methods (NOM), and Total Lines of Code (TLoC).

Project name & version	NOP	NOC	NOM	TLOC
AngryIPScanner-3.0	20	270	1049	19,965
Apeiron-2.92	6	62	641	8908
checkstyle-6.2.0	15	277	1267	41,104
DigiExtractor-2.5.2	10	80	412	15,668
Freemind-1.0.1	50	782	5987	106,396
FullSync-0.10.2	22	169	1272	24,323
GanttProject-2.0.10	52	621	5004	66,540
JasperReports-4.7.1	110	1797	15,645	350,690
jAudio-1.0.4	38	416	4257	117,615
Java graphplan-1.0.7	17	50	455	1049
JCLEC-4.0.0	33	311	1379	37,575
JDistlib-0.3.8	12	78	924	32,081
JFreeChart-1.0.X	37	499	7989	206,559
JHotDraw-5.2	12	151	1474	17,807
KeyStore Explorer-5.1	54	384	2266	83,144
Lucene-3.0.0	103	606	3837	81,611
Matte-1.7	65	603	4090	52,067
Mpxj-4.7	25	553	12,574	261,971
OmegaT-3.1.8	106	716	4472	121,909
Plugfy-0.6	10	28	91	2337
pmd-4.3.x	90	800	5256	82,885
sMeta-1.0.3	14	222	1308	30,843
SQuirreL SQL Client-3.7.1	249	20,495	1138	71,626
xena-6.1.0	281	1975	25,163	61,526
Total	**1431**	**12,587**	**127,307**	**1,896,199**

3.1.3. Machine learning techniques

A comparison of 28 machine learning algorithms has been conducted, These 28 classifiers were previously used in the context of design smell detection [21–23]. As shown in Table 2, these classifiers belongs to 11 different families [52–54], such as Naive Bayes (NB), Decision Trees (DT), Support Vector Machines (SVM), and Neural Networks (NN) etc.

Furthermore, to conduct this experiment, the selected classifiers requires training data that involve classified instances. To this end, in our experiment, we used a binary label to classify the dataset. For each classifier, there are one or more parameters that should be tuned.

Tuning classifier parameters have a robust effect on their performance. For this purpose, to select the most significant parameters for each classifier and the parameter values, we checked and reviewed several documentations, such as [55], which include a set of learning classifiers selected from the R[3] statistical computing language.

3.2. Data collection

To answer the research questions, a source code of software systems is needed to enable the extraction the set of related metrics, as well as detecting the God Class design smell, and then, formulating these data in the appropriate format to be input source for machine learning classifiers. For this purpose, in this section we describe in detail the tool that was used to collect the set of metrics and the set of design smell tools used in God Class detection.

3.2.1. Metric extraction tool

Several tools have been developed to analyze source code and to compute different measurements (metrics) concerning various quality dimensions, such as size, complexity, coupling, cohesion, and inheritance. This activity of computing code metrics can be developed in a standalone tool or besides other activities, such as design smell detection, prioritization, or refactoring in the same tool. In this work,

[1] https://github.com/github.
[2] https://sourceforge.net/.
[3] http://r-project.org.

Table 2
Machine learning techniques considered in this work grouped by their families.

No.	Family	Classifiers
1	Discriminant analysis (DA)	Linear Discriminant Analysis (LDA) Quadratic Discriminant Analysis (QDA)
2	Naive Bayesian (NB)	Multinomial Naive Bayes (MNB) Complement Naive Bayes (CNB) Gaussian Naive Bayes (GNB) Bernoulli Naive Bayes (BNB)
3	Neural Networks (NN)	Multi-Layers Perceptrons (MLP)
4	Support vector machines (SVM)	Support Vector Machine (SVM) Linear Support Vector Machine (LSVM) Nu-Support Vector Machine (Nu-SVM)
5	Decision trees (DT)	Decision Tree (DT)
6	Boosting (BST)	Gradient Boosting (GB) Cat Boost (CBoost) Light GBM (LGBM) eXtreme Gradient Boosting with RandomForest (XGBRF) eXtreme Gradient Boosting (XGB) AdaBoost (AB)
7	Bagging (BAG)	Bagging
8	Random Forests (RF)	Random Forest (RF) Extra Trees (ET)
9	Nearest neighbor methods (KNN)	K-Nearest Neighbors (K-NN) Nearest Centroid (NC)
10	Gaussian Process (GP)	Gaussian Process (GP)
11	Linear approaches	Ridge Logistic Regression (LR) Perceptron Passive Aggressive (PA) Stochastic Gradient Descent (SGD)

we have chosen RefactorIt[4] to analyze the selected software systems and extracted the required metrics. We decide to use an external tool (not of the five detection tools used as advisors in first automatic labeling strategy) to avoid any effect on the detection results of God Class.

RefactorIt is an open-source tool used to examine the software systems implemented in Java based on metrics and semantic rules. Using RefactorIt, we are able to compute 30 metrics for projects, packages, classes, methods, types, members, and constructors. It is available in two versions: standalone or integrated with eclipse [56]. A set of 16 object-oriented metrics have been computed for each software systems in the dataset. The set of metrics related to different levels and quality dimensions as can be seen in Table 3. The set of chosen metrics are common in the literature [12–14] for design smell detection.

3.2.2. Design smell detection tools

A large set of tools has been proposed for design smell detection according to the literature in [57] in order to help software developers towards their enhancement of software quality. These tools have various features as follows: Metric tools-which checks if it is available, open source or not, working environment (standalone or plug-in), supporting different languages, detection technique, and are designed to detect different types of design smells. From this study, to select the candidate tools, we followed a set of criteria that include: *supporting Java*, *available*, *detecting God Class*, and it should have a *high detection precision*. According to the criteria, the results shows a large list of tools and prototypes. Therefore, we decide to avoid prototypes and only focus on fully automatic tools. From this list, we have chosen a collection of five tools that include: *iPlasma*, *DECOR*, *JDeodorant*, *PMD*, and *Together*. This set of tools is represented as the most cited works in the context of design smell detection according to our previous work in [1]. A brief description of the aforementioned tools is given in the next paragraphs.

DECOR [7] is an approach for specifying and detecting 6 types of design smells/antipaterns based on a mixture of semantic and metric rules. Definition of design smell is expressed using a custom language called Domain Specific Language (DSL). Decor uses the design smell definition expressed in DSL to generates the detection code automatically. DETEX is the name of an instantiation of the DECOR approach.

iPlasma [6] is an open source integrated environment for software systems quality analysis. It calculates 80 metrics and detects 21 different types of design smells for software systems developed using Java and C++. Also, it can detect God Class from the source code by metric-based strategy that incorporates three well-known metrics.

JDeodorant [18] is an Eclipse plug-in tool that can detect automatically 5 types of design smells in software systems implemented in Java. In addition to the design smell detection activity, it can resolves the detected problem by applying effective automatic refactoring. JDeodorant uses clustering algorithm to find the God Classes in software systems that leads to apply the "Extract class" refactoring operation.

PMD [20] is developed as standalone or plugin environment to detect 3 types of design smells in software systems implemented in Java, Javascript, and other languages. This tool is mutual with iPlasma in the metrics combination that used to detect God Class but with different threshold values. In addition, it is the only tool that has a command-line interface while the rest require to be used through a graphical user interface (GUI).

Together [19] is a commercial Integrated Development Environment (IDE) designed to support software architects and developers, and incorporates an automated tool for code smell detection. The tool can detect 11 types of design smells and compute 55 different quality metrics for software systems implemented in Java and C++. Together can detect God Class from source code or UML diagrams using a combination of different metrics.

Table 3
Definition of chosen metrics.

No.	Metric	Definition	Granularity level	Dimension
M1	TLOC	Total Lines of Code	Project	Size
M2	NCLOC	Non-Comment Lines of Code	Project	Size
M3	CLOC	Comment Lines of Code	Project	Size
M4	EXEC	Executable Statements	Project	Complexity
M5	DC	Density of Comments	Project	Complexity
M6	NOT	Number of Types	Package	Complexity
M7	NOTa	Number of Abstract Types	Package	Complexity
M8	NOTc	Number of Concrete Types	Package	Complexity
M9	NOTe	Number of Exported Types	Package	Complexity
M10	RFC	Response for Class	Class	Coupling
M11	WMC	Weighted Methods per Class	Class	Complexity
M12	DIT	Depth in Tree	Class	Inheritance
M13	NOC	Number of Children in Tree	Class	Inheritance
M14	DIP	Dependency Inversion Principle	Class	Coupling
M15	LCOM	Lack of Cohesion of Methods	Class	Cohesion
M16	NOA	Number of Attributes	Class	Size

3.3. Data analysis

In this section, we give a description of the methodology that has been employed throughout this paper to analyze the collected data, and in order to answer the research questions. Several stages have been used to illustrate how the methodology has been applied and an explanation is given further on with a representation that is shown in Fig. 1.

Stage1: Dataset building. After obtaining the software systems from the repositories, we used the metrics tool "RefactorIt" to analyze them and extract the set of required metrics. As mentioned in Section 3.2.1, a set of 16 metrics have been extracted for each software

[4] http://RefactorIt.sourceforge.net.

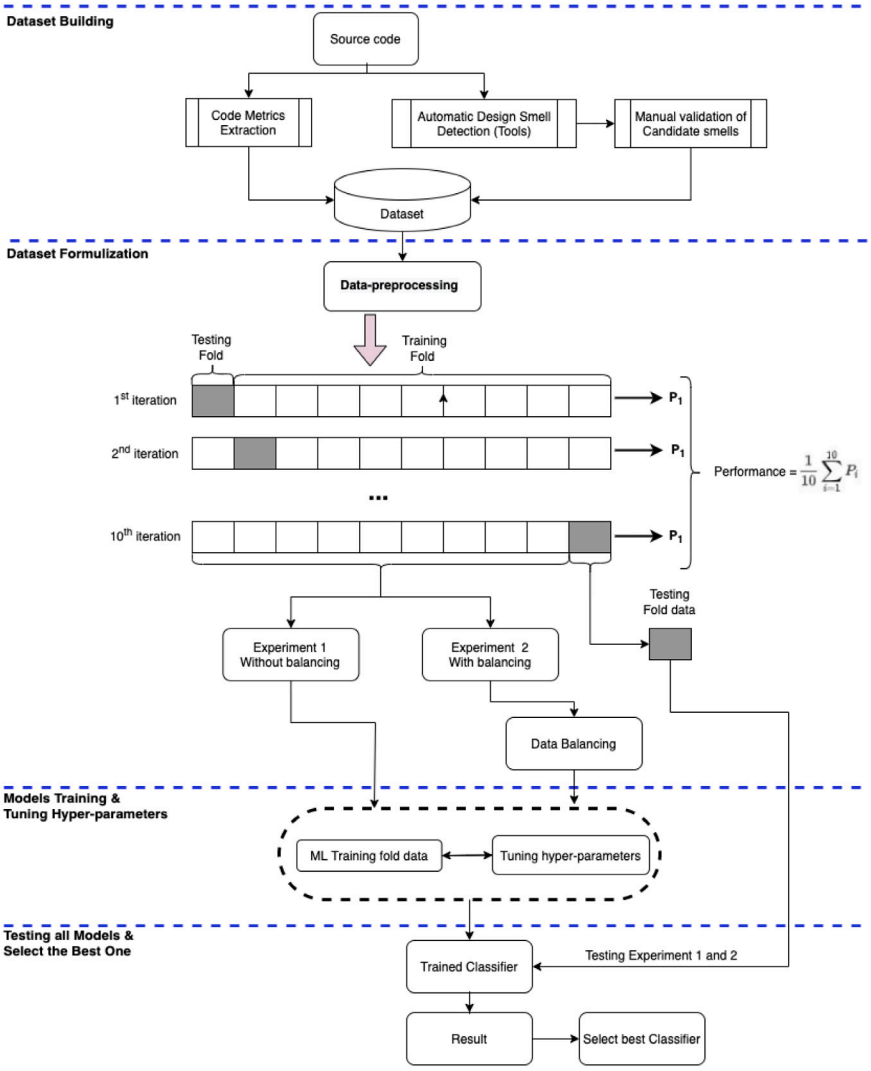


Fig. 1. Proposed methodology.

system, and the group of detection tools (iPlasma, Decor, JDeodorant, PMD, and Together) was used as automatic early advisors for God Class detection.

