Learning to Detect Community Smells in Open Source Software Projects

Nuri Almarimi[a], Ali Ouni[a], Mohamed Wiem Mkaouer[b]

[a]*ETS Montreal, University of Quebec, QC, Canada.*
[b]*Rochester Institute of Technology, NY, USA.*

Abstract

Community smells are symptoms of organizational and social issues within the software development community that often lead to additional project costs. Recent studies identified a variety of community smells and defined them as sub-optimal patterns connected to organizational-social structures in the software development community. To early detect and discover existence of potential community smells in a software project, we introduce, in this paper, a novel machine learning-based detection approach, named csDetector, that learns from various existing bad community development practices to provide automated support in detecting such community smells. In particular, our approach learns from a set of organizational-social symptoms that characterize the existence of potential instances of community smells in a software project. We built a detection model using Decision Tree by adopting the *C4.5 classifier* to identify eight commonly occurring community smells in software projects. To evaluate the performance of our approach, we conduct an empirical study on a benchmark of 74 open source projects from Github. Our statistical results show a high performance of csDetector, achieving an average accuracy of 96% and AUC of 0.94. Moreover, our results indicate that the csDetector outperforms two recent state-of-the-art techniques in terms of detection accuracy. Finally, we investigate the most influential community-related metrics to identify each community smell type. We found that the number of commits and developers per time zone, the number of developers per community, and the social network betweenness and closeness centrality are the most influential community

characteristics.

Keywords: Community smells detection, social debt, socio-technical metrics, machine learning

1. Introduction

Software engineering can be primarily described as an organized effort of "social" activity of organizations, individuals, and stakeholders to build a software product. Therefore, the organizational social structure in a software development community, including the interactions among developers, is an essential prerequisite for a successful software product.

An intrinsic characteristic of software projects resides in their (rapid) evolution involving a large number of clientele and stakeholders [1]. Although such growth is beneficial for the project's success, there reveals another set of challenges that go beyond maintaining the software quality and functionality, and that impact the socio-managerial structure of the project. Hence, several researchers and practitioners revealed how critical is the evolution of organizational aspects to prevent it from decay, and consequently software projects failures [2, 3]. Recent studies coined the set of socio-technical patterns, negatively impact the organizational health of the project, as *community smells* [4]. Community smells are connected to circumstances based on poor organizational and social practices that lead to the occurrence of *social debt* [5, 6]. Social debt is connected to poor organization structures that often lead to short and/or long term social issues within a project. These problems could manifest in several forms, *e.g.*, lack of communications or coordination among members in a software community. Indeed, M. Scott Peck pointed out that :

> *"There can be no vulnerability without risk; there can be no community without vulnerability; there can be no peace, and ultimately no life, without community."*

For example, one of the common community smells, is the "organizational silo effect" [7, 8, 9] which manifests as a recurring social network sub-structure

featuring highly decoupled developers community structures. From an analytical perspective, this community smell can be seen as a set of patterns over a social network graph and could be detectable using different graph connectivity measurements.

To early detect potential instances of poor community management during a software development life-cycle, efficient and automated techniques are needed [6, 10, 11]. Although there have been a few studies to define, characterize, and detect community smells in open source projects, they are applied, in general, to a limited set of projects, and their generalizability requires a manual effort and human expertise to define and calibrate a set of detection rules to match the symptoms of a community smell with the actual characteristics of a given software project [3, 12, 7, 13].

We build on top of previous studies to develop a model that learns from previously detected community smells, by considering the patterns of their characteristics into features, which allows their classification. In this paper, we introduce a novel machine learning based approach for community smell detection, named csDetector. The proposed approach learns from a set of organizational-social symptoms that characterize the existence of a potential community smell. csDetector adopts a detection model to provide a learn-by-example process to automatically detect community smells from various software projects. The proposed approach allows practitioners to choose project examples that are mostly similar to their context, as it learns from their associated community smells, to provide a personalized detection model for their project.

To evaluate our approach, we experiment several existing machine learning algorithms, including C4.5, JRip, Random Forest, Nave Bayes, SMO, and LibSVM, on eight commonly occurring community smell types [6, 11, 14] extracted from 74 open-source software systems. Results show that the experimented techniques can provide high performance for all the considered community smells. However, the highest performance was achieved by the C4.5 classifier. In particular, the obtained results indicate that C4.5 is able to detect the different types of community smells with an average accuracy of 96%, an F-measure of

92% and an Area Under the receiver operating characteristic Curve (AUC) of 94%.

The main contributions of the paper can be summarized as follows:

- We introduce a novel approach, named csDetector, for community smells detection using decision tree to learn from a set of organizational-social symptoms that characterize the existence of potential instances of community smells in active real word open source software projects.
- We report the results of an empirical study with an implementation of our approach on a benchmark of 74 open source projects from Github and eight common community smells. Results show a high performance of the proposed approach, achieving an average accuracy of 96% and AUC of 0.94. Moreover, we found that the C4.5 decision tree algorithm outperforms 5 widely used machine learning algorithms including JRip, random forest, Naïve Bayes, SMO, and LibSVP. Moreover, the statistical analysis of the obtained results show that our approach outperforms two recent state-of-the-art techniques in terms of detection accuracy.
- We conduct a set of experiments to investigate the most influential community-related metrics to identify each community smell type. We found that the number of commits and developers per time zone, the number of developers per community, and the social network betweenness and closeness centrality are the most influential community characteristics. We also conduct a sensitivity analysis to assess if there exists influential data points than may influence the stability of our model.
- We provide our comprehensive dataset collected and used in this study publicly available for replication purposes, and to foster research in the field of community smells and social debt [15].

Replication package. We provide our comprehensive dataset collected and used in this study publicly available for replication purposes [15].

Paper organization. The remainder of the paper is organized as follows. Section 2 provides the necessary background and the related work on community smells. In Section 3, we describe our approach for community smells detection. In Section 4, we present our empirical evaluation setup, and then present and discuss the obtained results in Section 4. We discuss, in Section 6, the threats to validity. Finally, in Section 7, we conclude and outline our future work.

2. Background and Related Work

In this section, we present the necessary background related to community smells. Then, we discuss the related work that translated the definition of community smells into actionable detection rules.

2.1. Community Smells Definitions

Community smells are defined as a set of social-organizational circumstances that occur within the software development community, and that have a negative effect on the relations health within the development community and may cause social debt over time [6]. In our work, we investigate the detection of common community smells that are identified and defined in the literature. We refer to following list of existing community smell types [6, 3]:

- **Organizational Silo Effect (OSE):** The Organizational Silo Effect smell is manifested when a too high decoupling between developers, isolated sub-groups, and lack of communication and collaboration between community developers occur. The consequence of this smell is an extra unforeseen cost to a project by wasted resources (*e.g.*, time), as well as duplication of code [6, 10].

- **Black-cloud Effect (BCE):** The black-cloud effect smell occurs when developers have a lack of information due to limited knowledge sharing opportunities (*e.g.*, collaborations, discussions, daily stand-ups, etc.), as well as a lack of expert members in the project that are able to cover the

experience or knowledge gap of a community. The BCE smell may cause mistrust between members and creates selfish behavioral attitudes [6].

- **Prima-donnas Effect (PDE):** The prima-donnas effect smell occurs when a team of people is unwilling to respect external changes from other team members due to inefficiently structured collaboration within a community. The presence of this smell may create isolation problems, superiority, constant disagreement, uncooperativeness and raise selfish team behavior, also called "prima-donnas" [6, 10].

- **Sharing Villainy (SV):** This smell is caused by a lack of high-quality information exchange activities (*e.g.*, face-to-face meetings). The main side effect of this smell limitation is that community members share essential knowledge such as outdated, wrong and unconfirmed information [6].

- **Organizational Skirmish (OS):** This community smell is caused by a misalignment between different expertise levels and communication channels among development units or individuals involved in the project. The existence of this smell leads often to dropped productivity and affects the project's timeline and cost [6].

- **Solution Defiance (SD):** The solution defiance smell occurs when the development community presents different levels of cultural and experience background, and these variances lead to the division of the community into similar subgroups with completely conflicting opinions concerning technical or socio-technical decisions to be taken. The existence of the SD smell often leads to unexpected project delays and uncooperative behaviors among the developers [6].

- **Radio Silence (RS):** The ratio silence smell occurs when a high formality of regular procedures takes place due to the inefficient structural organization of a community. The RS community smell typically causes changes to be retarded, as well as a valuable time to be lost due to complex and rigid formal procedures. The main effect of this smell is an unexpected

massive delay in the decision-making process due to the required formal actions needed [6].

- **Truck Factor Smell (TFS):** The truck factor smell occurs when most of the project information and knowledge are concentrated in one or few developers. The presence of this smell eventually leads to a significant knowledge loss due to the turnover of developers [14, 3].

In this paper, we focus primarily on the above-mentioned smells that are frequently occurring in the software industry [5, 11, 6, 7, 14]. We selected these smells as they are defined and investigated in recent studies and known to have negative effects in practice, which may lead to social debt [6] and poor source code quality [7].

