Software Quality Journal (2020) 28:1063–1086
https://doi.org/10.1007/s11219-020-09498-y

Code smell detection using multi-label classification approach

Thirupathi Guggulothu[1] · Salman Abdul Moiz[1]

Published online: 4 April 2020

Abstract
Code smells are characteristics of the software that indicates a code or design problem which can make software hard to understand, evolve, and maintain. There are several code smell detection tools proposed in the literature, but they produce different results. This is because smells are informally defined or subjective in nature. Machine learning techniques help in addressing the issues of subjectivity, which can learn and distinguish the characteristics of smelly and non-smelly source code elements (classes or methods). However, the existing machine learning techniques can only detect a single type of smell in the code element that does not correspond to a real-world scenario as a single element can have multiple design problems (smells). Further, the mechanisms proposed in the literature could not detect code smells by considering the correlation (co-occurrence) among them. To address these shortcomings, we propose and investigate the use of multi-label classification (MLC) methods to detect whether the given code element is affected by multiple smells or not. In this proposal, two code smell datasets available in the literature are converted into a multi-label dataset (MLD). In the MLD, we found that there is a positive correlation between the two smells (long method and feature envy). In the classification phase, the two methods of MLC considered the correlation among the smells and enhanced the performance (on average more than 95% accuracy) for the 10-fold cross-validation with the ten iterations. The findings reported help the researchers and developers in prioritizing the critical code elements for refactoring based on the number of code smells detected.

Keywords Code smells · Software quality · Code smell correlation · Multi-label classification · Code smells detection · Machine learning techniques · Refactoring

This article belongs to the Topical Collection on *Quality Management for Information Systems*
Guest Editors: Mario Piattini, Ignacio García Rodríguez de Guzmán, Ricardo Pérez del Castillo

✉ Thirupathi Guggulothu
thirupathi.gugguloth@gmail.com

Salman Abdul Moiz
salman@uohyd.ac.in

[1] School of Computer and Information Sciences, University of Hyderabad, Hyderabad, Telangana, India

1 Introduction

Code smell refers to an anomaly in the source code that shows the violation of basic design principles such as abstraction, hierarchy, encapsulation, modularity, and modifiability (Booch 1980). Even if the design principles are known to the developers, they are often violated because of inexperience, deadline pressure, and heavy competition in the market. Fowler et al. (1999) have defined 22 informal code smells. These smells have different granularities based on their affected element such as class-level (God class, data class, etc.) and method-level (long method and feature envy, etc.) code smells. One way to remove them is by using refactoring techniques (Opdyke 1992), i.e., a technique that improves the internal structure (design quality) of the code without altering the external behavior of the software.

In the literature, there are several techniques (Kessentini et al. 2014) and tools (Fontana et al. 2012) available to detect different code smells. Each technique and tool produces different results. According to Kessentini et al. (2014), the code smell detection techniques can be classified into seven categories: cooperative based (Abdelmoez et al. 2014), visualization based (Murphy-Hill and Black 2010), search based (Palomba et al. 2015; Liu et al. 2013; Palomba et al. 2013), probabilistic based (Rao and Reddy 2007), metric based (Marinescu 2004; Moha et al. 2010a; Tsantalis and Chatzigeorgiou 2009), symptoms based (Moha et al. 2010b), and manual techniques (Travassos et al. 1999; Ciupke 1999) which differ in the underlying algorithm. Bowes et al. (2013) compared two smell detection tools on message chaining and showed the disparity of results between them. Due to the differing results, Rasool and Arshad (2015) classified, compared, and evaluated existing detection tools and techniques so as to understand the categorization better. There are three main reasons for the disparity in the results: (1) The developers can subjectively interpret the code smells, and hence detected in different ways, (2) agreement between the detectors is low, i.e., several tools or rules recognize different smells for different code elements, and (3) the threshold value for identifying a smell can vary from one detector to another.

To address the above limitations, in particular, the subjective nature, Fontana et al. (2016b) proposed a machine learning (ML) technique (supervised classification) to detect four code smells with the help of 32 classification techniques. The authors showed that most of the classifiers achieved more than 95% performance in terms of accuracy and F-measure. After observing the results, authors have suggested that ML classifiers are the most suitable approach for the code smell detection. Di Nucci et al. (2018) addressed some of the limitations of Fontana et al. (2016b); one of the drawback reported is that the prepared datasets do not represent a real-world scenario, i.e., in the datasets, metric distribution between smelly and non-smelly instances is highly variant. This may enable the ML classifiers to easily distinguish two classes (smelly and non-smelly). In the real-time environment, the boundary between smelly and non-smelly characteristics is not always clear (Tufano et al. 2017; Fontana et al. 2016a). To avoid this limitation and simulate more realistic datasets, Di Nucci et al. (2018) configured the datasets of Fontana et al. (2016b) by merging the class-level and method-level datasets, respectively. The merged datasets have reduced the metric distribution and maintain more than one type of code smell instances. The authors experimented the same ML techniques of the Fontana et al. (2016b), on revised datasets and achieved an average of 76% accuracy in all models. The authors claimed that the performance of ML classifiers is reduced when dataset represents the real-time scenario.

In this work, we addressed the reason why ML classifier performed less in the Di Nucci et al. (2018) and showed that ML classifiers perform good even under the real-time scenario. That is, in the merged datasets of Di Nucci et al. (2018), some of the instances are identical but they are assigned as different (one is smelly and another one is non-smelly) decision labels called disparity. Due to disparity instances in the datasets, the classifiers performed poorly in their work. Di Nucci et al. (2018) considered that the long method dataset has an instance which is smelly, if the same instance exists in feature envy dataset irrespective of smelly or non-smelly, then that instance of feature envy is merged into long method as non-smelly. Now, long method dataset has same instance with 2 different decision labels called disparity instance. This disparity will confuse the ML algorithms and result in poor performance of Di Nucci et al. (2018). In this paper, we have removed the disparity instances in the method-level merged datasets and experimented same tree-based classifier techniques on them. The performance achieved in our work is similar to the performance of Fontana et al. (2016b).