The strategy that was followed, was to label the detection tools result as follows: If one tool or more detects the class as God Class, then the label will be (1), otherwise (0). This strategy will increase the number of detected smells God Classes in the dataset reducing the probabilities of false negative. Afterwards, we applied a manual validation process on the God Class list detected by one or more of the selected tools. A group of three experts with doctorate degree in information technology (software engineering and computer science) have excellent experience with object-oriented programming, and very good knowledge of design smells accomplished the task. To label the God classes, the following strategy was followed: If two or more of the experts identified the class as a God Class, the class was taken to be God Class and the label given was (1), otherwise (0), with this manual validation the probability of having false positives is reduced. Next, the set of metrics and the manual validation results have been formulated in rows. Each row consists of 16 metrics labeled M1 to M16 plus a label that presents the manual validation results as a binary decision to show if it is God Class or not.

Stage2: Dataset preparation. In this stage, we conducted data preprocessing steps to fill the missing values with the corresponding feature average, and normalize the data in range [0, 1] because there are some algorithms cannot handle negative value. Then the prepared data was used as an input source to train, validate, and test the machine learning classifiers used in this study. The training process required enough data samples in order to capture the data pattern behavior and guarantee a very good learning process, while the validation step used for tune the classifiers hyper-parameters by selecting the suitable values that can fit with the ML problem. Finally, the evaluation of the ML can be performed in the testing phase, which give an indication how good the trained classifier is. Next, a feature selection process was conducted to avoid the non-significant metrics from the whole set (M1 to M16) and which may negatively influence the accuracy and performance of the model. For this purpose, we used the R package Rminer [58], that allows to calculate the importance of each feature for a given classifier or regressor, providing a wide list of available models. Specifically, we calculated these importances for the `rpart` classifier, selected by its speed and easy configuration (it has no tunable hyper-parameter). The classifier choice should not condition the feature importance, that should not depend on the classifier used to calculate it, i.e., the importance is expected to be similar using different classifiers. In general, it is expected to detect a few numbers of God Class design smells compared to the total classes that will be analyzed.

The detected numbers of God Classes have been distributed between the training, validation, and testing sets. The low fraction of smells in these sets leads to an imbalanced classification problem in what

Table 4
List of the classifiers, with their tunable hyper-parameters and their tried values.

Classifier	Hyperp. (values)	Classifier	Hyperp. (values)
LGBM	lr = 0.05, n_estimators = 200 num_leaves = 300, max_depth = 25	GB	lr = 0.01, max_features = 7 max_depth = 10 min_samples_split = 200 min_samples_leaf′ = 50
XGB	gamma = 0.4, reg_lambda = 1, reg_alpha = 0.1 lr = 0.25, n_estimators = 100, max_depth = 17	Ridge	alpha = 0.001, solver = lsqr
CBoost	max_depth = 8, lr = 0.01 l2_leaf_reg = 10	LDA	solver = svd
RF	n_estimators = 500, criterion = entropy max_depth = 300, min_samples_split = 4 min_samples_leaf = 7	NB	–
Bagging	criterion = entropy, max_depth = 20 max_features = sqrt, min_samples_leaf = 3 min_samples_split = 6	LR	C = 90, solver = newton-cg
XGBRF	gamma = 0.2, max_depth = 19, lr = 0.01 min_child_weight = 3, n_estimators = 10 num_parallel_tree = 10, reg_lambda = 0.01 reg_alpha = 0.1	SGD	alpha = 0.0005, lr = 0.001 l1_ratio = 0.0004
ET	max_depth = 400, min_samples_split = 4 n_estimators = 500	LSVM	C = 100.0, loss = hinge
AB	lr = 0.9, n_estimators = 800	PA	C = 0.01
MLP	activation = tanh, lr = 0.04 hidden_layer_sizes = (29, 40, 2)	Percep-tron	alpha = 0.0005, penalty = l1
DT	criterion = entropy, min_samples_split = 5 max_depth = 20, min_samples_leaf = 3	QDA	reg_param = 0.001
Nu-SVM	gamma = 10.0, nu = 0.02	BNB	–
SVM	C = 100.0, gamma = 0.1, kernel = rbf	GNB	–
KNN	n_neighbors = 4	CNB	–
GP	length_scale = 2, nu = 2.6	MNB	alpha = 1e−05, fit_prior=False

low classification performance is expected. Therefore, we decided to conduct two different experiments, without dataset balancing (Experiment 1) and with dataset balancing (Experiment 2) to check whether the performance of the classifiers will be influenced or not, therefore, reflecting on the detection accuracy through performance indicators previously mentioned.

The Synthetic Minority Oversampling Technique (SMOTE) [58] has been used in experiment 2 to solve the imbalanced dataset problem, where the distribution of a class is heterogeneous with other classes. Balancing both classes (God Class/Not God Class) in the training set was performed. The parameter K (number of nearest neighbors) in SMOTE was tuned by executing KNN on the training set with values K=1,2,3, . . . ,12. The value of K that provides the highest performance is selected for SMOTE.

Stage3: Models training and hyper-parameters tuning. As mentioned previously, 10-folds cross-validation were generated to train, hyper-parameter tuning, and evaluate the models. Most of the classifiers have a set of hyper-parameters that need to be tuned-which also have a significant influence on the behavior and performance of the machine learning model. Therefore, we used 10-fold cross-validation to find the best suitable value for each parameter based on the description for each classifier [55]. For each algorithm, the hyper-parameters were tuned using the obtained values reported in Table 4. The selected final values for the hyper-parameter are those that maximize the average performance over the validation sets.

Stage 4: Testing all models and selecting the best classifier. Finally, each trained classifier has been evaluated over testing data fold in terms of both accuracy and performance using different metrics such as Cohen's Kappa, ROC area, precision, F-measure, and Matthews Correlation Coefficient (MCC), respectively.

According to [1], the selected performance measures are well-known and widely used for this purpose. Accuracy is the ratio of instances that are correctly classified (true positive and true negative).

Table 5
Interpretation of the Cohen kappa values.

Kappa value	Agreement
$k < 0.20$	Poor
$0.21 \leq k < 0.40$	Weak
$0.41 \leq k < 0.60$	Moderate
$0.61 \leq k < 0.80$	Good
$0.81 \leq k \leq 1.00$	Very Good

In our case, it is the ratio of classes that are predicted correctly as (God Class/Not God Class). This performance indicator is not enough to decide that the classification model is good, especially, if the classified instances is imbalanced. In this study, we have studied an comparing performance measures when using imbalanced and balanced datasets. In the context of this study, accuracy is computed by dividing the number of correct prediction to the total number of prediction, multiplied by 100%.

Cohen's Kappa [59] is another essential performance indicator that was used in this study. Cohen kappa is used to measure the agreement between the true label of instance and the one predicted by the selected classifier. The kappa values range from −1 to 1. The interpretation of Cohens kappa values is shown in Table 5 where kappa result of all classifiers is shown in percentages.

Receiver Operating Characteristic (ROC) area [60] is a good measure to visualize the performance of an classifier. It is a plot that compares the sensitivity (true positive rate) of the classifier against the specificity (false positive rate). Table 6 show the interpretation of ROC values. When the value is close to 1, the classifier performance is better.

Precision indicates the portion of correctly detected God Classes within the set of all detected God Classes. The precision result range was between 0 and 1.

Table 6
Interpretation of the ROC area.

ROC value	Interpretation
$0.5 < ROC \leq 0.6$	Fail
$0.6 < ROC \leq 0.7$	Poor
$0.7 < ROC \leq 0.8$	Fair
$0.8 < ROC \leq 0.9$	Good
$0.9 < ROC \leq 1$	Excellent

F-measure is a classical method of evaluating the performance of the classifier. It combines the precision and recall into one metric. We used the F-measure for a trade-off between the classifiers.

The final assessment indicator is the Matthews Correlation Coefficient (MCC) [61]. It is used to measure the binary quality between the observed and predicted classifications. The MCC range from −1 to 1 in which the higher the value (1) of the coefficient, indicates the stronger the correlation (i.e. agreement between prediction and observation). If the value is zero, then no correlation, and a disagreement between prediction and observation when the value is (−1).

In the conducted experiments, the Wilcoxon rank-sum test [62] was used to assess separately the difference between the accuracy, kappa, F-measure, precision, ROC, and MCC values achieved by the 28 classifiers with and without data balancing. The null hypothesis of the accuracy test is: The accuracy comes from distributions with equal mean with and without data balancing (i.e. classifiers have the same behavior on God Class detection with and without data balancing), the same hypothesis is tested for kappa, F-measure, precision, ROC and MCC. When the p-value is less than 0.05, we reject the null hypothesis, so the difference between cases is statistically significant. This measure is widely used for this purpose according to existing literature [14,29,63–65].

All necessary information for replicating the work, such as data description, raw data, and scripts is available in the replication package [25].

4. Results analysis

The obtained results that were used to statistically study the hypotheses in this paper are discussed in this section. In the first subsection, we analyzed and discussed the results of the selected detection tools and manual validation that we used to assign a label of the classes (Dataset building), while in the second, we focus on the performance of the classifiers (Dataset formulation, tuning hyper-parameter, and testing the models) that will answer the formulated research questions.