2.2. Related Work

In recent years, the software engineering community studied the importance of organizational and social patterns in open source software systems [3, 12, 7]. Tamburri et al. investigated potential community smells in software projects, and found that such smells lead to a social debt in sub-optimal organizational-social structures in software systems [6, 14, 7].

A number of approaches have been devised to support and analyze community smells in software projects. Tamburri et al. defined a set of community smells based on various socio-technical characteristics and patterns which may lead to the emergence of social debt [10]. Later, Tamburri et al. proposed a tool called YOSHI to automate to detect organizational structure patterns across open source communities [12]. The proposed tool maps open-source projects onto community patterns, and introduces measurable attributes to identify organizational and social structure types and characteristics through formal detection rules. CodeFace is another tool developed as a Siemens product [16] and designed to identify developers communities based on building developer networks. Another tool called *Codeface4Smell* has been proposed later as an

extension of CodeFace to identify community smells based on community metrics and statistical values to measure the quality and health characteristics of software development communities [13]. *Codeface4Smell* uses the development history and mailing lists to assess the social network among developers. Avelino et al. introduced a tool called *Truck Factor* [14, 3] to measure information concentration within community members and support the software development community to deal with turnover of developers. The proposed approach estimates the truck factor values of Github projects based on historical information about developers contributions and collaborations from the commit log. Recently, Palomba et al. found that community smells are connected to code smells within open source software projects [7].

While the existing approaches attempt mainly to characterize and analyze organizational-social structures in software development communities, they do not cover multi-perspective characteristics of software projects. They mostly devise a limited and generic list of characteristics and symptoms to generate smells detection rules that characterize community smells. However, such rules will need a substantial human effort and expertise to calibrate these rules for each smell type and adapt them to different projects, organizations and contexts. Hence, we believe that appropriate detection is needed to fill this gap. Our approach aims at learning from existing smells to detect new ones to help developers better allocate their resources and save time and efforts through an automated detection of smells.

Besides studies on community smells, many research efforts have been carried out to study social debt in software engineering. In recent years, several studies on open-source and industrial projects have shown that collaboration and social structure are critical in software development and have an impact on the final software product quality [17, 18, 19, 20]. Many other research studies investigated the effect of organizational decisions on different collaborations and software product quality [21, 22, 23]. Tamburri et al. studied social debt by a comparison with technical debt based on common real-life scenarios that exhibit sub-optimal development communities [24]. Bird et al. used different

social network analysis techniques to assess the coordination between groups of developers with socio-technical dependencies [25, 26]. Cataldo et al. studied socio-technical congruence, *i.e.*, the degree to which technical and social dependencies match, using formal terms and conducted an empirical study to assess its impact on software product quality [27, 28, 18].

3. csDetector: Community Smells Detection Using Machine Learning

In this section, we present our approach, csDetector, for detecting community smells. csDetector aims at detecting various social-technical issues that can take place in open source projects. Figure 1 presents an overview of our proposed approach which consists of three main components (1) training data collection and organizational-social metrics extraction, (2) training model building using C4.5 classifier, and (3) community smells detection. In the following, we explain each of these components.

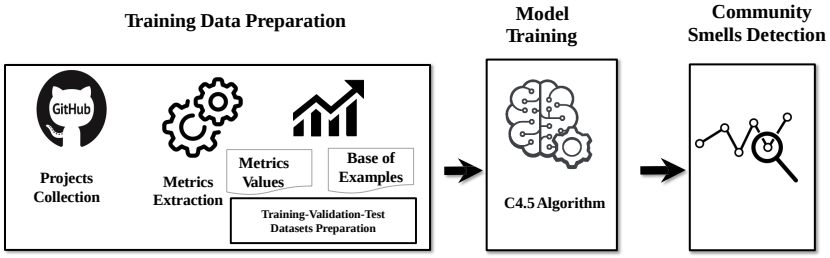

Figure 1: An overview of the proposed csDetector approach.

3.1. Data Collection

To build a base of real-world community smell examples that occur in software projects, we select a set of software projects which are diverse in nature (*e.g.*, size, application domain, popularity, etc.) that are most likely to experience community smells. Additionally, we considered open-source projects to access to their development history to calculate their social-organizational characteristics. We selected a set of projects from Github that exhibit various characteristics by considering the following criteria :

Table 1: Metrics Framework.

Dimension	**Acronym**	**Definition**	**Ref.**
Developer Social Network metrics	NoD	*Number of developers (NoD)*: is the total number of Developers who have changed the code in a project. The more developers modify the same files, the higher the need for collaboration between developers.	[29]
	TAP	*Number of active days of an author on a Project (TAP)*: is ratio of the total number of active days for each developer with respect to a project's lifetime by the total number of developers in a project.	New
	LCP	*Number of changed lines of a code per author in a project (LCP)*: is the total number of times that the code has been changed by a developer with respect to the total number of lines of code and the total number of developers in a project.	[29]
	CD	*Number of core developers (CD)*: is the total number of core developers in a project. A developer is considered as a core community member if he has a high degree of centrality of more than > 0.5 within the developer's social network.	[13]
	RCD	*Ratio of core developers (RCD)*: is the ratio of the number of core developers with respect to the total number of developers in a project.	[13]
	SD	*Number of sponsored developers (SD)*: is the total number of sponsored developers in a project. We consider a developer that hold a sponsored status if at least 95% of her/his commits are executed during weekdays and the working day time between 8am and 5pm.	[13]
	RSD	*Ratio of sponsored developers (RSD)*: is the ratio of the number of sponsored developers by the total number of developers in a project.	[13]
Social Network Analysis metrics	DC	*Graph Degree centrality (DC)*: is the number of connections that a developer has. The more connections with others a developer has, the more important the developer is.	[30]
	BC	*Graph Betweenness centrality (BC)*: is a measure of the information flow from one developer to another and devised as a general measure of social network centrality. It represents the degree to which developers stand between each other. A developer with higher BC would have more control over the community as more information will pass through her/him.	[30]
	CC	*Graph Closeness centrality (CC)*: is a measure of the distance between a developer to other developers in the network. This metric is strongly influenced by the degree of connectivity of a network.	[30]
	ND	*Network Density (ND)*: is a measure of a social network as a dense or sparse graph.	[13]
Community metrics	NC	*Number of communities (NC)*: is the total number of communities in a project and a measure of the strength of the structure of a project community.	[13]
	RCC	*Ratio of commits per community (RCC)*: is the ratio of the number of commits assigned to each community with respect to the total number of commits in a project. It provides a view of the distribution of the commit per community.	New
	RDC	*Ratio of developers per community (RDC)*: is the ratio of the number of developers assigned to each community in a project with respect to the total number of developers in a project. .	New
Geographic Dispersion metrics	TZ	*Number of time zones (TZ)*: is the total number of different time zones of developers in a project.	[31]
	RCZ	*Ratio of commits per time zone (RCZ)*: is the ratio of the number of commits in each time zone by the total number of time zones in a project. It provides a view of the distribution of the commit per time zone.	New
	RDZ	*Ratio of developers per time zone (RDZ)*: is the ratio of the number of developers per time zone in a project by the total number of time zones in a project. It provides a view of the distribution of the developers per time zone.	New
Formality metrics	NR	*Number of Releases in a project (NR)*: is the total number of releases present in a project.	New
	RCR	*Ratio of Commits per Release (RCR)*: is the ratio of the number of commits assigned to each release with respect to the total number of releases in a project.	New
	FN	*Formal network (FN)*: is calculating by the ratio of milestones assigned to the project and lifetime of the project.	[30]
Truck Number metrics	BFN	*Bus Factor Number (BFN)*: is the ratio of core developers present in a project with respect to the total number of developers in a project.	[32]
	TFN	*Truck Factor Number (TFN)* : is the total number of key developers in a project who can be unexpectedly lost, i.e hit by a truck before the project is discontinued.	[3]
	TFC	*Truck Factor Coverage (TFC)*: is the percentage of core developers and their associated authored files in a project.	[3]

- *Commit size*: the projects vary from medium size <10 KLOC, large size $(10-60)$ to very large size >60 KLOC.

- *Community size*: the projects team size vary from medium <100 members, large (100-900) members, to very large >900 members.

- *Programming language*: the selected projects are implemented in different programming languages including Java, C#, Python and C.

- *Existence of community smells:* the project contains at least one community smell.

From the obtained projects we randomly selected among them. We, then, manually inspected all of these projects to identify the potential existence of community smells using a crawler to collect and analyze their change history based with the assistance of existing guidelines from the literature [10, 6, 13, 24, 14, 7] to summarize and visualize different aspects, *e.g.*, social networks, metrics variations, developers commits contributions and collaborations, etc. Each author independently identified potential community smells. In case of a conflict about a smell, *i.e.*, there is a disagreement about the presence of a smell, the authors attempt to resolve it through an open discussion to reach an agreement about the salient symptoms of the candidate smell. All projects for which there was no agreement about a candidate community smell, were excluded from our ground truth dataset. We finally ended up with 74 projects from an initial list of 107 projects that have diverse types of community smells [15]. The details of the collected dataset are reported in Table 4. Each system repository can have one or many projects. Each project is one data point, so the total number of data points is 74 for each smell type. The dependent variable is a specific smell type. Moreover, Table 2 and Figure 2 provide more statistical details about our analyzed dataset in terms of the distribution of smells, developers, commits and days lived.