From the datasets of Fontana et al. (2016b) and Di Nucci et al. (2018), we have observed that there are 395 common instances in method level, which are labeled with the two smells. These instances led to an idea to form a multi-label dataset. Through this dataset, the disparity can be eliminated, maintained similar metric distribution as in Di Nucci et al. (2018), and more than one smell can be detected. In the literature (Azeem et al. 2019; Pecorelli et al. 2019b; Zaidi and Colomo-Palacios 2019), only one code smell was detected for the same method with the help of ML (single label) classifier. In addition to it, no one has detected the code smells by considering the correlation among them. In this paper, we formulate the code smell detection as a multi-label classification (MLC) problem. The two factors (multiple smell detection, correlation) can be achieved through the methods of MLC. It is important for the developers to detect multiple code smells so that they (developers) can schedule the smells accordingly for refactoring. The effort required in refactoring varies from one smell to the other as they have correlation (one may influence the other) between them. Refactoring such correlated smells results in reducing the developer's effort.

For our study, we have considered two method-level code smell (long method and feature envy) datasets from Fontana et al. (2016b) and converted them into a multi-label dataset (MLD). From the MLD, we found (by using *lift* measure) that there is a positive correlation between the considered smells. Three MLC methods (binary relevance (BR), classifier chain (CC), label combination (LC)) are applied on MLD by using 10-fold cross-validation with the ten iterations. In the classification phase, among the three methods, BR does not consider the correlation. The other two (CC, LC) approaches take advantage of positive correlation and results in improved performances (on average 95%) than the BR (on average 91%).

The structure of the paper is organized as follows; The second section explains the background of ML classification technique and introduces a work related to the detection of code smell using ML techniques; the third section defines the code smells used in preparation of the multi-label dataset; the fourth section describes the proposed approach and addresses few research questions; the fifth section shows the experimental study of the multi-label classification; the sixth section shows the results of the proposed study and answers the research questions; the sixth section outlines the threats to the validity of our work; the final section gives conclusion and future directions.

2 Related work

Over the past fifteen years, researchers have presented various tools and techniques for detecting code smells. According to Kessentini et al. (2014), there are seven different classification categories to detect code smells. They are cooperative-based approaches, visualization-based approaches, machine learning–based approaches, probabilistic approaches, metric-based approaches, symptoms-based approaches, and manual approaches. In this section, we only consider the machine learning approaches for detecting the code smells.

2.1 Machine learning (supervised classification) approaches to detect code smells

Supervised classification is the task of using algorithms that allow the machine to learn associations between instances and decision labels. Supervision comes in the form of previously labeled instances, from which an algorithm builds a model to predict the labels of new instances automatically. Figure 1 shows the working procedure of supervised classification algorithm. In ML, classification is of three types: binary (yes or no), multi-class, and multi-label classification (MLC). In the literature (Azeem et al. 2019), code smell detection is treated as a single label (binary) classifier which detects single type code smell (presence or absence) only. Below is the summary of the related work of code smell detection using single label classifiers.

Kreimer (2005) introduced an adaptive method to find the design flaws (viz., big class/large class and long method) by combining the known approaches based on the software metrics by using classification techniques called decision trees. IYC system and WEKA package are the two software systems used for analysis purpose.

Khomh et al. (2009) have proposed a Bayesian approach to detect occurrences of the Blob anti pattern on open-source programs (GanttProject v1.10.2 and Xerces v2.7.0). Khomh et al. (2011) presented BDTEX (Bayesian Detection Expert) Goal Question Metric approach to build Bayesian Belief Networks from the definitions of anti-patterns and validate BDTEX with blob, functional decomposition, and spaghetti code anti-patterns on two open-source programs.

Maneerat and Muenchaisri (2011) collected datasets from the literature to evaluate seven bad smells. In order to predict those bad smells, seven machine learning algorithms and 27 design model metrics (extracted by a tool as independent variables) are used to detect these smells. The author has not made any explicit references to the dataset.

Maiga et al. (2012) have introduced the SVM Detection approach by using the support vector machine to detect anti-patterns . They have used open-source programs ArgoUML, Azureus, and Xerces to study their subjects such as blob, functional decomposition,

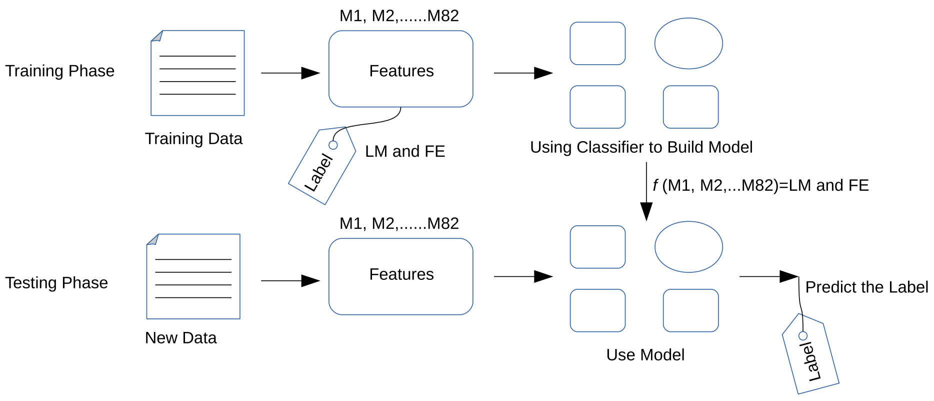


Fig. 1 Working procedure of ML supervised classification technique

spaghetti code, and Swiss army knife anti-patterns. They have enlarged their work by initiating SMURF, taking the practitioner's feedback into account.

Wang et al. (2012) have proposed a method that will help in understanding the destructive nature of predetermined cloning operations by using Bayesian Networks and a set of features such as code, destination, and history.