4.1. God class detection

Table 7 shows a summary of the God Class detection results using automatic detection tools (GC-tool) and the manual validation (GC-manualvalidation) for each software system according to the labeling strategy mentioned in stage one (Dataset Building) of the proposed methodology in Section 3.3. According to the initial tool labeling strategy, a set of 1958 God Classes have been detected in the software systems, which is the maximum number of smells detected in the whole dataset. After applying the labeling strategy of the manual validation on the detection tools results, only 485 God Classes (out of 1958) were found and representing 4% of the whole dataset (12,587). This ratio is normal and confirms previous conclusions in the literature about the low degree of agreement on design smell detection between different evaluators (humans, tools, and human vs. tools) [40,41]. This combined labeling method helps in first reducing the probability of false negatives, and after that, reducing the probability of false positives, as mentioned before. The largest number of God Classes were detected using tools in Jasperreport 4.7.1 while the lowest number in Plugfy-0.6. On the other hand, using manual validation, the largest number of God Classes was detected in Xena 6.1.0 while no God Class was detected in the group of (AngryIPScanner-3.0, checkstyle-6.2.0, FullSync-0.10.2, JCLEC-4.0.0, Plugfy-0.6,). The full details about the numbers of God Class detected by each tool in each software system found in the data description part at the replication package [25].

Table 8 shows the number of detection tools against the number of God Classes detected by tools, manual validation, and the ratio of God Classes (manual validation) over the whole dataset. As can be seen, the higher the number of detection tools, the lower the number of detected God Classes. Therefore, the gap between both labels (God Class/Not God Class) is increased, and the imbalanced dataset problem will be more complex. According to [17], in their study, state that if the dataset is exceptionally imbalanced, the best balancing techniques fails although permits the model to be more accurate, because the model is difficult to be applicable due to the few number of label that belonging to the minority class (God Class in our case). It is important that the dataset includes a large number of design smells in order to train the learning classifiers on a good training set. Therefore, the classifier obtains a better behavior. The last row of Table 8 reports the Pearson correlations between the number of God Classes detected for each number of tools and the number of God classes after manual validation is 0.97 which is highly positive value show that whether the number of God Classes detected by tools increased the number of God Classes manually detected also increased.

4.2. Training the classifiers

In this stage, after the dataset has been prepared to be ready for inputting to the classifiers, as the first step, we conducted an exploratory experiment before the main two experiments to investigates whether the using the whole set of metrics (M1 to M16) have an influence to the classifiers results. For this purpose, we assessed the importance of each metric using the importance function in the rminer package, overcomes 0.5. Ten metrics were selected by the importance function: TLOC, NCLOC, CLOC, WMC, RFC, EXEC, DIT, NOA, NOTc, and DC. In this experiment, the classifiers have been trained on the whole set of metrics and on the important set.

Fig. 2 shows the Kappa, ROC, and MCC values of machine learning performance without data balancing using all the metrics and only the important set. As it can be seen, a better performance is achieved using the whole set of metrics (blue line) compared with important set (red line) in all figures, except the set of (BNB, CNB, GB) in the Kappa and MCC tests, and the GB in the ROC area test where obtained a slightly better performance. The differences between Kappa values of all metrics (blue line) and the important set (red line) were analyzed with a Wilcoxon rank-sum test. The null hypothesis in this case were "Classifiers have the same behavior on God Class detection using the whole and important set of metrics". The results show that the null hypothesis is rejected, with a p-value (0.00016). Therefore, the result is significant which means classifiers have different behavior. Also, the classifiers achieved better performance (better behavior) with the whole set of metrics in the ROC area and MCC tests. Therefore, the null hypothesis is rejected with a p-values (0.00001, 0.00008) for the ROC and MCC respectively. The same approach was followed in cases of accuracy, precision, and F-measure tests where the null hypothesis also is rejected with p-values less than (0.00001). Due to the paper size, we do not show the figures of accuracy, precision, and F-measure Since the performance is better using the whole set of metrics.

4.2.1. Experiment 1: Without dataset balancing

In this section, we will answered the first research question:

RQ1 *To what extent can machine learning techniques be effectively leveraged for God Class design smell detection?*

Table 7
Distribution of detected God Classes in the selected software systems using the set of tools (GC-tool) and manual validation (GC-ManualValidation) results.

Project	GC-tools	GC-manual	Project	GC-tools	GC-manual
AngryIPScanner-3.0	2	0	JFreeChart-1.0.X	161	37
Apeiron-2.92	16	1	JHotDraw-5.2	26	3
checkstyle-6.2.0	21	0	KeyStore Explorer-5.1	47	12
DigiExtractor-2.5.2	27	2	Lucene-3.0.0	146	42
Freemind-1.0.1	42	23	Matte-1.7	32	6
FullSync-0.10.2	13	0	Mpxj-4.7	137	57
GanttProject-2.0.10	167	15	OmegaT-3.1.8	194	27
JasperReports-4.7.1	311	81	Plugfy-0.6	1	0
jAudio-1.0.4	122	21	pmd-4.3.x	29	10
Java graphplan-1.0.7	18	3	sMeta-1.0.3	20	4
JCLEC-4.0.0	88	0	SQuirreL SQL Client-3.7.1	89	20
JDistlib-0.3.8	22	14	xena-6.1.0	227	107
Total #GC-tools	**1958**				
Total #GC-manual	**485**				

Table 8
The number of God Classes detected by tools, human, and their ratio against the whole dataset.

#Tools	#GC-tools	GC-manual	Manual/WholeDataset (%)
1	1419	229	1.8%
2	300	99	0.78%
3	160	89	0.71%
4	55	45	0.36%
5	24	23	0.17%
Alltools	1958	485	4%
Correlation	**0.97**		

Table 9
Accuracy, Kappa, F-measure, Precision, ROC. and MCC performance values achieved by each classifier without dataset balancing.

	Classifier	Accuracy	F-measure	Precision	Kappa	ROC	MCC
1	CBoost.	99.1579	0.9916	0.9918	0.8880	0.9508	0.8885
2	RF.	99.1261	0.9913	0.9913	0.8823	0.9426	0.8826
3	XGBRF.	99.1102	0.9911	0.9912	0.8798	0.9415	0.8802
4	LGBM.	99.0864	0.9908	0.9909	0.8755	0.9373	0.8761
5	XGB.	99.0863	0.9909	0.9909	0.8767	0.9404	0.8770
6	Bagging.	98.9672	0.9896	0.9896	0.8585	0.9274	0.8591
7	ET.	98.8719	0.9887	0.9887	0.8476	0.9209	0.8480
8	AB.	98.8480	0.9882	0.9882	0.8384	0.9065	0.8398
9	MLP.	98.8083	0.9881	0.9883	0.8396	0.9215	0.8409
10	GP.	98.6415	0.9862	0.9862	0.8124	0.8963	0.8133
11	KNN.	98.6018	0.9857	0.9856	0.8026	0.8820	0.8044
12	SVM.	98.5620	0.9856	0.9858	0.8071	0.9042	0.8077
13	SGD.	98.5382	0.9846	0.9848	0.7822	0.8578	0.7883
14	LR.	98.3555	0.9826	0.9826	0.7526	0.8395	0.7590
15	DT.	98.3396	0.9832	0.9832	0.7711	0.8798	0.7724
16	LSVM.	98.3078	0.9822	0.9821	0.7473	0.8392	0.7529
17	PA.	98.2522	0.9815	0.9816	0.7366	0.8307	0.7446
18	LDA.	98.2363	0.9809	0.9812	0.7243	0.8162	0.7353
19	Nu-SVM.	98.1929	0.9820	0.9822	0.7670	0.8872	0.7676
20	NC.	98.1410	0.9818	0.9824	0.7095	0.8974	0.7624
21	QDA.	97.5689	0.9770	0.9792	0.7095	0.8975	0.7149
22	Perceptron.	97.4021	0.9739	0.9768	0.6428	0.8497	0.6624
23	Ridge.	97.2750	0.9665	0.9711	0.4679	0.6621	0.5340
24	GNB.	96.9572	0.9732	0.9810	0.6901	0.9569	0.7159
25	MNB.	96.1548	0.9431	0.9366	0.1094	0.5029	0.0384
26	GB.	96.1469	0.9426	0.9244	0.0000	0.5000	0.0000
27	BNB.	87.8923	0.9117	0.9694	0.3411	0.9232	0.4472
28	CNB.	75.5304	0.8303	0.9614	0.1659	0.8252	0.2811

The classification in this experiment has been performed with the original dataset where the God Classes are randomly distributed on training, validation, and testing sets. Table 9 reports the Accuracy, Kappa, F-measure, Precision, ROC, and MCC values for each classifier with the best configuration that has been selected with 10-fold cross-validation. The results sorted by decreasing order to the accuracy values.

All classifiers have achieved high performance values with a few exceptions. The accuracy, precision, and F-measure values of all the classifiers were high on average 97%, 97.9%, and 97.3% respectively, compared with kappa 68.9%, ROC 85.9%, and MCC 70.3%. CBoost is one of the Boosting family classifiers that achieved the highest performance values in all metrics except ROC, while CNB from Naive Bayes achieved the lowest performance in accuracy. F-measure and GB from the Boosting the lowest performance in kappa, precision, ROC, and MCC. The high performance may be related to the features of these metrics (accuracy, precision, and F-measure) that consider the capability of the classifier model to classify the true negative and positive classes. Although the accuracy, F-measure, and precision results of most classifiers are high, it is not enough definitely to indicate the advantage of using machine learning classifiers [22,23]. For this reason, we included the kappa, ROC, and MCC tests as explained before.

The kappa values are shown in the fifth column of Table 9. More than 78% (22 out of 28) of the classifiers have achieved kappa greater than (66%). Especially, the set of CBoost, RF, XGBRF, XGB, LGBM, and Bagging has obtained the best results (>85%), which interpreted as very good behavior. The higher value was 88.8% achieved by CBoost while the lower value was zero by GB. Also, as it can be seen, 93% of the classifiers (26 out of 28) were achieved ROC values greater than 80%, and 50% of this set (13 out of 26) greater than 90% which means according to the ROC interpretation shown in Table 6 that classifiers have been obtained good to excellent behavior. The higher ROC value was 96% achieved by GNB while the lower value was 50% by GB. The last column of the table shows the MCC values for each classifier. Near to 42% of classifiers have achieved MCC greater than 80% (12 out of 28), which means that classifiers have very good agreement in God Class prediction. CBoost has obtained the higher value (88.9%) while the GB the lowest value (50%)

Based on above, we find the vast majority of machine learning classifiers have achieved high and accurate performance values according to the accuracy, Kappa, precision, F-measure, ROC, and MCC measurements, such as CBoost, RF, and XGBRF because it is represents the natural scenario when detection design smells. Therefore, we conclude that machine learning techniques effectively can be leveraged for God Class design smell detection. The null hypothesis: *(Machine learning techniques cannot be effectively leveraged for God Class design smell detection.)* is rejected.