Table 2: Dataset Statistics.

Data	Statistic
Number of systems	23
Number of projects	74
Number of projects having at least one smell	74
Total number of smells	236
Average number of smells per project	3.18
Number of projects <50 developers	30
Number of projects 50 − 150 developers	21
Number of projects >150 developers	23
Average number of commits per project	1,103
Average number of days in each project	3,233

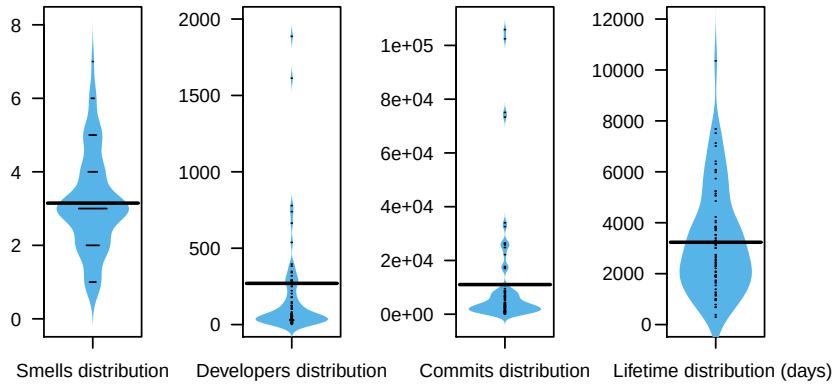

Figure 2: Beanplots for the distribution of smells, developers, commits, and lifetime among the studied projects.

3.2. Metrics framework

To capture community smells symptoms, we rely on (1) a set of metrics defined in previous studies [3, 13, 29, 30, 31], and (2) a new set of metrics that we introduce to capture more community-related proprieties that can be mined from the projects history. These metrics analyze different aspects in software development communities including organizational dimensional, social network characteristics, community developer's collaborations, and truck numbers. Table 1 depicts our list of metrics as well as state-of-the-art ones used in this research study. Our proposed metrics extend existing metrics to provide more

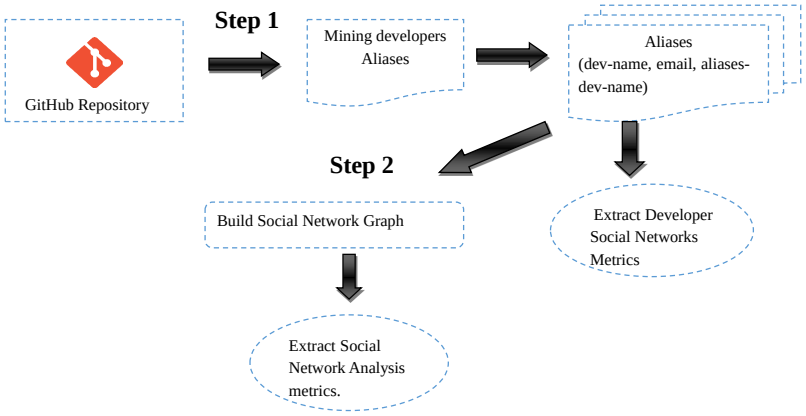

Figure 3: An overview of our social-organizational metrics framework.

project characteristics generalizations including developer social network, community structures, geographic dispersion, and developer network formality. For instance, the geographic dispersion metrics, *e.g.*, the *average number of commits per time zone* and the *average number of developers per time zone* would provide a whole view of the distribution of commits and developers per time zone.

Our metrics framework calculates the set of metrics from a target open source software project by analyzing its repository through commit information history available in its version control system (GitHub).

Figure 3 presents an overview our metrics framework which consists of two core steps to extract metrics.

Step 1. Mine developers aliases: The author alias mining and consolidation consists of the following substeps [3]: (*i*) Retrieval all unique dev-emails, where for each git commit has a dev-email associated with it, (*ii*) Retrieval of GitHub logins related to developers, (*iii*) Similarity matching of emails and logins: by applying Levenshtein distance [2], all aliases are compared and, with a certain degree of threshold value, consolidated, and (*iv*) Replacement of author emails by their respective aliases. The final transformation goes through all the commits once again and replacing original authors by their primary alias. As

a result, if there is a developer associated with commits with different names, we consider them as a single developer, and the output will be presented in a new aliases list. For example, "Bob.Rob" and "Bob Rob" are different names for a single developer, correspondent to commits, we consider them under the same developer, in a new aliases list, as a single identical substitution. We used a threshold value of 0.8 for the Levenshtein distance to identify different aliases used by the same developer, following the of Joblin et al. [16]. As future work, we will investigate different similarity techniques and threshold values to assess the accuracy of our approach.

Step 2. Build a social network graph: Social network analysis (SNA) has been used for studying and analyzing the collaboration and organization of developers who are working in teams within software development projects [21]. Our developers network model is based on a socio-technical connections during a software project development. Different social network analysis metrics have been devised to describe a community structure and predict quality factors in a software development project. Our approach builds a developer network from the version control system and tracks the change logs. Our adopted developers network is presented as a graph of nodes and edges, where the nodes represent developers and edges are the connections between two developers that are working on the same file and where they make a version control commit within one month of each side. Such social network allows then to calculate the different metrics including the degree centrality, betweenness centrality, closeness centrality, network density, etc. (cf. Table 1).

3.3. Training model

To prepare our detection model, it is important to perform a correlation analysis within the used metrics. We use our collected dataset to carry out a multi-collinearity analysis between the different metrics using Spearman ρ [33]. Figure 4 shows a visualization of the initial auto-correlations among all the variables in the dataset. We observe a few number of correlated metrics (dark-colored circles in blue or red for positive and negative correlations, respectively).

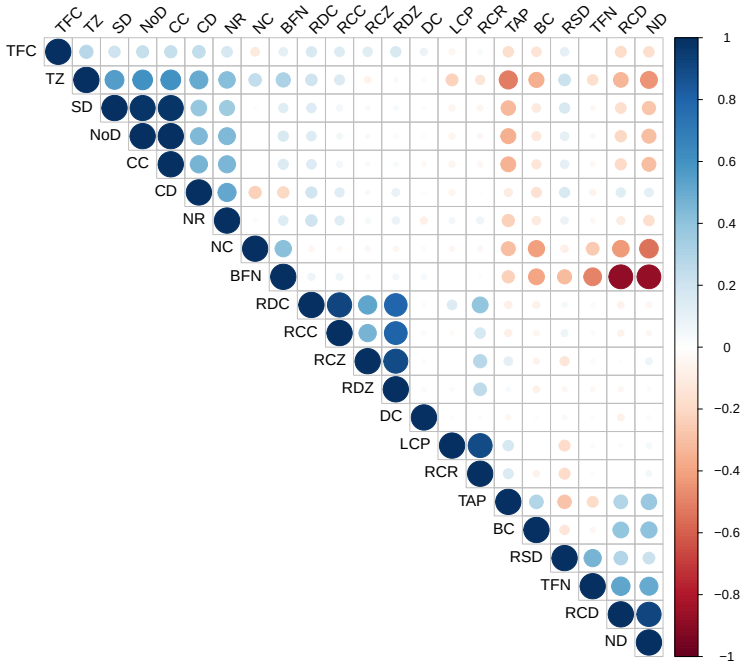

Figure 4: Visualization of the metrics correlation analysis.

To minimize collinearity among our detector variables, for each pair of metrics having a ρ correlation higher than 0.8, we remove one of the variables. We repeated the technique until there was no pair of metrics that met the criteria. After the correlation analysis, we excluded the following metrics with the highest correlation: NoD, CC, RCD, and ND.

3.4. Data preparation pipeline

As several ML algorithms are available, we considered in our study the adopted C4.5 model, along with five widely used ML models, *JRip*, *Random Forest*, *Naïve Bayes*, and two SVM algorithm instances, *SMO*, and *LibSVM* implemented in Weka [34]. We first randomly divided our dataset (cf. Section 3.1) into training, validation, and test datasets following Hastie et al. guidelines [35]. We used the training dataset to fit each individual classifier model, and used the validation dataset to estimate the prediction error for our models comparison and selection as well as hyper-parameters tuning, and finally left

the test dataset to assess the final chosen models. In our experiments, we used 50% of the data as training dataset, 25% as validation dataset, and 25% as test dataset [35].

Moreover, as we deal with an imbalanced dataset, we opted for a common resampling technique by applying the Synthetic Minority Oversampling TEchnique (SMOTE) to over-sample minority classes as suggested by Turhan [36] in building software engineering prediction models. As for the models training and testing, we randomly split our dataset into three parts: training (50%), validation (25%), and test (25%). We use the training dataset to fit each individual classifier model, and the validation dataset to estimate the prediction error/performance for the model's comparison and selection, finally the test dataset to assess the performance of the final studied models.

4. Empirical Study Setup

In this section, we present our empirical study to evaluate our approach. We define four research questions to be addressed and present the experimentation motivation and setup. We then present and discuss the obtained results.