Yang et al. (2015) studied the decisions of individual users by applying machine learning algorithms on each code clones. White et al. (2016) detected code clone by using deep learning techniques. The authors have sampled 398 files and 480 method levels pairs across eight real-world Java software systems.

Amorim et al. (2015) studied to recognize the code smells through decision tree algorithms. For this, the authors have experimented on four open-source projects and the results were compared with the manual oracle, with existing detection approaches and machine learning algorithms.

Fontana et al. (2016b) experimented 16 ML classification techniques on the four code smell datasets (viz., data class, long method, feature envy, God class) to detect them. For this study, the authors have used 74 Java systems that belongs to the Qualitus Corpus (Tempero et al. 2010).

Fontana and Zanoni (2017) classified the code smells severity by using a machine learning method. This approach can help software developers to prioritize or rank the classes or methods. Multinomial and regression classification techniques are applied for code smell severity classification.

Di Nucci et al. (2018) have covered some of the limitations of Fontana et al. (2016b). The authors configured the datasets of Fontana and provided new datasets that are suitable for real-time scenarios.

Pecorelli et al. (2019a) investigated several techniques to handle data imbalance issues to understand their impact on ML classifiers for code smell detection.

When observed, the major difference of the previous work with respect to the proposed approach is the detection of code smells viewed as multi-label classification. No other approaches have considered the correlation among the smells when detecting the code smell. Table 1 presents a comparison of our study with respect to the referenced papers.

3 Evaluated code smells

In this paper, two method code smells have been used to create the multi-label dataset. These code smells are more frequent and faultm prone or change prone as per literature (Ferme 2013). They also cover different object-oriented quality dimension problems such as complexity, size, coupling, encapsulation, cohesion, and data abstraction (Marinescu 2005).

Table 1 Comparison with prior work and this paper

Study	Detecting multiple code smell detection	Code smell correlation consideration	Dataset applicable for real use-case scenario
Fontana et al. (2016b)	No	No	No
Di Nucci et al. (2018)	No	No	Yes
Pecorelli et al. (2019a)	No	No	No
This paper	Yes	Yes	Yes

Table 2 lists the selected two method code smells with their affected entities and quality dimensions. The below subsections reports the characteristics of selected code smells.

- *Long method:* A code smell is said to be long method when it has many lines of code and uses much of the data of the other classes. This increases the functional complexity of the method and it will be difficult to understand.
- *Feature envy:* Feature envy is the method-level smell which uses more data from other classes rather than its own class, i.e., it accesses more foreign data than the local.

4 Multi-label classification approach for code smell detection

This section describes the methodology followed in the empirical study that includes referred datasets used to prepare the multi-label dataset by using the instances of referred datasets. Then, applied multi-label classification models on them. Figure 2 shows the flow graph (principle steps) of code smell detection using multi-label classification approach and the following subsections describes them briefly:

4.1 Reference datasets

In this paper, we have considered two method-level datasets (long method and feature envy) from Fontana et al. (2016b). In existing literature, these datasets are used for the detection of one smell. In the following subsections, we briefly describe the data preparation methodology of Fontana et al. (2016b). These datasets are available at https://drive.google.com/file/d/15aXc_el-nx4tQwU3khunQ-I5ObSA1-Zb/view [or] http://www.essere.disco.unimib.it/machine-learning-for-code-smell-detection/

4.1.1 Selection of systems

Fontana et al. (2016b) have analyzed Qualitus Corpus software systems which was collected from Tempero et al. (2010). Among 111 systems of the corpus, 74 systems are considered for smell detection. The remaining 37 systems cannot detect code smells as they are not successfully compiled. The sizes (lines of code etc.,) of the 74 Java projects are shown in Table 3. These projects also cover different application domains like database, tool, middleware, and games. The complete characteristics (sizes, release date, etc.) of each project and the domain they belong to are shown in the link that is available at https://github.com/thiru578/Multilabel-Dataset

4.1.2 Metric extraction

For the given 74 software systems, Fontana et al. (2016b) have computed metrics at all levels (project, package, class, method) by using the tool called "Design Features and Metrics for

Table 2 Selection of code smells

Code smell	Affected entities	Impacted on OO quality dimensions
Long method	Method	Complexity, size, coupling, cohesion
Feature envy	Method	Data abstraction, coupling

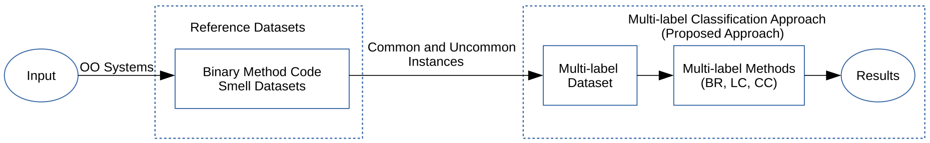

Input
OO Systems
Reference Datasets
Binary Method Code Smell Datasets
Common and Uncommon Instances
Multi-label Classification Approach (Proposed Approach)
Multi-label Dataset
Multi-label Methods (BR, LC, CC)
Results

Fig. 2 Flow graph of code smell detection using multi-label classification approach

Java" (DFMC4J). This tool parses the source code of Java projects through the Eclipse JDT library. The computed metrics became features or attributes to the datasets and cover the different quality dimensions, as shown in Fig. 3. The detail computation of each metric is available at https://github.com/thiru578/Multilabel-Dataset.

4.1.3 Dataset preparation

Automatic code smell detection tools are used by Fontana et al. (2016b) to detect whether the source code element is smelly or not. Table 4 reports the detection tools used to build code smell datasets. For the built dataset, tools produced few false positive results, i.e., tools may not identify actual code smell elements. To cope with this problem, authors have used stratified random sampling on the elements of the considered systems. The sampling method produced 1986 elements (826 smelly elements and 1160 non-smelly elements) which are manually validated by the authors. To normalize the training datasets, the authors randomly removed smelly and non-smelly elements resulted in the formation of 4 datasets. Each dataset has 420 instances, among them, 1/3 (140) are smelly and 2/3 (280) are non-smelly. These datasets (training) are given as input to the ML classification techniques.