4.2.2. Experiment 2: With dataset balancing

This section will answered the second research question:

RQ2 *To what extent can avoiding data balancing have an influence to the accuracy of God Class design smell detection?*

The second experiment was developed over balanced data, where SMOTE has been used to balance the training data as aforementioned in stage 2. Table 10 reports the performance values for each classifier with the best configuration that has been selected with 10-fold cross-validation. Also, the results sorted by decreasing order to the accuracy values.

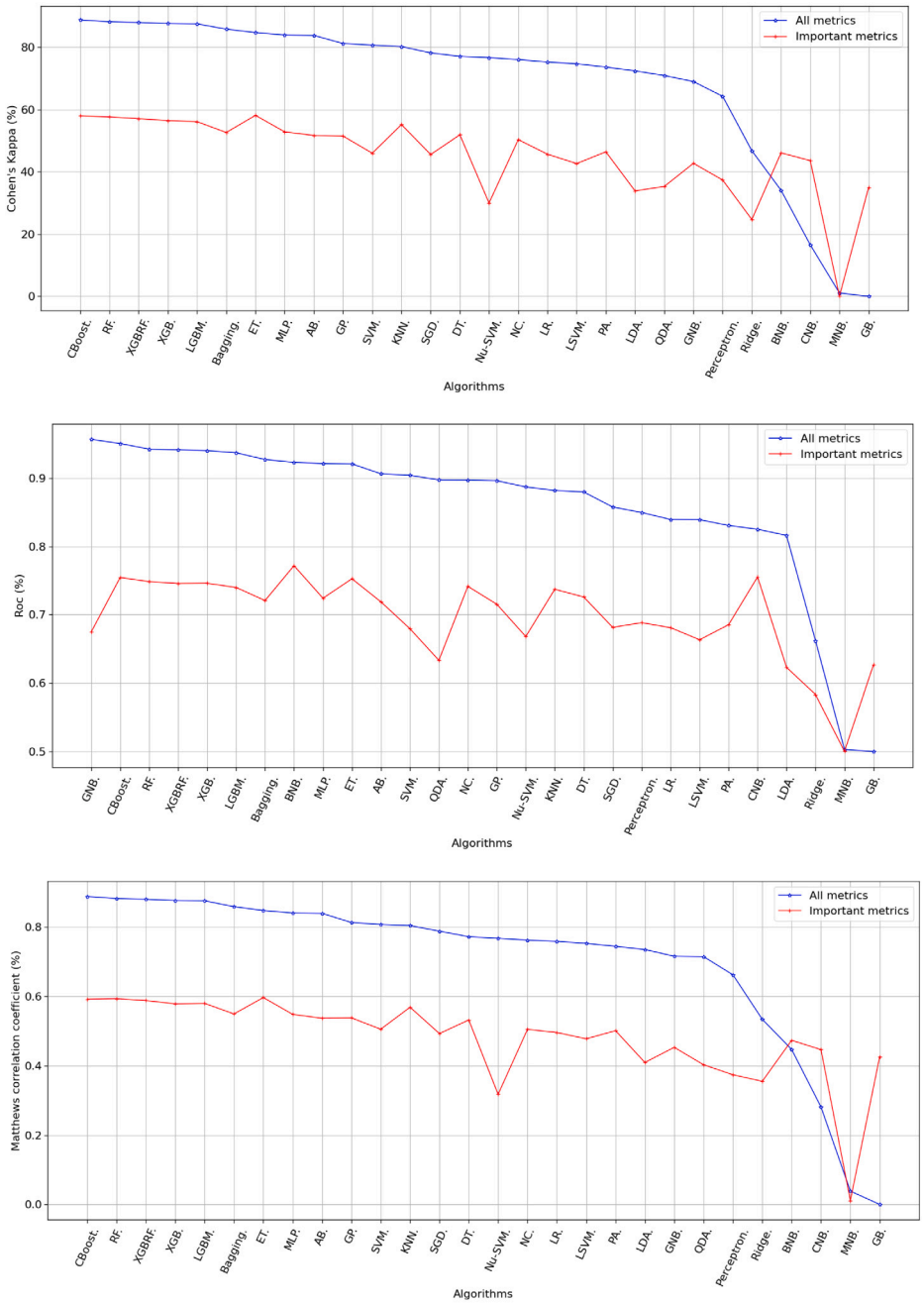


Fig. 2. Top: Kappa values with all the metrics and important metrics sorted by decreasing values of the former. Middle: ROC values with all the metrics and important metrics sorted by decreasing values of the former. Bottom: MCC values with all the metrics and important metrics sorted by decreasing values of the former.

All classifiers have achieved high performance values on average for 96.1% for accuracy, 71% for kappa, 96.8% for F-measure, 98.1% for precision, 92.6% for ROC, and 72.9% for MCC respectively. LGBM, XGBRF, XGB from the Boosting family are the classifiers that achieved the best performance, while the CNB and MNB of Naive Bayes family have achieved the lowest performance values. As it can be seen, comparing the average performance values of this experiment (with data balancing) with the first experiment (without data balancing), we observed its slightly better, in particular, the value of precision (97.9% to 98.1%), kappa (68.9% to 71%), ROC (85.9% to 92.6%), and MCC (70.3% to 72.9%), while the accuracy and F-measure have been slightly decreased from 97% to 96.1% and 97.3% to 96.8% respectively. The achieved results after data balancing also confirm the answer to the first research question **(RQ1)** that machine learning techniques can be leveraged effectively for God Class design smell detection. After using SMOTE and comparing Tables 9 and 10, all classifiers obtained different order based on their accuracy. For example, in experiment 1 (without data balancing), LGBM has achieved the fifth rank in terms of kappa, F-measure, precision, and MCC while in experiment 2 (with data balancing) it is full front to the first rank, i.e., the best classifier with data balancing in terms of accuracy, kappa, F-measure, and precision. Table 11 shows the confusion matrices, the Kappa and ROC (in percentage) of LGBM classifier over the 10-fold cross validation without (left part) and with (right part) data balancing. LGBM achieves the best Kappa and ROC using data balancing (right part). Compared to the left part, the changes rose slightly, where the percentage of true positives is raises from 2.89% to 3.32%, while the false negatives reduces from 0.24% to 0.17%. However, the false positive rose from 0.45% to 0.55%, while Kappa and ROC also rose slightly.

Table 10
Accuracy, Kappa, F-measure, Precision, ROC, and MCC performance values achieved by each classifier with dataset balancing.

	Classifier	Accuracy	F-measure	Precision	Kappa	ROC	MCC
1	LGBM.	99.1261	0.9915	0.9918	0.8883	0.9621	0.8894
2	XGB.	99.0270	0.9905	0.9909	0.8796	0.9617	0.8938
3	CBoost.	98.9911	0.9902	0.9907	0.8732	0.9596	0.8747
4	RF.	98.9911	0.9901	0.9906	0.8729	0.9588	0.8743
5	Bagging.	98.8481	0.9890	0.9901	0.8615	0.9725	0.8658
6	XGBRF.	98.8243	0.9888	0.9900	0.8591	0.9736	0.8637
7	ET.	98.6019	0.9865	0.9876	0.8291	0.9492	0.8323
8	AB.	98.5789	0.9847	0.9850	0.8152	0.9188	0.8195
9	MLP.	98.3635	0.9841	0.9850	0.7956	0.9233	0.7982
10	DT.	98.1888	0.9825	0.9835	0.7755	0.9153	0.7780
11	Nu-SVM.	98.0297	0.9810	0.9820	0.7538	0.9040	0.7560
12	SVM.	97.9349	0.9797	0.9802	0.7416	0.8849	0.7422
13	KNN.	97.9187	0.9803	0.9824	0.7558	0.9253	0.7616
14	GP.	97.8554	0.9793	0.9804	0.7410	0.8987	0.7432
15	GB.	97.7959	0.9788	0.9799	0.7347	0.8961	0.7371
16	Ridge.	97.6564	0.9774	0.9788	0.7118	0.8827	0.7153
17	LDA.	97.5453	0.9764	0.9779	0.7001	0.8793	0.7038
18	NC.	97.3249	0.9762	0.9786	0.7047	0.9103	0.7128
19	LR.	96.8700	0.9723	0.9797	0.6828	0.9466	0.7064
20	SGD.	96.7350	0.9713	0.9796	0.6755	0.9497	0.7019
21	LSVM.	96.6320	0.9706	0.9797	0.6776	0.9577	0.7065
22	PA.	96.6000	0.9704	0.9794	0.6681	0.9518	0.6968
23	Perceptron.	96.4967	0.9694	0.9784	0.6581	0.9391	0.6853
24	QDA.	96.3060	0.9680	0.9779	0.6444	0.9436	0.6749
25	BNB.	96.0757	0.9664	0.9779	0.6342	0.9503	0.6697
26	GNB.	95.4322	0.9618	0.9769	0.6004	0.9537	0.6456
27	CNB.	76.1262	0.8344	0.9616	0.1706	0.8292	0.2862
28	MNB.	76.1183	0.8343	0.9619	0.1718	0.8323	0.2886

Table 11
Confusion matrix, Kappa and ROC (in %) of the best classifier (LGBM) without data balancing (left part), and with data balancing (right part).

	LGBM/without balancing		LGBM/with balancing	
	Not-GodClass	GodClass	Not-GodClass	GodClass
Not-GodClass	**96.29**	0.45	**95.78**	0.55
GodClass	0.24	**2.89**	0.17	**3.32**
	Kappa (%)	ROC (%)	Kappa (%)	ROC (%)
	87.6	93.7	88.8	96.2

Fig. 3 shows the Kappa, ROC, and MCC values of classifiers with and without data balancing. According to the ROC area figure, most of classifiers achieved slightly better performance when the training set is balanced (red line) compared with the imbalanced set (blue line) with some exception in case of GNB and SVM. On the other hand, according to the values of kappa (Top figure) and MCC (Bottom figure), most of classifiers achieved better behavior without data balancing, despite the slightly improvement in the remained classifiers. In general, we can say that classifiers have the same behavior on God Class detection with and without data balancing.

A wilcoxon rank-sum test comparing kappa, ROC, MCC, accuracy, precision, and F-measure values with and without balancing separately is used to test the null hypothesis *"Avoiding data balancing will not influence the accuracy of God Class design smell detection"*. The test gives a p-values greater than 0.05 in terms of kappa (0.1585), precision (0.2340), and MCC (0.2460) which means that the null hypothesis is accepted because the difference between classifications with and without balancing is not significant. On the other hand, The p-values were less than 0.05 in terms of accuracy, ROC, and F-measure. The test gives a p-values of 0.00054, 0.0010, and 0.00001 for accuracy, F-measure and ROC respectively, which means the difference between classifications with and without balancing is statistically significant. Therefore, the null hypothesis is rejected.

Although the null hypothesis is rejected, in the case of accuracy, ROC, and F-measure, the data balancing does not influence the performance of the classifiers. Therefore, we concludes that data balancing slightly influence the performance of the classifiers.

Table 12
Characteristics of projects in the first dataset used in the replication experiments.