4.1. ***RQ1:*** *To what extent can the employed learner model efficiently detect community smells?*

Motivation. This first RQ is a sanity check to validate whether our machine learning model could accurately detect the different types of community and does not have a bias towards the detection of any specific smell type.

Approach. To answer RQ1, we perform a comparative study between our used machine learner C4.5 and five widely used models, *JRip*, *Random Forest*, *Naïve Bayes*, and two SVM algorithm instances, *SMO*, and *LibSVM*. The data preparation pipeline is explained in Section 3.4.

Our models performance comparison is based on widely-used machine learning performance metrics, mean accuracy, F-Measure, and AUC (Area Under The Curve) [37, 35]. Furthermore, we measure for each algorithm its kappa

statistic, and the Root Mean Squared Error (RMSE). Moreover, to compare the different classifiers performance with respect to the eight community smells, we use the non-parametric statistical test Wilcoxon in a pairwise fashion to detect performance differences between the compared algorithms. While the Wilcoxon test verifies the statistical significance of the outcomes, it does not control the difference in magnitude. Hence, we use the non-parametric effect Cliff's delta (δ) [38] to compute the effect size. The value of effect size is statistically interpreted as :

- Negligible if $\mid \delta \mid < 0.147$,
- Small if $0.147 \leq \mid \delta \mid < 0.33$,
- Medium if $0.33 \leq \mid \delta \mid < 0.474$, or
- High if $\mid \delta \mid \geq 0.474$.

Moreover, to ensure a fair comparison between the used algorithms, we provide in Table 3 the different default parameters used in our comparative study.

4.2. ***RQ2:*** *How does the proposed approach perform compared to state-of-the-art community smells detection approaches?*

Motivation. Although the first research question serves as a sanity check to assess the efficiency of learning techniques for the community smells detection problem, RQ2 aims at evaluating the efficiency of the proposed approach compared to recent state-of-the-art techniques for community smells detection to see what improvement can our approach bring.

Approach. We apply our approach, csDetector, to detect the set of community smells identified from our test dataset of 74 open-source software projects as detailed in Section 3.1. We, thereafter, compare our approach with two recent state-of-the-art tools, namely CodeFace4Smells [13], and Truck Factor [14]. CodeFace4Smells is implemented based on a Siemens tool [39] which can detect the four following community smells considered in our experiments: Organizational Silo Effect (OSE), Radio-silence (RS), Prima-donnas Effect (PDE), and

Table 3: Algorithms parameters configuration.

Algorithm	Parameter	Value
C4.5	minNumObj	2
	subteeRaising	True
	confidenceFactor	0.25
	unpruned	False
	useLaplace	False
JRip	batchSize	100
	checkErrorRate	True
	numDecimalPlaces	2
Random Forest	maxDepth	10
	numfeatues	24
	numTree	100
	seed	1
Naïve Bayes	usekernelEstimator	False
	useSupervisedDiscretization	False
SMO	buildLogisticsModels	False
	c	1.0
	epsilon	1.0e-12
	filterType	Nomalize training data
	kernel	PolyKernel
	casheSize	250007
	exponent	1.0
	toleranceParamter	0.001
LibSVM	SVMType	C-SVC classification
	doNotReplaceMissingValues	False
	Kernel	RBF
	batchSize	100

Black Cloud Effect (BCE). Truck Factor is another specialized tool that is designed to particularly detect the Truck Factor Smell (TFS). Our comparison is based on the detection accuracy. The accuracy is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

where TP refers to True Positive, FP for False Positive, FN for False Negative, and TP for True Negative. The values are calculated according the confusion matrix in Table 5.

Table 4: A summary of the identified smells in each studied system.

System	#projects	OSE	BCE	PDE	SV	OS	SD	RS	TF
Apache	38	29	20	7	2	13	21	3	13
Eclipse	7	4	5	1	0	4	2	0	4
KDE	9	9	3	5	3	5	2	4	4
Ganttproject	1	1	0	0	0	0	1	0	1
Qemu/qemu	1	1	1	0	0	1	0	0	0
Nginx/ngin	1	1	0	0	0	0	1	0	0
Bitcoin/bitcoin	1	0	1	1	0	1	0	0	0
Python/cpython	1	0	1	1	0	1	0	0	0
Rails	1	1	1	1	0	1	0	0	0
Audacity	1	1	0	0	0	1	0	1	0
GitLab	1	1	1	1	0	1	0	0	0
Scala	1	0	1	1	0	1	0	0	0
Torando	1	1	1	1	1	1	0	1	1
Arduino	1	1	1	1	0	1	0	0	1
Capistrano	1	1	1	1	1	1	0	1	0
liferay	1	1	1	0	0	0	1	0	1
The practical dev	1	1	1	0	0	0	0	0	1
Floweisshardt/atf	1	1	0	0	0	0	1	0	0
Cloudera	1	1	1	0	0	1	0	0	1
Pdfsam	1	1	0	0	0	0	0	0	0
Squirrel-sql	1	1	0	0	0	0	0	0	1
Direct memory	1	1	1	0	0	0	0	0	1
Flue	1	1	1	0	0	1	1	0	1
Total	74	59	42	21	7	34	30	10	30

Table 5: The confusion matrix to calculate the accuracy.

		Actual	
		Positive	Negative
Detected	Positive	TP	FP
	Negative	FN	TN

4.3. ***RQ3.*** *What are the most influential characteristics that can indicate the presence of community smells?*

Motivation. The C4.5 classifier results show that we can use a variety of metrics to identify community smells. In addition to the detection accuracy, it is interesting to chart community smells management plans to examine the

influence of characteristics of each community smell type (*i.e.*, smells symptoms that are characterized through metrics). Identifying the most influential characteristics of each community smell could be used as an indicator by software project manager to avoid such smells in their organizations. Therefore, in this RQ3, we set out to analyze our model to investigate the characteristics that influence the existence of community smells.

Approach. To answer RQ3, we evaluate the influence and contribution of each metric on the different types of community smells achieved by our C4.5 classifiers. We design an experimental study to estimate variables importance in a random classification [40, 41]. For each community smell type, we use this technique to compute the Mean Decrease Accuracy (MDA) for each metric. The larger the MDA of the metric is, the more influential the metric to the model is. This experiment is performed using the environment of the *importance* function of the R *randomFoest* package. Moreover, we measure the direction of the relationship between each metric and the likelihood of the occurrence of community smell. To do so, we use a Spearman rank correlation (ρ) [33] to estimate the correlation between each metric and the response (*e.g.*, 1 if a metric indicates the occurrence of a community smell, and 0 if no smell). A positive correlation in terms of a Spearman rank indicates that the metric has a positive relationship with the likelihood of the occurrence of a community smell, whereas a negative correlation identifies an inverse relationship.

4.4. ***RQ4.*** *Can the proposed metrics improve the performance of community smells detection?*

Motivation. Just like code smells [42], community smells definitions and symptoms are subject to various interpretations, and therefore it can be captured using various metrics. To this end, our aim in extending the list of existing soci-organizational and technical metrics (*i.e.*, metrics tagged as "*New*" in the last column of Table 1) was to cover more salient smells symptoms. Indeed, extending the base of metrics allows to properly characterize community smells properties along with extending the feature space for the classifiers.

Approach. We measure the classification performance, with and without the set of newly introduced metrics (cf. Table 1), *e.g.*, *Number of active days of an author on a project*, *Ratio commits per community*, *Ratio developers per community*, *Ratio commits per time zone*, *Ratio developers per time zone*, *Number of releases in a project* and *Ratio commits per release*. An increase in terms of detection accuracy indicates that the new metrics are useful to better capture the salient characteristics of community smells, whereas a non-increase indicates that the introduced metrics do not extend the classifiers feature space, and thus, they are not relevant to capture smells properties.

4.5. ***RQ5.*** *What is the sensibility of our model with respect to outlier/influential data points?*

Motivation. While we deal with a relatively limited size dataset, we observe from the dataset that the majority of the examined projects come from either the Apache, Eclipse and KDE communities. In particular, we observe that there are clusters of similar projects in the whole dataset, with notable differences among them as can be seen in Figure 2 and Table 2. It is thus not clear to what extent these clusters may have an impact on the accuracy and reliability of the obtained results. This suggests the importance of conducting a sensibility analysis to evaluate the possible presence of outliers and/or influential data points that may exert an impact on the classification results. Studying influential instances helps to identify which training instances should be checked for errors and give an impression on the robustness of the model. Indeed, we might not trust a model if a single instance has a strong influence on the model predictions and parameters [43, 44].

Approach. We study the presence of outliers and/or influential data points. We study the extent to which deleting one of the training instances can affect the resulting model using the deletion diagnostics [43]. A training instance is considered "influential" if its deletion from the training data considerably changes (increases or decreases) the prediction performance of the trained model. The measurement of the influence for all 74 training instances requires to train the

model once on the training data and then retrain it n times (*i.e.*, the size of training data) with one of the instances removed each time. We use the influence measure defined by Molnar [43] for the effect on the model predictions which is defined as follows:

$$Influence^{(-i)} =\mid \hat{y}_j - \hat{y}_j^{(-i)} \mid \tag{2}$$

where $Influence^{(-i)}$ returns the differences between prediction performance value of the model with and without the i-th instance, over the dataset. The general form of deletion diagnostic measures consists of choosing a performance measure and calculating the difference of the measure for the model trained on all instances and when the instance is deleted. In our experiment, we use the accuracy measure as a proxy of the model performance. We apply the deletion diagnostic of each model of the eight community smell types.