4.2 Proposed approach

In this section, we discuss the construction of a multi-label code smell dataset and then present a multi-label classification approach to classify the multiple smells on the prepared dataset.

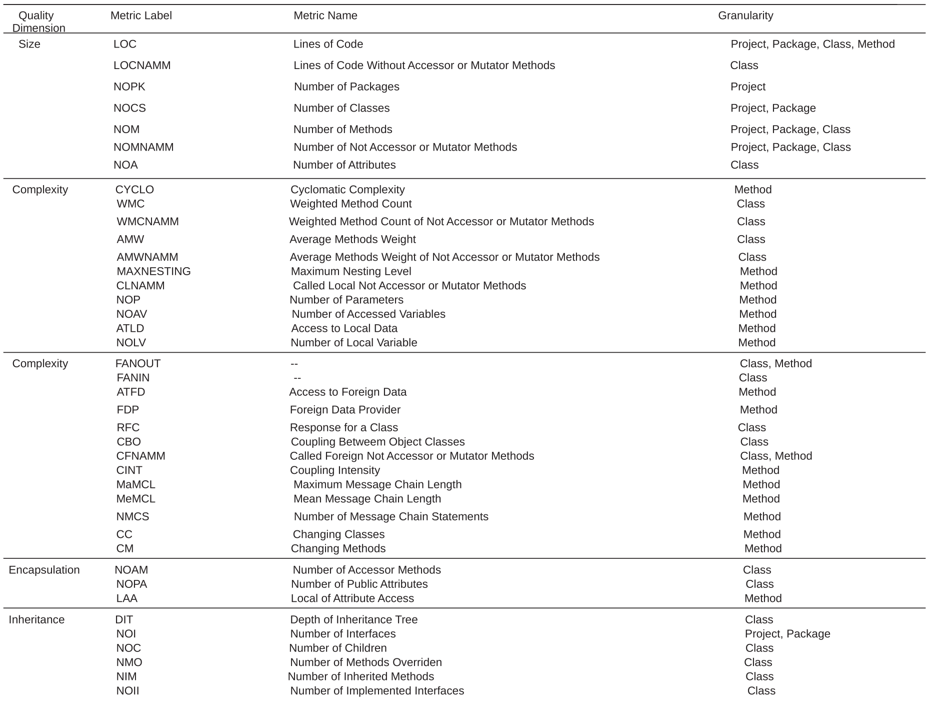
4.2.1 Construction of multi-label dataset

The considered long method and feature envy datasets have 420 instances each, which are used for the construction of a multi-label dataset. The following are the steps involved in creation of the multi-label dataset.

1. Initially, each data set has 420 instances. From those, 395 common instances are added to the multi-label dataset with their corresponding two decision labels.
2. The remaining 25 uncommon instances of the LM dataset acts as input to the top classifier model of FE (Fontana et al. 2016b) that predicts the FE label. Similarly, 25

Table 3 Summary of 74 projects

Number of projects	Number of lines in all projects	Number of packages in all projects	Number of classes in all projects	Number of methods in all projects
74	6,785,568	3420	51,826	404,316



Quality Dimension	Metric Label	Metric Name	Granularity
Size	LOC	Lines of Code	Project, Package, Class, Method
	LOCNAMM	Lines of Code Without Accessor or Mutator Methods	Class
	NOPK	Number of Packages	Project
	NOCS	Number of Classes	Project, Package
	NOM	Number of Methods	Project, Package, Class
	NOMNAMM	Number of Not Accessor or Mutator Methods	Project, Package, Class
	NOA	Number of Attributes	Class
Complexity	CYCLO	Cyclomatic Complexity	Method
	WMC	Weighted Method Count	Class
	WMCNAMM	Weighted Method Count of Not Accessor or Mutator Methods	Class
	AMW	Average Methods Weight	Class
	AMWNAMM	Average Methods Weight of Not Accessor or Mutator Methods	Class
	MAXNESTING	Maximum Nesting Level	Method
	CLNAMM	Called Local Not Accessor or Mutator Methods	Method
	NOP	Number of Parameters	Method
	NOAV	Number of Accessed Variables	Method
	ATLD	Access to Local Data	Method
	NOLV	Number of Local Variable	Method
Complexity	FANOUT	--	Class, Method
	FANIN	--	Class
	ATFD	Access to Foreign Data	Method
	FDP	Foreign Data Provider	Method
	RFC	Response for a Class	Class
	CBO	Coupling Betweem Object Classes	Class
	CFNAMM	Called Foreign Not Accessor or Mutator Methods	Class, Method
	CINT	Coupling Intensity	Method
	MaMCL	Maximum Message Chain Length	Method
	MeMCL	Mean Message Chain Length	Method
	NMCS	Number of Message Chain Statements	Method
	CC	Changing Classes	Method
	CM	Changing Methods	Method
Encapsulation	NOAM	Number of Accessor Methods	Class
	NOPA	Number of Public Attributes	Class
	LAA	Local of Attribute Access	Method
Inheritance	DIT	Depth of Inheritance Tree	Class
	NOI	Number of Interfaces	Project, Package
	NOC	Number of Children	Class
	NMO	Number of Methods Overriden	Class
	NIM	Number of Inherited Methods	Class
	NOII	Number of Implemented Interfaces	Class

Fig. 3 Features of the multi-label dataset

uncommon instances of FE dataset are given as input to the top classifier model of LM (Fontana et al. 2016b) that predicts the LM label. The predicted 50 uncommon instances with their LM and FE decision labels are included in the multi-label dataset.