Project&Version	NOP	NOC	NOM	TLOC	God class
ant-rel-1.8.3	80	1473	13,213	119,256	6
argouml-VERSION_0_14	94	1373	9045	199,075	2
cassandra-cassandra-1.1.0	48	699	11,360	110,712	2
apache-wicket-1.4.11	260	1568	12,429	174,033	4
derby-10.3.3.0	50	1746	5987	535,187	24
hadoop-release-0.2.0	20	327	2460	34,662	2
hsqldb-2.2.0	52	590	5004	254,014	11
incubator-livy-0.6.0-incubating	11	1016	450	130,696	6
nutch-release-0.7	53	532	3220	50,578	0
qpid-0.18	226	2172	21,448	189,271	6
xerces-Xerces-J_1_4_2	42	489	6088	150,445	6
eclipse-R3_4	12	5061	924	423,423	25
elasticsearch-v0.19.0	570	1395	21,739	315,619	2
Total	**1518**	**18,441**	**113,367**	**2,686,971**	**96**

5. Experiments replication

To mitigate a threat of the study where results can be biased by a dataset build by the authors themselves, in this section, the replication of both experiments (1 and 2) is presented using the same set of machine learning classifiers and two different datasets proposed by other authors to confirm machine learning classifiers could be leveraged during design smell detection and analyzing the influence of data balancing on design smell detection.

5.1. Data collection

The new datasets are constructed on the basis of datasets used in the literature. The first dataset is previously used in [16,66] while the second dataset points in the work by [21]. Both datasets consists of manually validated instances of design smells, where the God Class is one of those smells. These datasets are available at the replication package [25]. The format of the new datasets is changed by including the same set of metrics shown in Table 3 of Section 3.2.1, in order to be in the same format of the dataset constructed by our team and described in Section 3.2.

The first of the two dataset consists of 125 releases of 13 projects, and more than 120,000 classes, while the second of these two dataset consists of 74 software projects and more than 55,000 classes. In this experiment, from the first dataset, we selected a release of each software project that include the higher number of detected God Classes and manually labeled as God Class instances. Also, regarding the second dataset, we selected the same sample of God Classes that were manually validated (420 classes) and used in the original experiment. Tables 12 and 13 reports a description of the first and second datasets respectively, gathered in the replication of the experiments, in addition, the number of detected God Classes in each project. Comparing the main dataset with the new datasets, we can see the ratio of detected God Classes in the main dataset is much higher (main dataset 485 God Class against 96 in the first dataset and 140 in the second dataset). The reasons are due to the difference between datasets regarding the number and nature (domain and size category) of the analyzed software systems in each dataset, the total number of classes found in the systems, the number of detection tools used for God Class detection, the number of experts who executed the manual validation process, and the proposed God class labeling strategy.

For example, the first dataset consisted of 13 software systems formed of 18,441 classes that did an analysis a using one tool developed by the authors to detect God Classes. The manual validation process was conducted by a group of two authors who individually validated the candidate God Classes. Regarding the second dataset [21], the number of the analyzed systems is 74, formed of more than 55,000 classes. The tools PMD and iPlasma were used to detect God Classes,

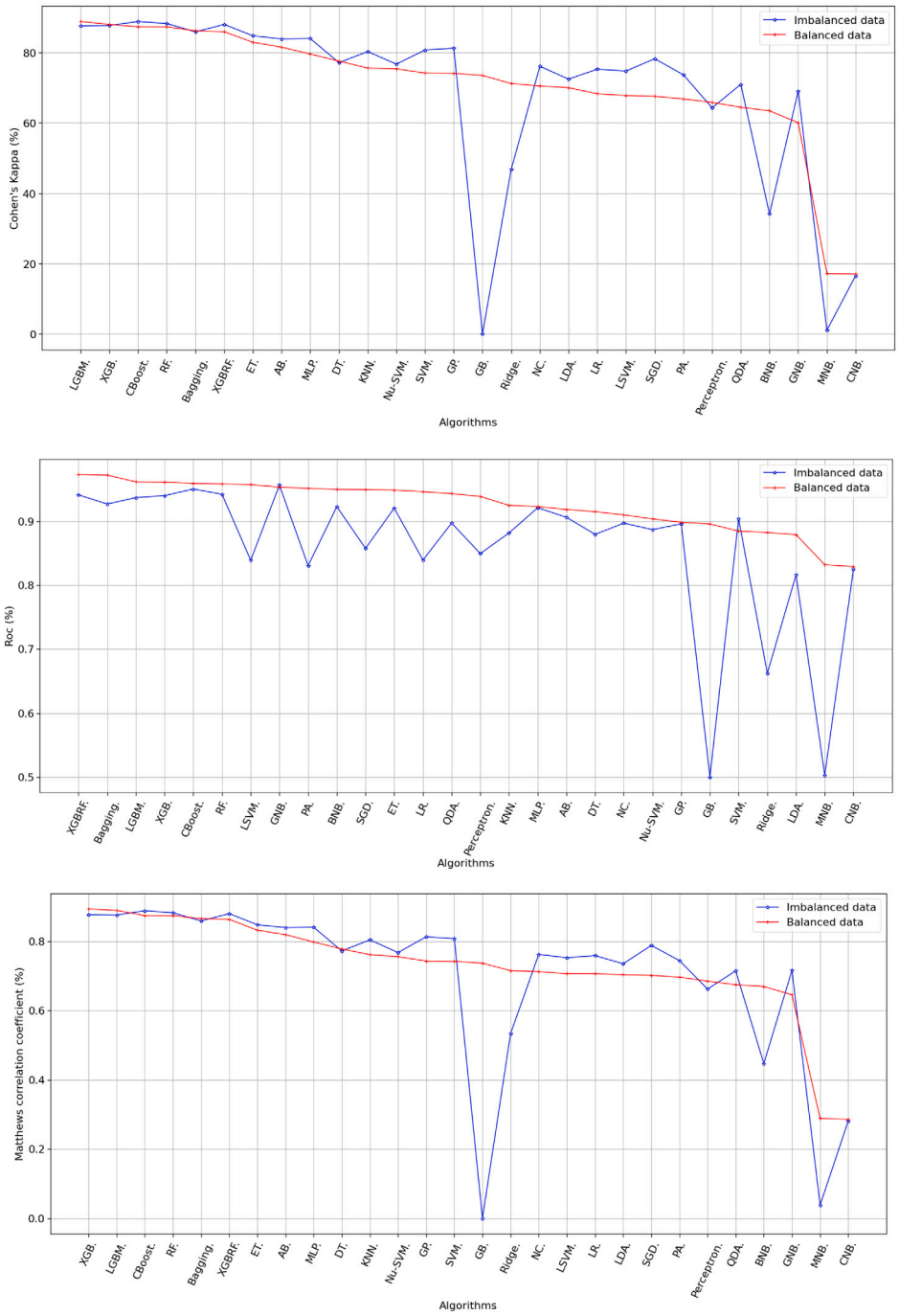


Fig. 3. Top: Kappa values with and without data balancing sorted by decreasing values of the former. Middle: ROC values with and without data balancing sorted by decreasing values of the former. Bottom: MCC values with and without data balancing sorted by decreasing values of the former.

and the manual validation performed by three Masters students on the selected sample of detected God Classes (420). Finally, compared with the main dataset we build and use on the original set of experiments, the number of analyzed software was 24 formed of 12,587 classes. A set of five detection tools include Decor, iPlasma, PMD, JDeodorant, and Borland Together were used for the first stage of labeling by automatic detection of God Classes. In addition, the manual validation was conducted on the whole set of detected God Classes (1958) by three persons who have a Ph.D. in different IT fields.

5.2. Results and discussion of the replication

The number of God Classes labeled by human experts in the first dataset are 96 of a total of 18,441 classes, which represents 0.52% of true positives while in the second dataset 140 of a total of the selected sample of 420 classes, which represents 33% . The percentage of first dataset (0.52%) is lower than the percentage of our dataset (4%), and the percentage of the second dataset (33%) is higher than the percentage of our dataset. In experiment 2 (with data balancing), LGBM achieved the first performance rank using our dataset according to accuracy, F-measure, precision indicators while the second and third rank according to MCA and ROC, respectively. While the seventh and fifteen rank using the first and second datasets according to the kappa

Table 13
Sample of classes of each project and the number of detected God classes (GC) in each sample (N) in the second dataset used in the replication experiments.

Project&Version	N	GC	Project&Version	N	GC
aoi-2.8.1	9	7	jmoney-0.4.4	6	3
argouml-0.34	2	0	jparse-0.96	4	1
axion-1.0.M2	3	1	jpf-1.0.2	8	2
castor-1.3.1	6	1	jruby-1.5.2	3	1
colt-1.2.0	3	0	jspwiki-2.8.4	6	2
columba-1.0	6	1	jsXe-04_beta	6	2
displaytag-1.2	5	1	jung-2.0.1	2	0
drawswf-1.2.9	6	0	junit-4.1	1	0
drjava-20100913-r5387	9	4	log4j-1.2.16	5	1
emma-2.0.5312	4	1	lucene-3.5.0	4	1
exoportal-1.0.2	13	0	marauroa-3.8.1	5	1
findbugs-1.3.9	7	0	megamek-0.35.18	5	2
fitjava-1.0.1	27	2	mvnforum-1.2.2-ga	10	2
fitlibraryforfitnesse-20100806	9	0	nekohtml-1.9.14	6	3
freecol-0.10.3	8	7	openjms-0.7.7-beta-1	2	0
freecs-1.3.20100406	4	3	oscache-2.4.1	5	1
freemind-0.9.0	2	0	picocontainer-2.10.2	2	0
galleon-2.3.0	14	6	pmd-4.2.5	4	0
ganttproject-2.0.9	5	1	poi-3.6	3	1
heritrix-1.14.4	1	1	cobertura-1.9.4.1	0	0
hsqldb-2.0.0	15	13	proguard-4.5.1	2	2
itext-5.0.3	12	6	quartz-1.8.3	6	4
jag-6.1	5	4	quickserver-1.4.7	6	2
jasml-0.10	6	1	quilt-0.6-a-5	6	1
jasperreports-3.7.3	2	0	roller-4.0.1	6	0
javacc-5.0	7	3	squirrel_sql-3.1.2	6	0
jedit4.3.2	4	3	sunflow-0.07.2	3	0
jena-2.6.3	1	0	tomcat-7.0.2	3	0
jext-5.0	5	1	trove-2.1.0	3	0
jFin_DateMath-1.0.1	4	0	velocity-1.6.4	1	0
jfreechart-1.0.13	2	1	wct-1.5.2	9	2
jgraph-5.13.0	5	3	webmail-0.7.10	4	2
jgraphpad-5.10.0.2	7	4	Weka-3.7.5	4	2
jgrapht-0.8.1	2	2	xalan-2.7.1	9	5
jgroups-2.10.0	5	2	xerces-2.10.0	20	12
jhotdraw-7.5.1	6	2	xmojo-5.0.0	4	1
jmeter-2.5.1	2	2	pooka-3.0-080505	6	3
Total number of God Classes	**140**				

values respectively. This classifier has not been able to keep the high performances achieved using our balanced dataset. Same behaviors were replicated with the whole remaining classifiers. The results show that the ratio of God Classes in the dataset influenced the performance of classifiers. Furthermore, the differences between the characteristics of software systems used in these datasets regarding the size and domain have an important role in the performance of the classifiers. It is worth noting the importance of this outcome as it significantly addresses important aspects needed by the research community towards obtaining accurate detection results using the machine learning approach.