5. Results and Discussions

In this section, we present and discuss the results of our empirical study with respect to our research questions RQ1-5 set out in Section 4.

5.1. Results for RQ1 : Detection Accuracy

Table 6 reports the obtained results for RQ1 to compare the six considered machine learners C4.5, Random Forest, JRip, SMO, LibSVN and Naïve Bayes in terms of accuracy, Kappa statistic, RMSE and AUC. In more details, Figure 5 shows the boxplots of each of the eight considered community smells. We observe from both the table and the figure that the C4.5 classifier is the most accurate algorithm achieving an average accuracy score of 96.93% while the LibSVM turns out to be the worst algorithm with a limited average accuracy of 61.65%. A high performance is also achieved by the C4.5 classifier in terms of both performance metrics, Kappa (0.89) and RMSE (0.14). However, in terms of AUC, Random Forest performs better than the other classifier by an average AUC of 0.95 which is slightly better than C4.5 (0.94). The Wilcoxon statistical

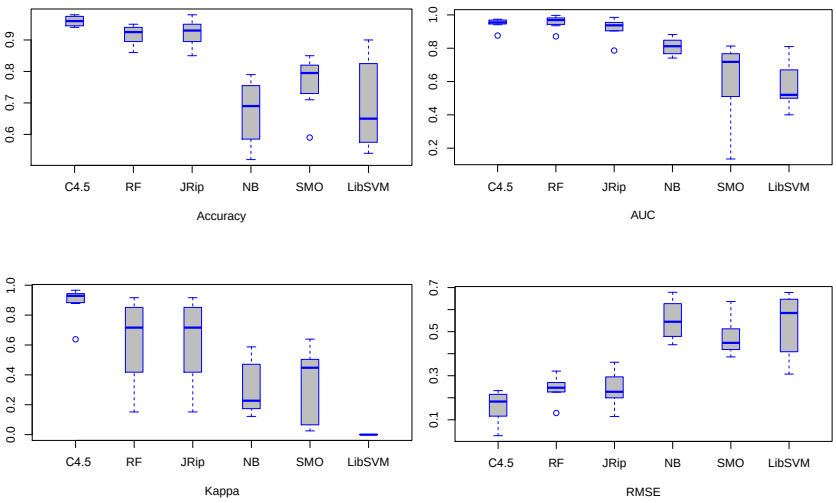

Figure 5: Boxplots of the achieved Accuracy, AUC, Kappa, and RMSE results for the eight considered community smells by each of the considered machine learners, C4.5, JRip, LibSVM, Naïve Bayes, Random Forest and SMO.

analysis of the results depicted in the boxplots of Figure 5 indicates that C4.5 is statistically different from the five other classifiers with a high effect size, except the AUC metric when comparing C4.5 and Random Forest. We thus conclude that C4.5 is the best algorithm choice among the compared machine learners.

Table 6: The average algorithms performance in terms of Accuracy, Kappa statistic, RMSE, and AUC.

Algorithm	Accuracy	Kappa	RMSE	AUC
C4.5	**96.93**	**0.89**	**0.14**	0.94
Random Forest	93.28	0.65	0.22	**0.95**
JRip	93.92	0.82	0.24	0.91
SMO	76.87	0.79	0.48	0.8
LibSVM	61.65	0	0.52	0.7
Naïve Bayes	68.70	0.30	0.54	0.5

To get more detailed results, we assess the accuracy detection for each individual community smell type. Table 7 reports the precision, recall and

F-measure results achieved by the compared algorithms for both classes, i.e., smelly class (S), and non-smelly class (NS). For the smelly class, we observe, from all the considered community smells, that our approach achieved promising detection results with an average precision of 96% and a recall of 91%. The lowest detection results were achieved in the detection of the Sharing Villainy smell (SV), with a precision of 81% and a recall of 59%. This could be due to the low number of smell instances (7 instances) in the base of examples that could be extended to improve the detection performance.

An interesting aspect that is noticed from our study is related to the nature of our dataset which is highly imbalanced. Indeed, not all community smell types are frequent in real-world systems (cf. Table 4). For example, the Sharing Villainy (SV) smell has the lowest number of smelly instances (7 projects) while 67 projects are not smelly, as it is less frequent compared to other more common smells. On the other side, the Organization Silos Effect (OSE) smell is found in 59 projects while the remaining 15 projects in our datset are non-smelly. Indeed, the class imbalanced datasets occur in many real-world applications and in particular in software engineering, where the class distributions of data are often highly imbalanced. Data resampling is a common approach to solve this problem. We used the Synthetic Minority Over-sampling Technique (SMOTE) [45] to deal with such severely imbalanced data. To get more detailed results, we present the precision, recall and F-measure for both classes, *i.e.*, smelly and non-smelly classes, in Table 7. We observe from the results that C4.5 achieves an average precision and recall score of 96% and 91%, respectively, for the smelly smelly class (S). For the non-smelly class (NS), the average precision and recall are 96% and 97%, respectively. In particular, for the most imbalanced smell, *i.e.*, SV, C4.5 achieved a relatively acceptable precision and recall of 81% and 59%, respectively for the smelly class (S) even only 7 instances are available. The achieved precision and recall scores are 95% and 98%, respectively, for the non-smelly class (NS). On the other hand, the libSVM and SMO turn out to be the worst classifiers, in particular for the smelly class, as can be shown Table 7. Regarding the balance between precision and recall, the best classifier is C4.5

which achieved the highest F-measure in both classes S and NS for the 8 smell types.

To get a more qualitative sense, we present in Table 8 six examples from our experiments that are detected by our approach. Looking, for instance, at the `Apache/mahout` project, we observe that this project experiences two community smells, the organizational silo effect (OSE), and the solution defiance (SD). Hence, the project has been developed in a time window that starts on January 14th, 2008 as a first commit until June 26, 2018 with a life time of 3,815 active days when we analyzed it. The Apache mahout project has a total number of 44 developers (NoD) from which 6 are sponsored developers (SD), and 3 communities (NC) and a low graph degree centrality (DC) of 0.008 based on its developers social network. In addition, the total number of time zones is 10 to which developers involved in the project belong to. Indeed, the number of communities in a project indicates the strength and size of the structure for which the different sub-communities in a project and low social network degree centrality may reveal potential disconnections within a community [13].

On the other hand, the SD smell occurs in this project with different levels of cultural and background within the software development community, and could be related to the high number of different time zones in a project (TZ) which is 10 time zones, and relatively low social network density (ND) which is equals to 0.257. Indeed, the high TZ metric indicates a high geographical location dispersion within the community, where different cultural backgrounds may contribute to dividing developers into subgroups based on cultural and linguistic differences [31]. Hence, different circumstances lead to the occurrence different smells such as *organizational silo effect smell (OSE)* and a *social defiance smell (SD)* in the project.

Table 7: The achieved results by each classifier for the studied smell types: S−Smelly Class, NS−Non Smelly Class; P−Precision, R−Recall, F−F-Measure.

Classifier	Class	OSE			PCE			PDE			SV			OS			SD			RS			TF			Average		
		P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
C4.5	S	**0.98**	0.96	**0.97**	0.94	0.96	**0.95**	**1**	**0.96**	**0.98**	**0.81**	0.59	**0.68**	0.94	**0.97**	**0.95**	0.97	**0.96**	**0.96**	**1**	**0.91**	**0.95**	**1**	**0.94**	**0.97**	**0.96**	**0.91**	**0.93**
	NS	**0.88**	**0.93**	**0.95**	**0.96**	0.92	**0.94**	**0.98**	**1**	**0.99**	0.95	0.98	**0.97**	**0.97**	0.95	**0.96**	**0.97**	0.97	**0.97**	**0.98**	**1**	**0.99**	**0.95**	**1**	**0.97**	**0.96**	**0.97**	**0.97**
JRip	S	**0.98**	0.96	**0.97**	0.89	0.83	0.86	0.82	0.90	0.86	0.42	0.42	0.42	**1**	0.91	**0.95**	**1**	0.9	0.94	**1**	**0.9**	**0.94**	0.96	0.8	0.87	0.88	0.83	0.85
	NS	0.87	**0.93**	0.90	0.8	0.87	0.83	0.96	0.92	0.94	0.94	0.94	0.94	0.93	**1**	**0.96**	0.93	**1**	0.96	**0.98**	**1**	**0.99**	0.87	0.97	0.92	0.91	0.95	0.93
LibSVM	S	0.80	**1**	0.89	0.58	**1**	0.73	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0.25	0.27
	NS	0	0	0	0	0	0	0.72	**1**	0.84	0.90	**1**	0.95	0.57	**1**	0.72	0.60	**1**	0.75	0.86	**1**	0.92	0.60	**1**	0.75	0.53	0.75	0.62
NB	S	0.93	0.47	0.62	0.87	0.69	0.77	0.54	0.61	0.57	0.20	**0.71**	0.32	0.82	0.70	0.76	0.6	0.2	0.3	0.26	0.8	0.4	0.45	0.9	0.60	0.58	0.64	0.54
	NS	0.29	0.86	0.44	0.68	0.87	0.76	0.84	0.79	0.81	**0.96**	0.71	0.82	0.77	0.87	0.82	0.62	0.90	0.74	0.95	0.65	0.77	0.8	0.27	0.40	0.74	0.74	0.70
RF	S	0.92	0.96	0.94	**0.95**	0.90	0.92	0.91	0.90	0.90	0.37	0.16	0.22	0.96	0.91	0.93	0.93	**0.96**	0.95	**1**	0.6	0.75	0.91	0.73	0.81	0.87	0.77	0.80
	NS	0.84	0.68	0.75	0.89	**0.93**	0.91	0.96	0.96	0.96	0.91	0.97	0.94	0.92	0.97	0.95	**0.97**	0.95	0.96	0.94	**1**	0.97	0.85	0.95	0.90	0.91	0.93	0.92
SMO	S	0.81	0.96	0.88	0.77	0.73	0.75	0.83	0.47	0.60	0	0	0	0.92	0.67	0.78	0.5	0.23	0.31	0	0	0	0.8	0.53	0.64	0.58	0.45	0.50
	NS	0	0	0	0.68	0.71	0.69	0.82	0.96	0.88	0.90	**1**	0.95	0.77	0.95	0.85	0.61	0.84	0.71	0.86	0.98	0.92	0.74	0.90	0.81	0.67	0.79	0.73