An overview of the procedure is depicted in Fig. 4. The result of the above steps are formed as training dataset for multi-label classification models. Below is the explanation on the training dataset.

Training dataset Figure 5 shows the representation of a multi-label dataset (training dataset). In the dataset, each row represents method instances and the column represents the features (metrics of all level) and decision variables (whether the method instance is smelly or not). The dataset contains 445 method instances, 82 metrics, and 2 decision variables. As per Di Nucci et al. (2018), the MLD is also applicable for real-time scenarios (reduced metric distribution and maintain different types of smell instances). In general, for each instance (either class or method), the metrics of the respective containers are included as features

Table 4 Automatic code smell detector tools

Code smell	Detectors
Long method	PMD[a] iPlasma (Marinescu 2005, 2002)
Feature envy	Fluid Tool (Nongpong 2012), iPlasma (Marinescu 2005)

[a]http://pmd.sourceforge.net/

in the dataset shown in Fig. 6. Our work is on method-based code smell instances hence included metrics of method, class, package, and project due to the containment relation. The containment relation defines that a method is contained in a class, a class is contained in a package, and a package is contained in a project.

4.2.2 Multi-label classification approach

Multi-label classification (MLC) is a way to learn from instances that are associated with a set of labels (predictive classes). That is, for every instance, there can be one or more labels associated with them. MLC is frequently used in some application areas like multimedia classification, medical diagnosis, text categorization, and semantic scene classification. Similarly, in the code smell detection domain, instances are code elements and set of labels are code smells, i.e., a code element can contain more than one type of smell which was not addressed by the earlier approaches.

The advantage of MLC over single label classification (SLC) is that in the classification phase, the methods of MLC consider correlation among the decision labels (code smells) whereas, in SLC, each smell is classified independently. In the literature, Palomba et al. (2017) stated that considered code smells are co-occurred frequently. The reason behind this co-occurrence is that there is a common symptom (Access to Foreign Data (ATFD))

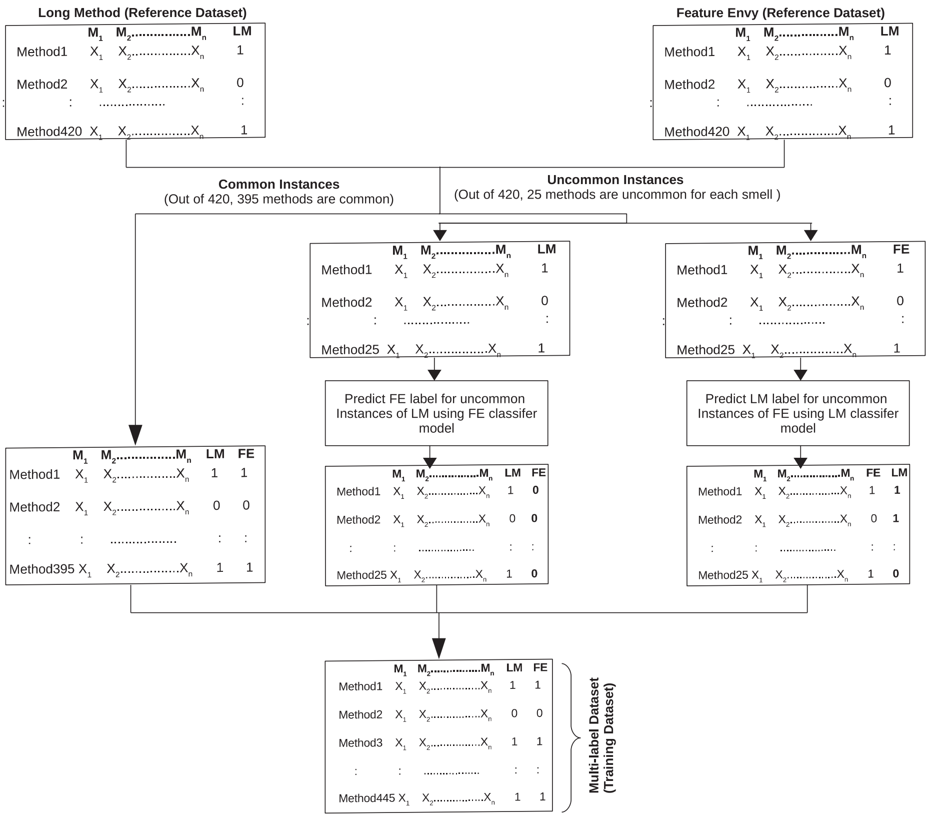

Long Method (Reference Dataset)
Feature Envy (Reference Dataset)
Common Instances
(Out of 420, 395 methods are common)
Uncommon Instances
(Out of 420, 25 methods are uncommon for each smell)
Predict FE label for uncommon Instances of LM using FE classifer model
Predict LM label for uncommon Instances of FE using LM classifer model
Multi-label Dataset (Training Dataset)
Method1
Method2
Method420
Method25
Method395
Method445
LM
FE

Fig. 4 Construction of multi-label dataset

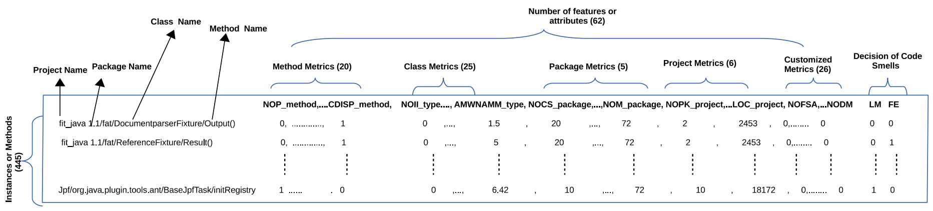

Number of features or attributes (62)
Class Name
Method Name
Project Name
Package Name
Method Metrics (20)
Class Metrics (25)
Package Metrics (5)
Project Metrics (6)
Customized Metrics (26)
Decision of Code Smells
Instances or Methods (445)
NOP_method,....CDISP_method, NOII_type...., AMWNAMM_type, NOCS_package,...,NOM_package, NOPK_project,...LOC_project, NOFSA,...NODM LM FE
fit_java 1.1/fat/DocumentparserFixture/Output() 0, ..., 1 0 ,...., 1.5 , 20 ,...., 72 , 2 , 2453 , 0,........ 0 0 0
fit_java 1.1/fat/ReferenceFixture/Result() 0, ..., 1 0 ,...., 5 , 20 ,...., 72 , 2 , 2453 , 0,........ 0 0 1
Jpf/org.java.plugin.tools.ant/BaseJpfTask/initRegistry 1 .... 0 0 ,...., 6.42 , 10 ,...., 72 , 10 , 18172 , 0,........ 0 1 0