All classifiers achieved high performance values in the replication experiments using the first and second datasets with some exceptions. The average values of accuracy, kappa, F-measure, precision, ROC, and MCC using the first dataset without data balancing were (98.9%, 29.6%, 99%, 99.4%, 67.3%, and 34.5%) respectively, while with data balancing were (98.4%, 38.8%, 98.7%, 99.5%, 84%, and 44%) respectively. Regarding the second dataset, the average values of accuracy, kappa, F-measure, precision, ROC, and MCC without data balancing were (92.04%, 81.8%, 92%, 93%, 91%, and 82.4%) respectively, while with data balancing were (92.3%, 82.5%, 92.3%, 92.7%, 91.4%, and 83%) respectively. As can be seen, in both datasets, they have a slight improvement in their behavior with data balancing with some exceptions.

The achieved results confirm the answer of the first research question **(RQ1)** in the main experiment of this study, that machine learning approaches can be effectively exploited for God Class design smell detection. Therefore, also in this experiment, the null hypothesis: *Machine learning techniques cannot be effectively leveraged for God Class design smell detection.* is rejected.

Table 14
P-values.

Dataset	Accuracy	F-measure	Precision	Kappa	ROC	MCC
FirstDataset	0.00022	0.01352	**0.1031**	0.00142	0.00001	0.00262
SecondDataset	**0.08364**	**0.05486**	0.04338	**0.05118**	0.01208	**0.05486**

Figs. 4 and 5 shows the Kappa, ROC, and MCC values for all classifiers using the new dataset with and without balancing. In both figures, according to the kappa and ROC values, most classifiers were achieved a slightly better performance behavior with data balancing (red line) compared with imbalanced (blue line). This behavior agrees with the results of the main experiment using our dataset shown in Fig. 3.

We conducted a Wilcoxon test on the accuracy, kappa, F-measure, precision, ROC, and MCC values, in order to test the second null hypothesis *"Avoiding data balancing will not influence the accuracy of God Class design smell detection"*. The p-values of this test are shown in Table 14. According to the p-values obtained using the first dataset, the results permit us to reject the null hypothesis (with exception of precision *p*-value), which means that the result is statistically significant. Also, the p-values according to the second dataset allow us to accept the null hypothesis (with the exception of precision and ROC p-values), which means that the result is not statistically significant. In general, the result from the replication experiment concludes that data balancing slightly influences the performance of classifiers on detecting God Class and agrees with the conclusion of the main experiment.

6. Threats to validity

The threats to the validity of the set of experiments designed will be discussed in this section.

6.1. Construct validity

The main threat to construct validity is the set of selected design smell detection tools used to detect God Class. Despite the fact that these tools are implemented based on different strategies according to the definition of God Class, they also allowed different set of metrics and threshold values to be used. This portrayed high accuracy on detection, and represented the set of the most cited state of the art according to our previous work [1], however, a study on threats to construction of this study is given.

To overcome this threat a manual validation process of the whole 1958 God Classes obtained by detection tools was conducted. This task was accomplished by a group of three researchers who have Doctorate Degrees in Information Technology (software engineering and computer science), and have excellent experience with object-oriented programming and a very good knowledge on design smells. The set of machine learning classifiers is considered another threat to the study — regarding the selected classifiers families, set of metrics that have been used to train the classifiers, and selecting the best model. To mitigate this threat, we selected a large set of learning classifiers well-known for this purpose by analyzing previous studies. The selected set belongs to eleven families in order to avoid the behavior of specific classifiers that can influence the detection accuracy and behavior. Additionally, we applied a feature selection experiment before the classification in order to identify the metrics with the most influence in the results and then we used the cross validation process to reduce the variance in the detection results.

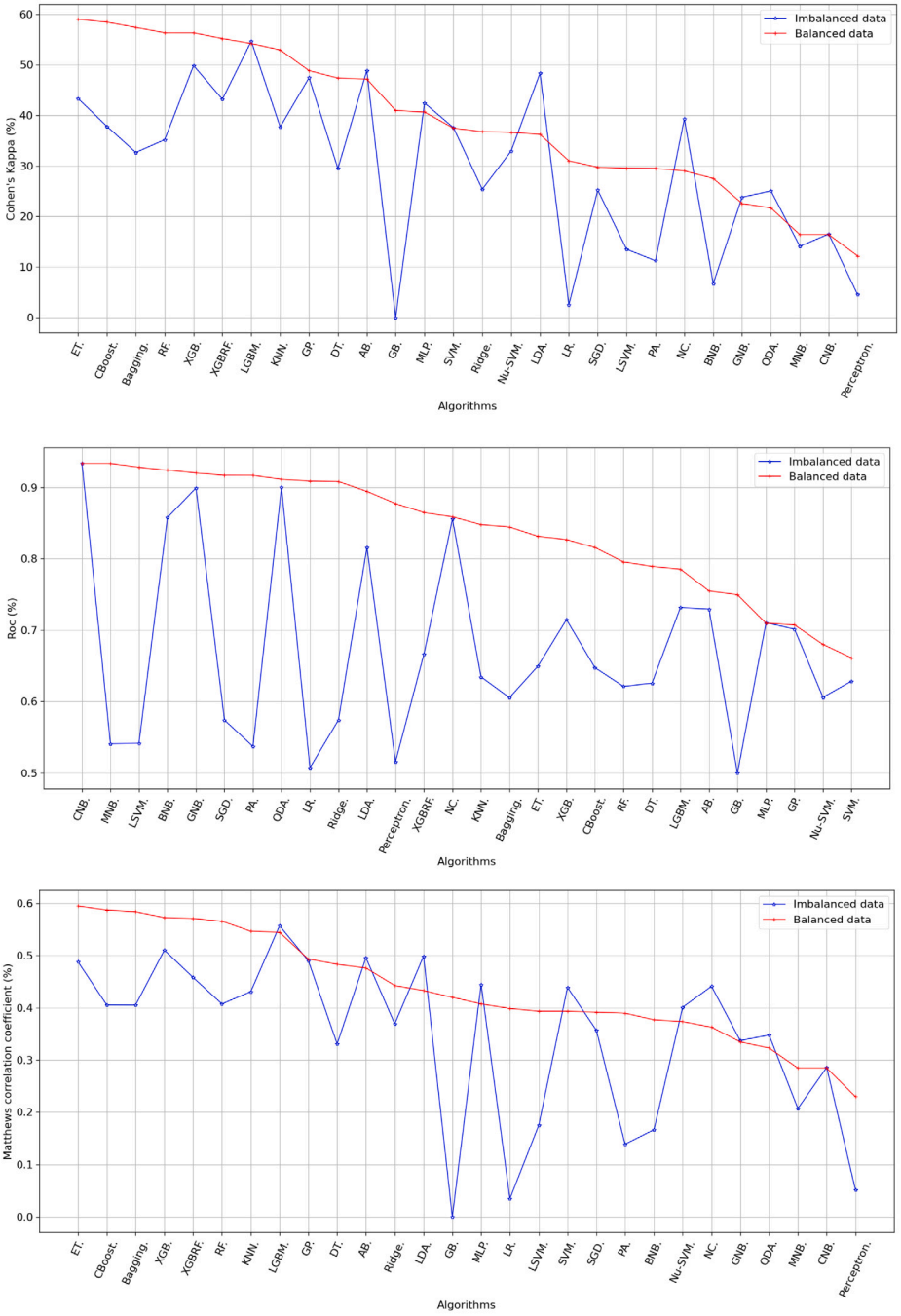


Fig. 4. Top: Kappa values with and without data balancing sorted by decreasing values of the former according to first dataset. Middle: ROC values with and without data balancing sorted by decreasing values of the former according to first dataset. Bottom: MCC values with and without data balancing sorted by decreasing values of the former according to first dataset.

6.2. Internal validity

Several terms are used to describe design smell in different context, for example, God Class is known as Large Class code smell, and The Blob anti-pattern. These concepts are related to different characteristics of God Class, such as, a class that has many responsibilities, high complexity, more functionality, and a very large number of lines of codes. Therefore, various sets of metrics were used to measure these characteristics and to construct the detection rules.

To overcome this threat we used different tools that were developed based on these concepts that had a focus of using different detection strategies. Another essential threat is the total number of detected God Classes in the dataset, especially, after the manual validation. But it is normal that we detected a few numbers of God Classes compared to the whole classes. This threat will lead to an unbalancing problem between class instances (God Class/Not God Class) in the dataset. To manage this threat, we used oversampling and under-sampling techniques in order to balance the dataset.

6.3. External validity

The nature of software systems used in the dataset is the main external threat. Only the set of open source systems that belong to different domain and size categories have been used in the experiment and may affect the generalization of results regarding the training process of machine learning classifiers. Most of the proposed detection