Table 8: Examples of projects and related smells.

Project	OSE	BCE	PDE	SV	OS	SD	RS	TFS
Apache/mahout	✔					✔		
Ganttproject	✔					✔		✔
Qemu/qemu	✔	✔			✔			
Bitcoin/bitcoin		✔		✔	✔			
GitLab	✔	✔	✔		✔			
KDE/okular	✔		✔	✔	✔		✔	

Summary for RQ1. Our machine learning-based approach, csDetector, using C4.5 achieves a high performance for the community smells detection problem with an average accuracy of 96.9%, and an AUC of 0.94.

5.2. Results for RQ2: State-of-the-art comparison

Table 9 shows the comparison results achieved by csDetector compared to two recent state-of-the-art techniques, CodeFace4smell [13] and Truck Factor [3]. CodeFace4smell can detect only four out the eight considered community smells namely, OSE, BCE, PDE and RS, while Truck factor is particularly design to detect the TFS smell. We observe from the table that the csDetector outperforms both state-of-the-art techniques in terms of the detection accuracy. For the eight considered smells, csDetector achieves an accuracy in the interval from 94% (for the BCE smell) to 98% (for PDE and RS), while CodeFcae4Smell achieved an accuracy score within the interval 33% (for the OSE smell) to 86% (for the RS smell). Moreover, as shown in Table 9, although the Truck Factor approach is particularly designed for the detection of the TFS, its accuracy does exceed 84%, while our approach achieves 97% for the truck factor smell.

Indeed, one of the limitations of CodeFace4Smells resides in its adopted smells-based detection rules which are based mainly on generic collaboration and communication metrics through a developer social network (based on the change history and mailing lists). Such rules may need a high calibration effort to be adapted to particular contexts. Moreover, its accuracy highly depends on the availability and completeness of the mailing communication traces. On the other hand, our approach learns from real world instances of community smells and uses a wide variety of socio-organizational and technical metrics to capture the key symptoms from different smell types which resulted in an improved detection accuracy.

Table 9: The comparison results of csDetector, CodeFace4smell and Truck Factor tools for community smells detection in terms of detection accuracy.

Smells	Detection Accuracy		
	csDetector	CodeFace4Smell	Truck Factor
OSE	96%	33%	-
BCE	94%	75%	-
PDE	98%	75%	-
SV	95%	-	-
OS	95%	-	-
SD	97%	-	-
RS	98%	86%	-
TFS	97%	-	84%

Summary for RQ2. Our approach, csDetector, outperforms two recent state-of-the-art techniques CodeFace4smell and Truck Factor with a high detection accuracy from 94% to 98% for the eight considered community smells.

5.3. Results for RQ3: Influential characteristics

Table 10 summarizes all the existing correlations between each community smell type and the list of organizational-social metrics, as well as the average

MDA for each metric. To identify the metrics that are highly influential in the detection of a particular community smell, we compute the average MDA for each metric. From table 10, we observe that the most influential metrics belong initially to the geographic dispersion metrics dimension and the social network analysis metrics dimension. In particular, the five most influential metrics are the *Ratio of commits per time zone* (RCZ), the *Ratio of developers per time zone* (RDZ), the *Ratio of developers per community* (RDC), the social *Graph betweenness centrality* (BC), and the social *Graph closeness centrality* (CC).

Furthermore, we identify the influential characteristics of the analyzed projects and the direction of their relationships with the likelihood of the occurrence of a community smell. The positive correlations are identified by the plus sign (+) while the negative ones are identified with a minus sign (-) in the table. For instance, the NoD metric (*the number of developers who modified the code*) is identified as an influential metric to the occurrences of the following community smells : *Organizational Silo Effect*, *Prima-donnas Effect*, *Sharing Villainy*, *Organizational Skirmish*, *Radio Silence* as it has a positive correlation relationship with these smells and negative ones for others. Looking also at the *number of time zones* (TZ) metric, we observe that it has a positive correlation with the following smells, *Black Cloud Effect* (BCE), *Sharing Villainy* (SV) and *Solution Defiance* (SD). These findings suggest that more attention should be payed to these particular socio-organizational characteristics within the software project community to avoid such smells.

Table 10: The analysis results of the most influential metrics on community smells detection.

Metrics Acronym	OSE	BCE	PDE	SV	OS	SD	RS	TFS	MDA
NoD	+	-	+	+	+	-	+	-	3.31
TAP	-	+	-	+	+	+	-	-	-0.99
LCP	+	-	-	-	+	-	-	-	-0.83
DC	+	-	-	-	-	-	+	-	2.29
BC	-	+	-	+	-	-	-	-	4.23
CC	-	+	+	-	+	-	+	-	4.21
TZ	-	+	+	+	-	+	-	-	2.24
CD	+	-	+	+	-	-	+	-	3.96
RCD	-		+	-	-	-	-	+	1.33
SD	+	-	+	-	-	-	-	-	1.67
RSD	+	-	-	-	-	+	-	-	1.35
ND	-	+	-	+	+	+	+	-	1.09
NR	-	-	+	-	-	-	-	-	0.39
RCR	+	-	-	-	-	-	-	-	2.28
RDC	+	+	+	+	-	+	+	-	4.92
NC	-	+	-	-	-	+	+	-	0.12
RCZ	-	-	+	+	-	-	-	-	8.76
RDZ	+	-	+	-	-	+	+	-	6.23
RCC	-	-	-	-	+	-	-	-	1.59
BFN	+	+	-	+	-	-	+	+	-1.70
FN	-	-	-	-	+	-	-	-	0.50
TFP	+	-	-	-	-	-	+	+	-0.57
TFC	-	+	+	-	-	-	+	+	1.96

Summary for RQ3. Our approach identifies the high influential metrics that can be used as indicators of the existence of community smells. The five most influential metrics are *Ratio of commits per time zone*, the *Ratio of developers per time zone*, the *Ratio of developers per community*, the social *Graph betweenness centrality*, and the social *Graph closeness centrality*.

5.4. Results for RQ4: Impact of the new metrics

Table 11 shows the results of our approach for each community smell, with and without the set of new metrics (cf. Table 1), based on the percentage of correctly classified instances (accuracy), kappa statistic, and Root Mean Squared Error (RMSE). We clearly observe from the table that the set of new metrics significantly improves the performance of community smells detection accuracy in all the eight community smell types. For instance, for the OSE smell, the new metrics improved the accuracy from 89% (without the new metrics) to 96% (with the new metrics). Different accuracy improvements were observed also for the PDE, SV, and RS, while the accuracy remains relatively constant for the BCE, OS, SD, and TFS. On average for the eight smell types, the statistical results indicate that the csDetector approach, with the set of new metrics, achieves a high average performance score in terms of accuracy (96.5%), kappa (0.89), and RMSE (0.16) than without the new metrics.

Table 11: The achieved accuracy, Kappa and RMSE results by csDetector with and without the newly considered metrics.

	with new metrics			**without new metrics**		
Smell	Accuracy	Kappa	RMSE	Accuracy	Kappa	RMSE
OSE	**96.12%**	**0.88**	**0.19**	89.23%	0.68	0.32
BCE	94.41%	0.89	0.24	94.62%	0.89	0.23
PDE	**98.42%**	**0.97**	**0.12**	93.35%	0.83	0.26
SV	**94.67%**	**0.64**	**0.23**	91.93%	0.46	0.28
OS	95.48%	0.92	0.21	95.81%	0.92	0.20
SD	97.08%	0.94	0.03	97.12%	0.94	0.16
RS	**98.52%**	**0.94**	**0.12**	95.94%	0.833	0.203
TFS	97.23%	0.94	0.16	97.31%	0.94	0.16
Average	**96.49%**	**0.89**	**0.16**	94.41%	0.18	0.23

Summary for RQ4. The set of new introduced metrics (cf. Table 1) improves the detection performance of four community smell types (OSE, PDE, SV and RS), while maintaining a constant performance in the four remaining smells (BCE, OS, SD and TFS).