Fig. 5 Multi-label dataset

between the smells, which are defined in Section 3. The correlation between the smells led to improve the classifier performance (Zhang and Zhou 2013) in MLC.

Methods of multi-label classification Two approaches are widely used to handle the problems of multi-label classification (Tsoumakas and Katakis 2007): Problem Transformation Method (PTM) and Algorithm Adoption Method (AAM). In PTM, the multi-label dataset is transformed to single label problem and solved by using appropriate classifiers. In AAM, multi-label dataset is handled by adapting a single label classifier to solve it. In this paper, only PTM is considered.

We have identified a set of specific research questions that guide us to classify the code smell using the multi-label approach:

- **RQ1:** How many disparity instances exist in the configured datasets for the concerned code smells in Di Nucci et al. (2018)?
- **RQ2:** What would be the performance improvement after removing the disparity instances?
- **RQ3:** What percent (confidence) of the method-level code smells are correlated to one another?
- **RQ4:** What would be the classifier performance with and without correlation consideration?

5 Experimental setup

MLC approach is a new perspective for detecting the code smells. In this section, we have explained the MLC experimental setup in detail so that it helps the researcher community

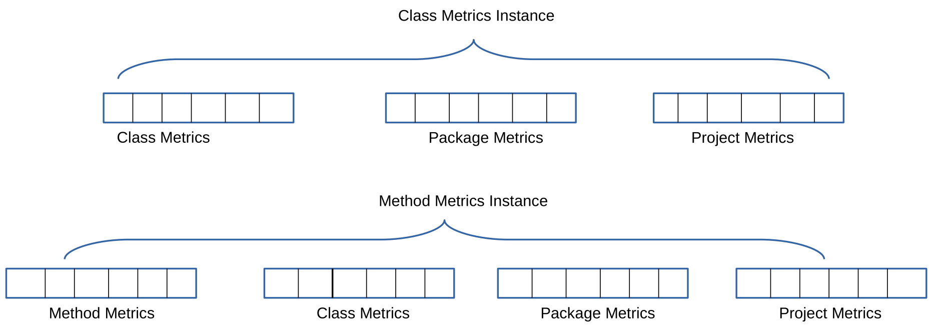
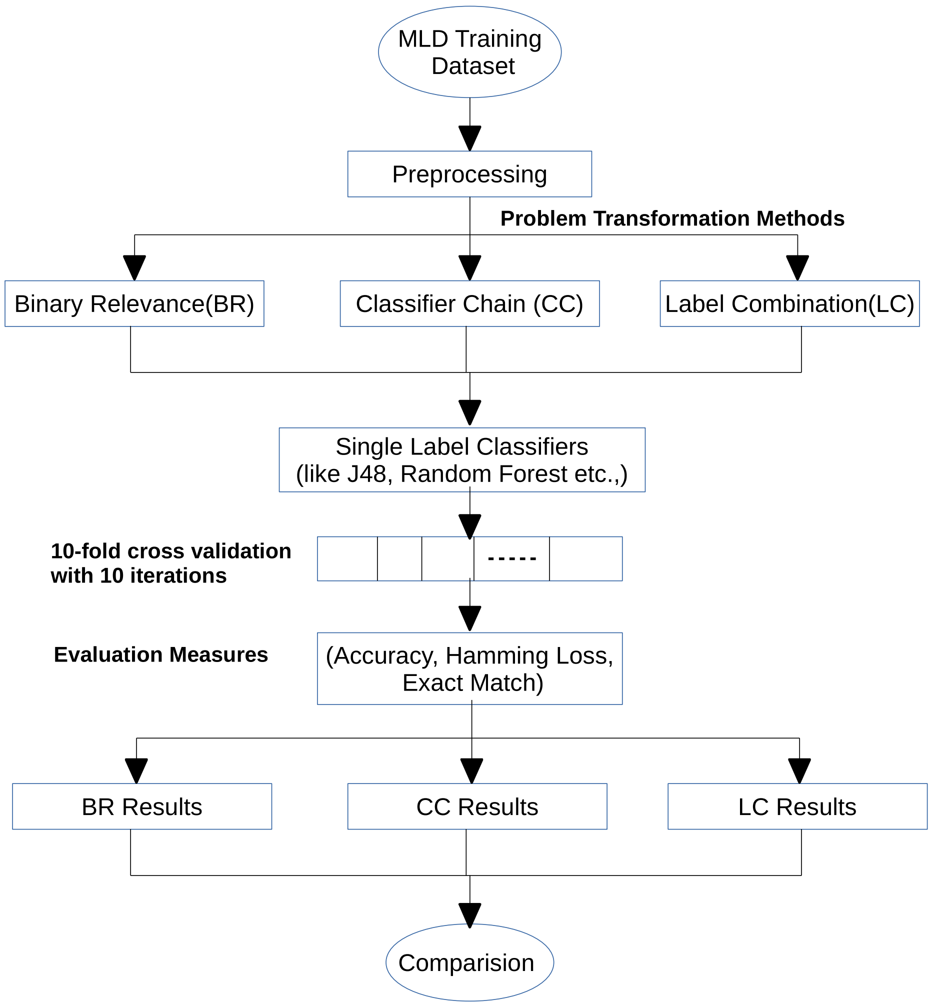


Fig. 6 Class and method metric instance

for further investigation on it. Figure 7 represents the flow of experimental setup for MLC. The phases of the MLC experimental setup are explained as follows:

Pre-processing: The constructed MLD training dataset has 82 metrics. Among them, 25 are class, 5 are package, and 6 are project level metrics which are irrelevant to the method-level code smells because they can not contribute to detection of the method-level code smells. Method metrics can cover all the structural information (coupling of other classes, etc.) of the methods. Before applying ML classifiers on our MLD, the other metrics such as class, package, and method metrics are removed from the dataset.