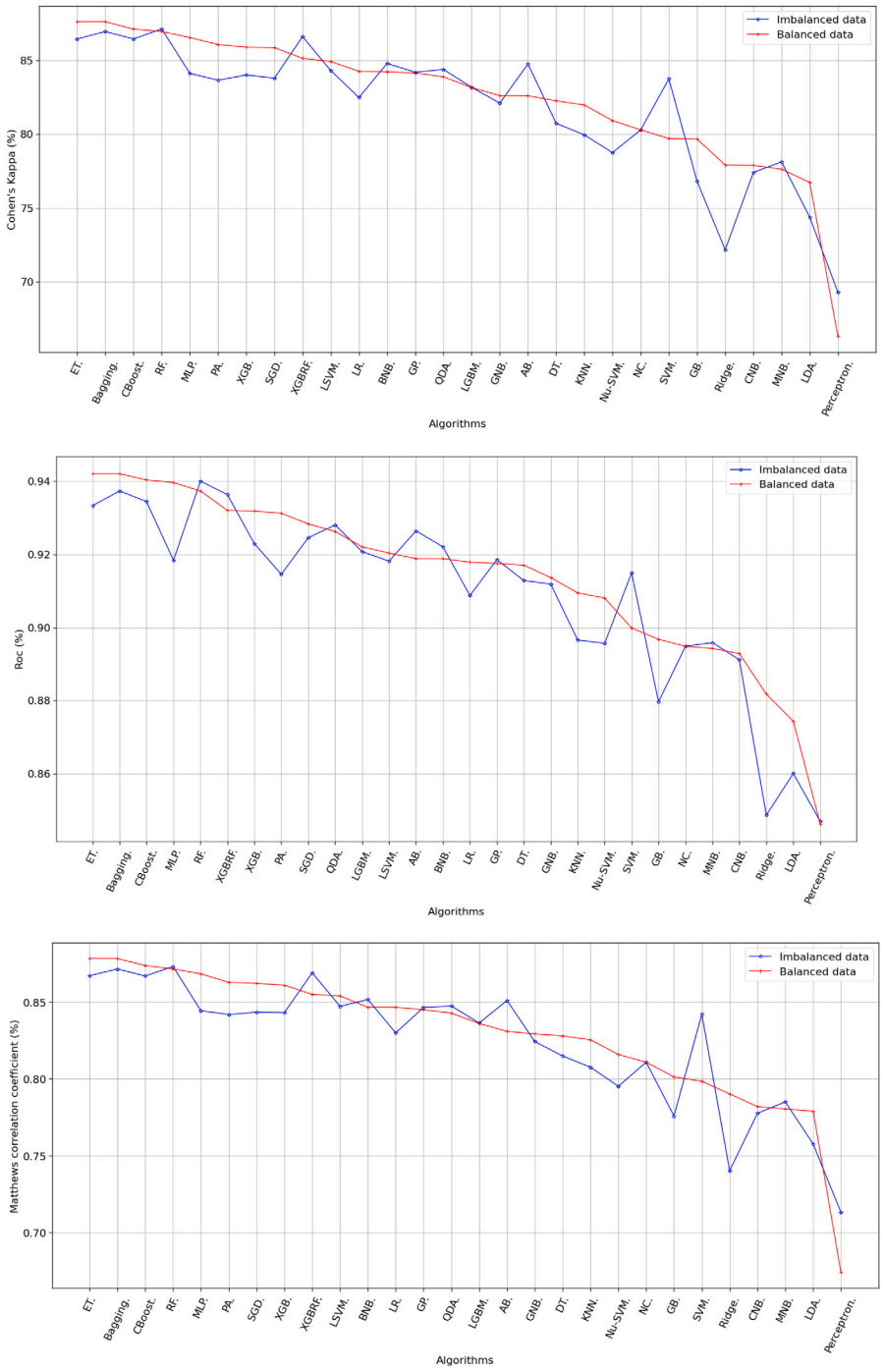


Fig. 5. Top: Kappa values with and without data balancing sorted by decreasing values of the former according to second dataset. Middle: ROC values with and without data balancing sorted by decreasing values of the former according to second dataset. Bottom: MCC values with and without data balancing sorted by decreasing values of the former according to second dataset.

approaches, techniques, and tools are evaluated on benchmarks or corpus of open source systems. These types of systems are stored in well-known repositories. We think there are no differences between the development of open source and commercial systems. Both types have been developed in similar languages and high quality. Therefore, whether the systems are open source or commercial will not influence the behavior of machine learning classifiers. We managed this threat by replicating the experiment on two other datasets of open source systems from different domains and size categories and involve a high number of classes.

Based on the outcomes of this study that was conducted only one design smell (God Class), with projects developed in a single programming language, and because only one balancing technique is used to compare with the imbalanced case. Consequently, we note a generic threat to the generalization of the study. we note that there may/exist other methods of data balancing besides SMOTE that we employed in this paper, also with the existence of different design smells, and projects developed in different programming language. The application of those other methods may improve the results. However, in our experiments, SMOTE did not improve God class detection. As per the proposition in this paper, we note that the study are not fully

generalizable but the conclusions are promising for the application in real design smells detection scenarios as mentioned above and the focus on other measures, such as Kappa, ROC, and MCC to assess the classifier behavior, once the accuracy is very good.

All these threats either internal, external, and construction affect the generalization of the study results but they are wide similar to the rest of the state of the art experiments

7. Conclusion and future work

The paper presents a study to ascertain whether machine learning classifiers can be exploited effectively for purposes of design smell detection with a specific focus to God class. Additionally, this study compares the performance of the classifiers in the imbalanced dataset case with the balanced dataset case. For this purpose, we designed a set of experiments that compared 28 supervised machine learning classifiers belonging to various families-to aid in detecting God Class design smell. To conduct this study, we constructed a large dataset of 24 open-source software from different domains and sizes that consist of 12,587 classes where 485 God Class were manually validated. The dataset was prepared using an external source code analyzer tool (Refactorit), a set of five well-known design smell detection tools that are common in God Class detection as automatic early advisors, together with a group of three human experts that have been used to label the class instances detected by one or more tool. The strategy we followed for labeling detection tools result was as follows: If one tool or more detects the class as God Class, then the label will be (1), otherwise (0). A set of 1958 God Classes were detected by the applied tools where the probabilities of false negative are reduced, and evaluated by a group of human experts where the final number of God Classes was 485 where the manual validation reduced the probabilities of false positives.

As a first step, we conducted a feature selection experiment to analyze the set of metrics and select the most important one. This previous experiment shows that it should be better working with the whole set of metrics. Next, we developed two different experiments for classification as follows: Experiment 1 using the original dataset without data balancing, and Experiment 2 using the Synthetic Minority Oversampling Technique (SMOTE) to solve the data unbalanced problem, which afterwards increases the classifier performance. For all the classifiers, the achieved performance results were evaluated in terms of accuracy, Cohen Kappa, ROC area, precision, and F-measure. According to the performance metrics values in the features selection experiment, all classifiers have achieved a better performance using the whole set of metrics compared with important set. For this purpose, we conducted the main experiments using the whole set. In experiment 1 (without data balancing) using the main dataset, XGB has obtained the highest performance in terms of accuracy, kappa, F-measure, precision, and ROC values. In contrast, in Experiment 2 (with data balancing), LGBM has obtained the highest performance using all performance metrics values except ROC, while XGB full back to the second rank. The results shows that most of classifiers have not able to keep the achieved high performances (i.e., their rank). According to the p-values obtained from the performance values for the combined values of accuracy, kappa, F-measure, precision, and ROC with and without balancing, the null hypothesis "the classifiers have the same behavior on God Class detection, with and without data balancing" is rejected in all cases. The results of this experiment shows that data balancing slightly affect on the performances of classifiers.

After replicating both experiments (1 and 2) using new datasets, all classifiers have achieved high performance values as in the main dataset. Based on the obtained performance results in the replication experiments, we confirm our conclusions using main dataset that machine learning techniques can be effectively leveraged for design smell detection particularly using God Class.

Despite the large number of previous works that confirmed the accuracy of machine learning techniques in design smell detection, this area has not been exploited sufficiently to improve the design smell detection. This situation was clearly observed in the replication of the experiments while using different datasets. For this purpose, other information related to software systems, such as size categories and domains should be taken into account by the industries and the research community when they developing new approaches, techniques, and tools as shows our work in [36].

Although we can see the results in Figs. 4 and 5 that are obtained using the new datasets (first and second), which also agrees with Fig. 3 that most of classifiers achieved slightly better performance, the null hypothesis "the classifiers have the same behavior on God Class detection with and without data balancing" cannot be rejected as shown in Table. As a consequence, we have an empirical evidence that data balancing does not have adequately influence on the performances of classifiers to detect God Class design smell. Also, from the perspective of this study, it is imperative to highlight that despite the conclusions drawn from our experiments that were performed only on one design smell on projects developed in a single programming language, and only one balancing technique is used to compare with the imbalanced case. We note a generic threat to the generalization of the study. there exist other methods of data balancing besides SMOTE that we employed in this paper, also with the existence of different design smells, and projects implemented in different programming languages. The application of those other methods may improve the results. However, in our experiments, SMOTE did not improve God class detection. As per the proposition in this paper, we note that the study are not fully generalizable but the conclusions are promising for the application in real design smells detection scenarios as mentioned above and the focus on other measures, such as Kappa, ROC, and MCC to assess the classifier behavior, once the accuracy is very good. The outcomes of this work from one side confirms that machine learning can be used effectively in the context of detection of true positive design smells (God Class in our case). This is also the case in the smells that have a negative impact on software quality, and from the other perspective, data balancing slightly influences the accuracy of design smell detection when machine learning classifiers are used. Also, in terms of practical applicability, the authors note that leveraging machine learning for smell detection not only can play a significant role in maintaining software quality but, based on the obtained results-this can as well can also be employed to automatically detect and predict the likelihood of design smells with a higher degree of certainty.

In the future, this work will be extended further in certain directions in order to improve and generalize the obtained results. To this end, we plan to replicate the conducted experiments by using a large-scale dataset that is manually evaluated, a wide set of software metrics, more types of design smells, apply more machine learning techniques, analyze software systems implemented in different programming languages, and includes other inputs that can help in obtaining better classifiers behavior.

CRediT authorship contribution statement

Khalid Alkharabsheh: Conceptualization, Methodology, Software, Validation, Formal analysis, Visualization, Data curation, Investigation, Supervision, Writing – original draft, Writing – review & editing. **Sadi Alawadi:** Conceptualization, Methodology, Formal analysis, Visualization, Software, Data curation, Validation, Writing – review & editing. **Victor R. Kebande:** Conceptualization, Methodology, Writing – review & editing. **Yania Crespo:** Methodology, Software, Investigation. **Manuel Fernández-Delgado:** Methodology, Software, Investigation. **José A. Taboada:** Methodology, Software, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] K. Alkharabsheh, Y. Crespo, E. Manso, J. Taboada, Software design smell detection: a systematic mapping study, Softw. Qual. J. (2018).

[2] F. Pérez, Refactoring Planning for Design Smell Correction in Object-Oriented Software (Ph.D. thesis), School of Engineering, Valladolid University, 2011.

[3] W.H. Brown, R.C. Malveau, H.W. McCormick, T.J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, John Wiley & Sons, Inc, 1998.

[4] M. Choinzon, Y. Ueda, Detecting defects in object oriented designs using design metrics, in: J. Conf. on Knowledge-Based Software Engineering, 2006, pp. 61–72.

[5] R. Fourati, N. Bouassida, H. Abdallah, A metric-based approach for anti-pattern detection in UML designs, Comput. Inf. Sci. (2011) 17–33.

[6] C. Marinescu, R. Marinescu, P.F. Mihancea, R. Wettel, IPlasma: An integrated platform for quality assessment of object-oriented design, in: Intl. Conf. Software Maintenance - Industrial and Tool Volume, 2005, pp. 77–80.

[7] N. Moha, Y.-G. Guéhéneuc, DECOR: a tool for the detection of design defects, in: Intl. Conf. on Automated Software Engineering, 2007, pp. 527–528.

[8] M.J. Munro, Product metrics for automatic identification of “bad smell” design problems in java source-code, in: Intl. Conf. Software Metrics, 2005, p. 15.

[9] R. Shatnawi, Deriving metrics thresholds using log transformation, J. Softw.: Evol. Process. 27 (2) (2015) 95–113.