5.5. Results for RQ5: Sensibility analysis

The results of the deletion diagnostics analysis are reported in Figure 6. We train the C4.5 models for each considered smell and check if some training instances were influential overall and for a particular community smell type using the deletion diagnostics. Since the detection of smells is a binary classification problem, we measure the influence as the difference in the model accuracy. An instance is influential if the model accuracy considerably increases or decreases on average in the dataset when the instance is removed from the model training data. The mean value of influence measures for all the eight models over all possible deletions is 1.15%. As can be seen in Figure 6, all the models achieve a median influence score ranging from 0.03% to 2.8%. The most stable models were the track factor (TF), organizational silo effect (OSE) and solution defiance (SD), while other models such as the black cloud effect (BCE) and sharity

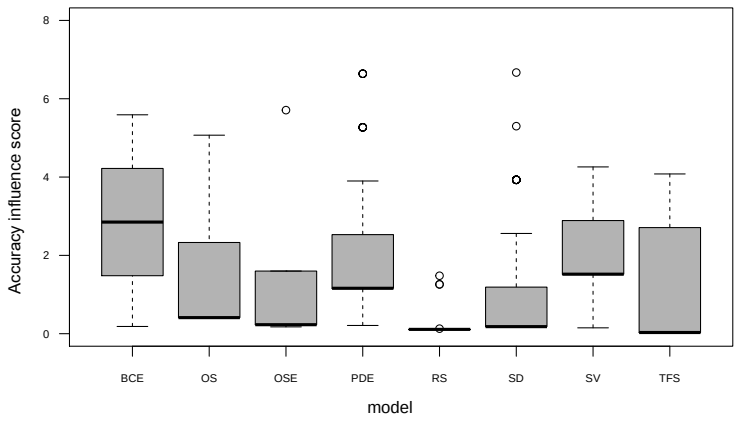

Figure 6: Results of the influential instances analysis on the models accuracy.

villainy (SV) exhibited slightly higher sensitivity. We also observe some outlier values in the OSE, PDE, RS and the SD models. Such influence variability could be related to the highly imbalanced dataset for some smell types such as the SV which have only very few instances. For example, in the SV model, the most influential instance has an influence measure of 4.2% on the accuracy. An influence of 4.2% means that if we remove the instance, the detection accuracy changes by 4.2% on average. This influence is rather low considering that the average accuracy of the SV model is 94.6%. To get a more qualitative sense, we examined this influential instance which is the `Apache/Thrift` project. Looking in deep into the project's characteristics, we found it with a medium number of developers (293) among all SV smell instances in the dataset, however it has the highest number of time zones (21 different time zones across all developers) making it the project with the lowest number of developers per time zone (16). Indeed such project characteristics make it hard to share knowledge among developers due to a potential lack of synchronized communication. Overall, the generated C4.5 models seem to be stable across all the eight community smells. As part of our future work, we plan to extend the training dataset and study the stability of our models on a larger scale.

Summary for RQ5. The sensibility analysis of our models indicate that the built model is relatively stable with a median sensibility to influential data instances of 1.15%.

6. Threats to Validity

In this section, we discuss the potential threats that might have affected the validity of our results.

6.1. Construct validity

Threats to construct validity describe concerns about the relationship between theory and observation and, generally, this type of threats is mainly constituted by imprecisions in performed measurements. Most of what we measured in our approach was based on standard metrics such as precision and recall that are widely accepted as good metrics for the quality of smells detection [46, 47, 48, 49, 50, 51, 52]. Moreover, we exploited the implementation of prediction models by the Weka framework [34], which is commonly considered as a reliable tool. Another potential threat could be related to the selection of classification techniques. Although we use the C4.5 technique which is known to have high overall accuracy, there are several other classification models that may produce a better classification performance. Hence, to mitigate this threat, we built classifiers using different techniques such as JRip, Random Forest, Naïve Bayes, SMO, and LibSVM. We found that our C4.5 classifiers achieve the highest performance.

The different algorithms used in this study are among the popular and widely applied to recent similar software engineering problems [12, 13, 14, 53, 54]. There could be of course several other decision-tree learning algorithms that could be used in our problem, yet we believe that C4.5 well represents this family of algorithms with high performance. As for overfitting, the optimal solution to challenge our model is through another dataset. But since there is no publicly

available set of community smells, we have tested the model using training-validation-test datasets. Our experiments allow the tuning of hyper-parameters with the validation set, and keeps the test set as a truly unseen dataset for assessing the final models performance. One of the issues we had is related to the high data imbalance. Another technical challenge could be related to the amount of data collected to train the machine learning algorithms which is highly imbalanced. While we used the SMOTE technique, we believe that having a larger dataset would allow more generic and reliable data and best performance. Thus, our future work on community smells will focus on how the problem of imbalanced data can be assessed. Moreover, we plan to assess different other algorithms from different families with different parameter settings to reach higher performance.

Moreover, some threats to internal validity could be related to the social network analysis between committers. Indeed, an inherent characteristics of OSS projects is that they may undergo a radical evolution of their committer base, especially, popular projects with large communities and multi-year history. As a result, some of the committers considered in the social network analysis (SNA) metrics computed from these data could be not active anymore. This inherent issue can exert a powerful influence on the calculated metrics as they may change as the project and the development evolve. As part of our future work, we plan to consider the time factor in our SNA (collaboration recency) and see how community smells may evolve over time.

6.2. Internal validity

The internal threat to validity concerns our ability that might have influenced our results, based on the relation between the outcome and the set of organizational-social metrics that are used as independent instances. While we used a collected set of metrics from six different dimensions (*i.e.*, developer social network, social network analysis, community metrics, geographic dispersion, developer network formality, and truck number metrics), adding other new metrics could improve the performance of the classifier. Another impor-

tant threat to validity could be related to the dataset. The manual identification of community smells is up to the point of a certain degree of error. To mitigate at least potential errors, the authors focused on individually identifying the organizational-social symptoms that characterize the existence of a potential community smell based on established state of the art definitions and guidelines [6, 10, 14, 3]. For the sake of a clean data, all projects for which there was no total agreement by the three authors was excluded from the dataset. As part of our future work, we plan to validate and extend our dataset in an industrial context. Another threat to validity could be related to our technique to solve potential committer aliases, based on the Levenshtein distance. This technique may not be robust and may represent some noise in the collected developers identifiers. To better mitigate this issue, we will filter common id strings to increase accuracy, and taking into account the length of e-mail ids as suggested in [55].

6.3. External validity

Threats to external validity are connected to the generalization of the obtained results. The influence of the considered metrics on our model is based on the project characteristics. In this study, we investigated the influence of organizational-social metrics that occur as the most influential metrics. Although some projects may not have the same metrics that are highly influential, nevertheless we believe that our results are still of value to determine what kind of organizational-social metrics can be used as an indicator of community smell. Moreover, our approach currently identifies and quantifies eight different community smells and considers them as indicators that may influence social debt existence; on the other hand, there could exist additional community smells not operationalized yet, that may act as critical indicators of the risk of social debt.

Furthermore, rebalancing was not applied to counteract majority class bias that inherently affects our data. Learning from imbalanced data poses new emerging challenges that need to be addressed to build robust models of knowledge from raw data [56]. Although the results show that our approach can

reasonably learn from both smelly and non-smelly instances for all community smells, replications are needed with larger datasets, using learning techniques specifically designed to deal with skewness in class distribution, to further assess the generality of our models. As part of our future work, we plan to extend our dataset with our industrial partner to add more instances of such smells in order to alleviate possible issues related to such imbalanced datasets.

7. Conclusion

In this paper, we introduced a community smells detection approach, named csDetector, using C4.5 classifier. We built a learning model based on a set of socio-organizational and technical metrics to detect eight common community smells. We also introduced seven novel metrics to better capture and characterize the key symptoms of community smells. To evaluate our approach, we conducted a set of experiments to assess our adopted C4.5 classifier on eight known community smells to a benchmark of 74 open source projects. Results show that the C4.5 classifier achieves an average AUC of 0.94, suggesting that our classifier can be used to detect community smells. Our experiments indicate also that C4.5 achieves the highest performance compared to five widely used classifiers namely, JRip, Random Forest, Naïve Bayes, SMO and LibSVM. Moreover, our results show that our approach outperforms two recent state-of-the-art community smells detection approaches in terms of accuracy. Our study shows also that the ratio of commits and developers per time zone, the social graph betweenness and closeness centrality are the most influential metrics that indicate the presence of community smells.

As future work, we plan to extend our study to other community smell types and projects socio-technical characteristics, while providing ampler empirical evaluation, over multiple open source projects in Github and other software repositories. Moreover, we plan to extend our approach to provide software project manager with community change recommendations to avoid such social debt in their projects. We also plan to assess the impact of community smells

on different aspects of software projects.