Problem transformation method: In PTM, multi-label dataset is transformed into single label problem and solved them by using appropriate single label classifiers. Many methods fall under PTM category. Among them, two methods can be thought as foundation to many other methods. (1) *Binary relevance method* (Godbole and Sarawagi 2004):


MLD Training Dataset
Preprocessing
Problem Transformation Methods
Binary Relevance(BR)
Classifier Chain (CC)
Label Combination(LC)
Single Label Classifiers (like J48, Random Forest etc.,)
10-fold cross validation with 10 iterations

Evaluation Measures
(Accuracy, Hamming Loss, Exact Match)
BR Results
CC Results
LC Results
Comparision

Fig. 7 MLC experimental setup

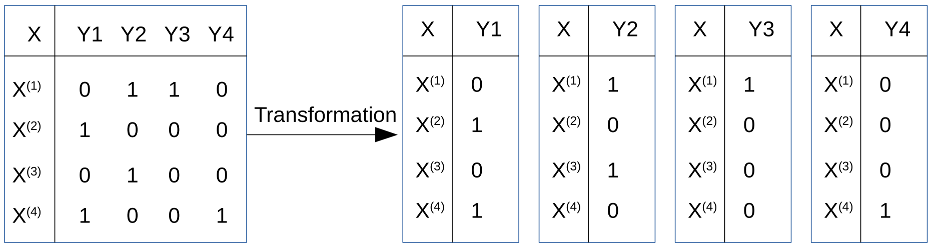

X
Y1 Y2 Y3 Y4
X(1) 0 1 1 0
X(2) 1 0 0 0
X(3) 0 1 0 0
X(4) 1 0 0 1
Transformation
X Y1: X(1) 0, X(2) 1, X(3) 0, X(4) 1
X Y2: X(1) 1, X(2) 0, X(3) 1, X(4) 0
X Y3: X(1) 1, X(2) 0, X(3) 0, X(4) 0
X Y4: X(1) 0, X(2) 0, X(3) 0, X(4) 1

Fig. 8 Procedure of BR transformation method

it will convert a multi-label dataset to as many binary datasets as the number of different labels that are present. All the different dataset predictions from the binary classifiers are merged to get the outcome. (2) *Label powerset method* (Boutell et al. 2004): it is used to convert a multi-label dataset to a multi-class dataset based on the label set of each instance as a class identifier. The predicted classes are transformed back to the label set using any multi-class classifier. In the transformation phase, these two methods do not lose the information unlike max, min, random, etc. where we can lose the information.

Therefore, several methods are developed under binary relevance and label powerset methods. In this paper, two are considered under BR called (1) binary relevance (BR) in itself is a method and (2) classifier chain (CC) and one is consider under LP called label combination (LC). In the following, a short description is reported for these 3 methods and MEKA (Read et al. 2016) tool is provided implementation of the selected methods.

– *Binary relevance:* This is the simplest technique, which treats each label as a separate single class classification problem. It does not model label correlation, i.e., smell correlation. Figure 8 shows an example of the BR transformation method procedure.
– *Classifier chains* (Read et al. 2011): The algorithm tries to enhance binary relevance by considering the label correlation. To predict the new labels, train "Q" (Q indicates the number of binary datasets to split according to the number of labels in multi-label dataset) classifiers which is connected to one another in such a way that the prediction of each classifier is added to the dataset as new feature. Figure 9 shows the example of the CC transformation method procedure.
– *Label combination* (Boutell et al. 2004): Treats each label combination as a single class in a multi-class learning scheme. The set of possible values of each class is a powerset of labels. Figure 10 shows an example of the LC transformation method procedure.

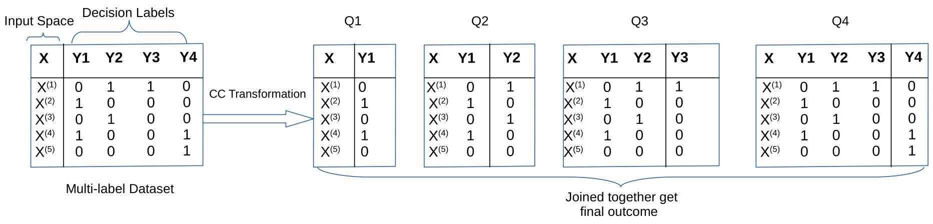


Fig. 9 Procedure of CC transformation method

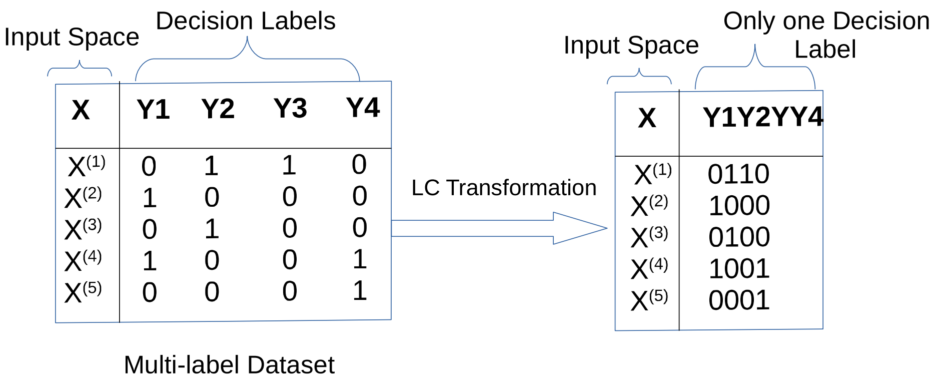


Fig. 10 Procedure of LC transformation method

The reason for choosing CC and LC methods is that they capture the label dependencies (correlation or co-occurrence) during classification to improve the classification performance (Guo and Gu 2011).