[10] L. Tahvildar, K. Kontogiannis, Improving design quality using meta-pattern transformations: a metric-based approach, J. Softw.: Evol. Process. 16 (4–5) (2004) 331–361.

[11] S. Hassaine, F. Khomh, Y.-G. Guéhéneuc, S. Hamel, Ids: an immune-inspired approach for the detection of software design smells, in: Intl. Conf. Quality of Information and Communications Technology, 2010, pp. 343–348.

[12] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, H. Sahraoui, BDTEX: A GQM-based Bayesian approach for the detection of antipatterns, J. Syst. Softw. 84 (4) (2011) 559–572.

[13] J. Kreimer, Adaptive detection of design flaws, Electron. Notes Theor. Comput. Sci. 141 (4) (2005) 117–136.

[14] N. Maneerat, P. Muenchaisri, Bad-smell prediction from software design model using machine learning techniques, in: Intl. J. Conf. on Computer Science and Software Engineering, 2011, pp. 331–336.

[15] K. Alkharabsheh, Y. Crespo, M. Fernández-Delgado, J.M. Cotos, J.A. Taboada, Assessing the influence of size category of the project in god class detection, an experimental approach based on machine learning (MLA), in: International Conference on Software Engineering & Knowledge Engineering, 2019, pp. 361–366.

[16] F. Pecorelli, D. Di Nucci, C. De Roover, A. De Lucia, A large empirical assessment of the role of data balancing in machine-learning-based code smell detection, J. Syst. Softw. (2020) 110693.

[17] F. Pecorelli, F. Palomba, D. Di Nucci, A. De Lucia, Comparing heuristic and machine learning approaches for metric-based code smell detection, in: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 2019, pp. 93–104.

[18] N. Tsantalis, T. Chaikalis, A. Chatzigeorgiou, Jdeodorant: Identification and removal of type-checking bad smells, in: Intl. Conf. on Software Maintenance and Reengineering, 2008, pp. 329–331.

[19] Borland, Together, 2008, http://www.borland.com/together.

[20] T. Copeland, PMD Applied, Centennial Books, 2005.

[21] F.A. Fontana, M.V. Mäntylä, M. Zanoni, A. Marino, Comparing and experimenting machine learning techniques for code smell detection, Empir. Softw. Eng. 21 (3) (2016) 1143–1191.

[22] M.I. Azeem, F. Palomba, L. Shi, Q. Whang, Machine learning techniques for code smell detection: A systematic literature review and meta-analysis, Inf. Softw. Technol. 108 (2019) 115–138.

[23] A. Al-Shaaby, H. Aljamaan, M. Alshayeb, Bad smell detection using machine learning techniques: A systematic literature review, Arab. J. Sci. Eng. 45 (2020).

[24] J. Alzubi, A. Nayyar, A. Kumar, Machine learning from theory to algorithms: An overview, J. Phys. Conf. Ser. 1142 (2018) 012012.

[25] k. Alkharabsheh, S. Alawadi, V. Kebande, Y. Crespo, M. Delgado, J. Taboada, Replication package of raw data, scripts and all necessary material for replication, 2021, URL: https://drive.google.com/drive/folders/1_Q7i52QPb-MogNzW6vpePWSNkYyA1gKX?usp=sharing.

[26] A. Maiga, N. Ali, N. Bhattacharya, A. Sabané, Y.-G. Guéhéneuc, G. Antoniol, E. Aïmeur, Support vector machines for anti-pattern detection, in: Intl. Conf. Automated Software Engineering, 2012, pp. 278–281.

[27] A. Maiga, N. Ali, N. Bhattacharya, A. Sabane, Y.-G. Gueheneuc, E. Aimeur, Smurf: A svm-based incremental anti-pattern detection approach, in: Intl. Conf. on Reverse Engineering, 2012, pp. 466–475.

[28] M. Peiris, J.H. Hill, Towards detecting software performance anti-patterns using classification techniques, ACM SIGSOFT Softw. Eng. Notes 39 (1) (2014) 1–4.

[29] D. Di Nucci, F. Palomba, D.A. Tamburri, A. Serebrenik, A. De Lucia, Detecting code smells using machine learning techniques: are we there yet? in: Intl. Conf. on Software Analysis, Evolution and Reengineering, 2018, pp. 612–621.

[30] N.V. Chawla, Data mining for imbalanced datasets: An overview, in: O. Maimon, L. Rokach (Eds.), Data Mining and Knowledge Discovery Handbook, Springer US, Boston, MA, 2010, pp. 875–886, http://dx.doi.org/10.1007/978-0-387-09823-4_45.

[31] S. Hassaine, F. Khomh, Y. Gueheneuc, S. Hamel, Ids: An immune-inspired approach for the detection of software design smells, in: 2010 Seventh International Conference on the Quality of Information and Communications Technology, 2010, pp. 343–348, http://dx.doi.org/10.1109/QUATIC.2010.61.

[32] K. Kourou, T.P. Exarchos, K.P. Exarchos, M.V. Karamouzis, D.I. Fotiadis, Machine learning applications in cancer prognosis and prediction, Comput. Struct. Biotechnol. J. 13 (2015) 8–17.

[33] D.D. Nucci, F. Palomba, D.A. Tamburri, A. Serebrenik, A.D. 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 Computer Society, 2018, pp. 612–621, http://dx.doi.org/10.1109/SANER.2018.8330266.

[34] J. Ali Reshi, S. Singh, Investigating the role of code smells in preventive maintenance, J. Inf. Technol. Manag. 10 (4) (2018) 41–63.

[35] Y.-G. Guéhéneuc, H. Sahraoui, F. Zaidi, Fingerprinting design patterns, in: 11th Working Conference on Reverse Engineering, IEEE, 2004, pp. 172–181.

[36] K. Alkharabsheh, Y. Crespo, M. Fernandez-Delgado, J. Viqueira, J. Taboada, Exploratory study of the impact of project domain and size category on the detection of the god class design smell, Softw. Qual. J. (2021).

[37] R. Per, H. Martin, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2) (2009) 131–164.

[38] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, Experimentation in Software Engineering, Springer, 2012.

[39] K. Alkharabsheh, S. Almobydeen, Y. Crespo, J.A. Taboada, Influence of nominal project knowledge in the detection of design smells: An exploratory study with god class, Int. J. Adv. Stud. Comput. Sci. Eng. 5 (11) (2016) 120–127.

[40] K. Alkharabsheh, Y. Crespo, E. Manso, J. Taboada, Comparación de herramientas de Detección de Design Smells, in: Jornadas de Ingeniería Del Software Y Bases de Datos, 2016, pp. 159–172.

[41] K. Alkharabsheh, Y. Crespo, E. Manso, J. Taboada, Sobre el grado de acuerdo entre evaluadores en la detección de design smells, in: Jornadas de Ingeniería Del Software Y Bases de Datos, 2016, pp. 143–157.

[42] S. Counsell, E. Mendes, Size and frequency of class change from a refactoring perspective, in: Int. Conf. on Software Evolvability, 2007, pp. 23–28.

[43] F.A. Fontana, P. Braione, M. Zanoni, Automatic detection of bad smells in code: An experimental assessment, J. Obj. Technol. 11 (2) (2012) 5–1.

[44] W. Li, R. Shatnawi, An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution, J. Syst. Softw. 80 (7) (2007) 1120–1128.

[45] J.A. Santos, M.G. de Mendonça, C.V. Silva, An exploratory study to investigate the impact of conceptualization in god class detection, in: Intl. Conf. on Evaluation and Assessment in Software Engineering, 2013, pp. 48–59.

[46] A. Yamashita, M. Zanoni, F.A. Fontana, B. Walter, Inter-smell relations in industrial and open source systems: A replication and comparative analysis, in: Intl. Conf. on Software Maintenance and Evolution, 2015, pp. 121–130.

[47] M. Lanza, 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.

[48] M. Fowler, K. Beck, Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional, 1999.

[49] N. Moha, Y.-G. Gueheneuc, L. Duchien, A.-F. Le Meur, Decor: A method for the specification and detection of code and design smells, IEEE Trans. Softw. Eng. 36 (1) (2010) 20–36.

[50] A. Yamashita, L. Moonen, Exploring the impact of inter-smell relations on software maintainability: An empirical study, in: Intl.Conf. on Software Engineering, 2013, pp. 682–691.

[51] A. Tiberghien, N. Moha, T. Mens, K. Mens, Répertoire des Défauts de Conception, Technical Report 1303, University of Montreal, 2007.

[52] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. 15 (1) (2014) 3133–3181.

[53] J. Wainer, Comparison of 14 different families of classification algorithms on 115 binary datasets, 2016, arXiv preprint arXiv:1606.00930.

[54] S. Alawadi, M.F. Delgado, D.M. Pérez, Machine Learning Algorithms for Pattern Visualization in Classification Tasks and for Automatic Indoor Temperature Prediction (Ph.D. thesis, Ph. D. thesis), Universidade de Santiago de Compostela, 2018.

[55] I.H. Witten, E. Frank, M.A. Hall, C.J. Pal, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, 2016.

[56] C. López, E. Manso, Y. Crespo, The identification of anomalous code measures with conditioned interval metrics, in: 13th TOOLS Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2010) MáLaga, Spain, MáLaga, 2010.

[57] G. Rasool, Z. Arshad, A review of code smell mining techniques, J. Softw.: Evol. Process. 27 (11) (2015) 867–895.

[58] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.

[59] N. Blackman, J. Koval, Interval estimation for cohen's kappa as a measure of agreement, Stat. Med. 19 (5) (2000) 723–741.
[60] A.P. Bradley, The use of the area under the roc curve in the evaluation of machine learning algorithms, Pattern Recognit. 30 (7) (1997) 1145–1159.
[61] B. Matthews, Comparison of the predicted and observed secondary structure of t4 phage lysozyme, Biochim. Biophys. Acta (BBA) - Protein Struct. 405 (2) (1975) 442–451.
[62] M. Hollander, D.A. Wolfe, E. Chicken, Nonparametric Statistical Methods, Vvol. 751, John Wiley & Sons, 2013.
[63] F.A. Fontana, P. Braione, M. Zanoni, Automatic detection of bad smells in code: An experimental assessment, J. Obj. Technol. 11 (2) (2012) 5:1–38, http://dx.doi.org/10.5381/jot.2012.11.2.a5.
[64] F.A. Fontana, M. Zanoni, A. Marino, M.V. Mantyla, Code smell detection: Towards a machine learning-based approach, in: Int. Conf. on Software Maintenance, 2013, pp. 396–399.
[65] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, Developing fault-prediction models: What the research can show industry, IEEE Softw. 28 (6) (2011) 96–99.
[66] F. Pecorelli, F. Palomba, D. Di Nucci, A. De Lucia, Comparing heuristic and machine learning approaches for metric-based code smell detection, in: Proceedings of the 27th International Conference on Program Comprehension, in: ICPC '19, IEEE Press, 2019, pp. 93–104, http://dx.doi.org/10.1109/ICPC.2019.00023.