Acknowledgements

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] M. M. Lehman, Programs, life cycles, and laws of software evolution 68 (9) (1980) 1060–1076.

[2] G. Navarro, A guided tour to approximate string matching, ACM Comput. Surv. 33 (1) (2001) 31–88.

[3] G. Avelino, L. Passos, A. Hora, M. T. Valente, A novel approach for estimating truck factors, in: IEEE 24th International Conference on Program Comprehension (ICPC), 2016, pp. 1–10.

[4] D. Tamburri, P. Kruchten, P. Lago, H. Vliet, What is social debt in software engineering?, 2013.

[5] D. A. Tamburri, R. Kazman, H. Fahimi, The architect's role in community shepherding, IEEE Software 33 (6) (2016) 70–79.

[6] D. A. Tamburri, P. Kruchten, P. Lago, H. v. Vliet, Social debt in software engineering: insights from industry, Journal of Internet Services and Applications 6 (1) (2015) 10.

[7] F. Palomba, D. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman, A. Serebrenik, Beyond technical aspects: How do community smells influence the intensity of code smells?, IEEE Transactions on Software Engineering (TSE) (2019) 1.

[8] D. A. Tamburri, P. Lago, H. v. Vliet, Organizational social structures for software engineering, ACM Computing Surveys 46 (1) (2013) 3.

[9] M. A. Bindrees, R. J. Pooley, I. S. Ibrahim, D. S. Bental, How public organisational structures influence software development processes, Journal of Computer Science 10 (12) (2014) 2593.

[10] D. A. Tamburri, R. Kazman, H. Fahimi, The architect's role in community shepherding, IEEE Software 33 (6) (2016) 70–79.

[11] N. Almarimi, A. Ouni, M. Chouchen, I. Saidani, M. W. Mkaouer, On the detection of community smells using genetic programming-based ensemble classifier chain, in: 15th ACM/IEEE International Conference on Global Software Engineering (ICGSE), 2020, pp. 1–12.

[12] D. A. Tamburri, F. Palomba, A. Serebrenik, A. Zaidman, Discovering community patterns in open-source: a systematic approach and its evaluation, Empirical Software Engineering.

[13] D. A. A. Tamburri, F. Palomba, R. Kazman, Exploring community smells in open-source: An automated approach, IEEE Transactions on software Engineering.

[14] M. Ferreira, G. Avelino, M. T. Valente, K. A. Ferreira, A comparative study of algorithms for estimating truck factor, in: Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), 2016, pp. 91–100.

[15] Replication package : Learning to detect community smells, 2019.
URL `https://github.com/communitysmells/replicationPackage`

[16] M. Joblin, W. Mauerer, S. Apel, J. Siegmund, D. Riehle, From developer networks to verified communities: A fine-grained approach, in: IEEE International Conference on Software Engineering (ICSE), Vol. 1, 2015, pp. 563–573.

[17] N. Nagappan, B. Murphy, V. Basili, The influence of organizational structure on software quality, in: ACM/IEEE 30th International Conference on Software Engineering, 2008, pp. 521–530.

[18] M. Cataldo, S. Nambiar, The impact of geographic distribution and the nature of technical coupling on the quality of global software development projects, Journal of software: Evolution and Process 24 (2) (2012) 153–168.

[19] J. D. Herbsleb, A. Mockus, An empirical study of speed and communication in globally distributed software development, IEEE Transactions on software engineering 29 (6) (2003) 481–494.

[20] M. Cataldo, S. Nambiar, On the relationship between process maturity and geographic distribution: an empirical analysis of their impact on software quality, in: Joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2009, pp. 101–110.

[21] A. Meneely, L. A. Williams, Socio-technical developer networks: should we trust our measurements?, 33rd International Conference on Software Engineering (ICSE) (2011) 281–290.

[22] A. Tosun, B. Turhan, A. Bener, Validation of network measures as indicators of defective modules in software systems, in: International Conference on Predictor Models in Software Engineering, 2009, pp. 5:1–5:9.

[23] J. Cusick, A. Prasad, A practical management and engineering approach to offshore collaboration, IEEE software 23 (5) (2006) 20–29.

[24] D. A. Tamburri, P. Kruchten, P. Lago, H. van Vliet, What is social debt in software engineering?, in: 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), 2013, pp. 93–96.

[25] C. Bird, N. Nagappan, H. Gall, B. Murphy, P. Devanbu, Putting it all together: Using socio-technical networks to predict failures, in: 20th International Symposium on Software Reliability Engineering, 2009, pp. 109–119.

[26] C. Bird, N. Nagappan, P. Devanbu, H. Gall, B. Murphy, Does distributed development affect software quality? an empirical case study of windows

vista, in: 31st international conference on software engineering, 2009, pp. 518–528.

[27] M. Cataldo, A. Mockus, J. A. Roberts, J. D. Herbsleb, Software dependencies, work dependencies, and their impact on failures, IEEE Transactions on Software Engineering 35 (6) (2009) 864–878.

[28] M. Cataldo, J. D. Herbsleb, K. M. Carley, Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity, in: ACM/IEEE International symposium on Empirical software engineering and measurement, 2008, pp. 2–11.

[29] N. Nagappan, B. Murphy, V. Basili, The influence of organizational structure on software quality: An empirical case study, in: 30th International Conference on Software Engineering, 2008, pp. 521–530.

[30] M. Pinzger, N. Nagappan, B. Murphy, Can developer-module networks predict failures?, in: 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, pp. 2–12.

[31] M. Nordio, H. C. Estler, B. Meyer, J. Tschannen, C. Ghezzi, E. D. Nitto, How do distribution and time zones affect software development? a case study on communication, in: International Conference on Global Software Engineering, 2011, pp. 176–184.

[32] V. Cosentino, J. L. C. Izquierdo, J. Cabot, Assessing the bus factor of git repositories, in: IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2015, pp. 499–503.

[33] J. H. Zar, Significance testing of the spearman rank correlation coefficient, Journal of the American Statistical Association 67 (339) (1972) 578–580.

[34] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The weka data mining software: an update, ACM SIGKDD explorations newsletter 11 (1) (2009) 10–18.

[35] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning: data mining, inference, and prediction, Springer Science & Business Media, 2009.

[36] B. Turhan, On the dataset shift problem in software engineering prediction models, Empirical Software Engineering 17 (1-2) (2012) 62–74.

[37] R. A. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[38] N. Cliff, Dominance statistics: Ordinal analyses to answer ordinal questions, Psychological Bulletin 114 (3) (1993) 494.

[39] S. tool, Codefcae4smell.
URL `https://github.com/maelstromdat/CodeFace4Smells`

[40] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32.

[41] A. Liaw, M. Wiener, Classification and regression by randomforest, Forest 23.

[42] M. Fowler, K. Beck, J. Brant, W. Opdyke, d. Roberts, Refactoring: Improving the Design of Existing Code, 1999.

[43] C. Molnar, Interpretable machine learning, Lulu.com, 2019.

[44] C. A. Scholbeck, C. Molnar, C. Heumann, B. Bischl, G. Casalicchio, Sampling, intervention, prediction, aggregation: A generalized framework for model-agnostic interpretations, in: P. Cellier, K. Driessens (Eds.), Machine Learning and Knowledge Discovery in Databases, 2020, pp. 205–216.

[45] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, Smote: synthetic minority over-sampling technique, Journal of artificial intelligence research 16 (2002) 321–357.

[46] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, A. Marino, Comparing and experimenting machine learning techniques for code smell detection, Empirical Software Engineering 21 (3) (2016) 1143–1191.

[47] F. A. Fontana, M. Zanoni, A. Marino, M. V. Mäntylä, Code smell detection: Towards a machine learning-based approach, in: IEEE International Conference on Software Maintenance, 2013, pp. 396–399.

[48] A. Ouni, M. Kessentini, K. Inoue, M. O. Cinnéide, Search-based web service antipatterns detection, IEEE Transactions on Services Computing (TSC) 10 (4) (2017) 603–617.

[49] M. Kessentini, A. Ouni, Detecting android smells using multi-objective genetic programming, in: IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2017, pp. 122–132.

[50] A. Ouni, M. Kessentini, H. Sahraoui, M. Boukadoum, Maintainability defects detection and correction: a multi-objective approach, Automated Software Engineering 20 (1) (2013) 47–79.

[51] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, K. Deb, Multi-criteria code refactoring using search-based software engineering: An industrial case study, ACM Transactions on Software Engineering and Methodology (TOSEM) 25 (3) (2016) 1–53.

[52] A. Ouni, R. Gaikovina Kula, M. Kessentini, K. Inoue, Web service antipatterns detection using genetic programming, in: Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 1351–1358.

[53] P. Thongtanunam, W. Shang, A. E. Hassan, Will this clone be short-lived? towards a better understanding of the characteristics of short-lived clones, Empirical Software Engineering 24 (2) (2019) 937–972.

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

[55] G. Robles, J. M. Gonzalez-Barahona, Developer identification methods for integrated data from various sources, ACM SIGSOFT Software Engineering Notes 30 (4) (2005) 1–5.

[56] H. He, E. A. Garcia, Learning from imbalanced data, IEEE Transactions on knowledge and data engineering 21 (9) (2009) 1263–1284.