Single label classifiers: After the transformation, top 5 tree-based (single label) classifiers are used to predict multi-label methods (BR, CC, LC). The previous study (Azeem et al. 2019) shows that these classifiers achieved high performance in the code smell classification.

Validation procedure: To test the performance of different prediction models built, we applied 10-fold cross-validation and ran them up to 10 times to cope up with randomness (Hall et al. 2011). Figure 11 shows the procedure of 10-fold cross-validation. The process is executed 10 times ($10 \times 10 = 100$). For each iteration of cross-validation, a different randomized dataset is used, and to create each fold, a stratified sampling on the dataset is carried out.

Evaluation measures: The evaluation metric of MLC is different from that of single label classification since for each instance, there are multiple labels, which may be classified as partly correct or partly incorrect. MLC evaluation metrics are classified into two

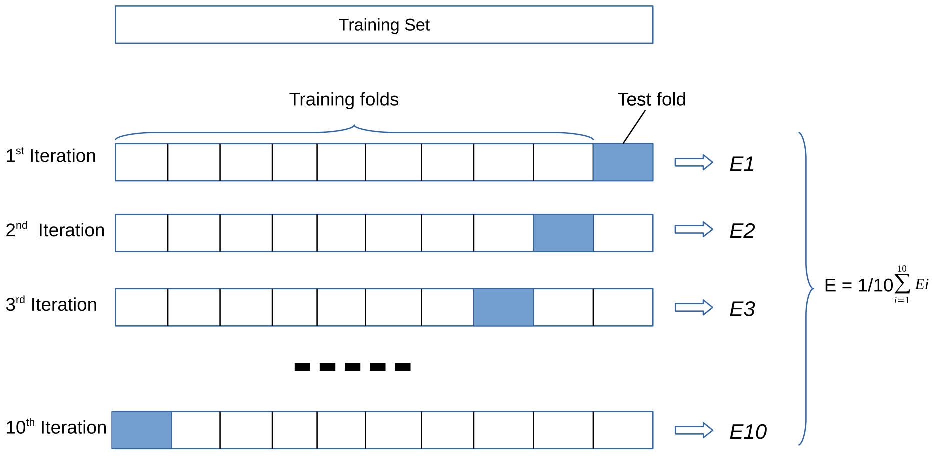


Fig. 11 Procedure of 10-fold cross-validation

groups: (1) example-based metrics and (2) label-based metrics. In the example-based metrics, each instance metric is calculated and the average of their metrics gives the final outcome. Label-based metrics are computed for each label instead of each instance. In this work, we have taken example-based measures. Label-based measures would fail to directly address the correlations among the different classes (Sorower 2010). Equations 1, 2, and 3 are used to measure the performances of MLC methods, which belong to the example-based metrics. In the given equations, D denotes the number of instances, L represents the number of labels, Y_i is the predicted labels for a given instance i, and Z_i indicates true labels for a given instance i. Detailed discussion of all other measures are defined in Sorower (2010).

– **Accuracy:** The proportion of correctly predicted labels with respect to the number of labels for each instance.

$$\text{Accuracy} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{1}$$

– **Hamming loss:** The prediction error (an incorrect label is predicted) and the missing error (a relevant label not predicted) normalized over a total number of classes and the total number of examples.

$$\text{Hammingloss} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \Delta Z_i|}{|L|} \tag{2}$$

– **Exact match ratio:** The predicted label set is identical to the actual label set. It is the most strict evaluation metric.

$$\text{Exactmatchratio} = \frac{1}{|D|} \sum_{i=1}^{|D|} I(Y_i = Z_i) \tag{3}$$

6 Experimental results

6.1 Dataset results

To answer the **RQ1**, we have considered the configured datasets of Di Nucci et al. (2018). The author merged the feature envy (FE) dataset into long method (LM) dataset and vice versa. The merged datasets are listed in Table 5. Each dataset has 840 instances, among them, 140 instances are smelly (affected) and 700 are non-smelly. While merging, there are 395 common instances among which 132 are smelly instances in the LM dataset. In the same way, when LM is merged with FE, there are 125 smelly instances in FE dataset. These

Table 5 Configured datasets

Number of instances	Method-level merged datasets			
	Long method		Feature envy	
	Smelly instances	Non-smelly instances	Smelly instance	Non-smelly instances
840	140	700	140	700

Table 6 Number of instances affected in multi-label dataset

Long method affected	Feature envy affected	Decision label in multi-label dataset	Number of instances affected	% of affected
Yes	Yes	11	100	22.47%
Yes	No	10	62	13.9%
No	Yes	01	40	8.9%
No	No	00	243	54.6%

132 and 125 instances suffer from disparity, i.e., the same instance is having two class labels (smelly and non-smelly). These disparity instances (Di Nucci et al. 2018) induce performance loss for standard single label ML classification techniques. The merged datasets are available at https://figshare.com/articles/Detecting_Code_Smells_using_Machine_Learning_Techniques_Are_We_There_Yet_/5786631

In our work, we create a multi-label dataset by merging 395 common and 50 uncommon (25 each) instances of LM and FE all put together; there are 445 instances. Table 6 shows the percentage and number of instances affected in the dataset. Out of 445, 100 instances are affected by both the smells. When concerned individually, there are 162 and 140 instances affected by LM and FE smell, respectively.

6.2 Multi-label dataset statistics

Table 7 lists the basic measures of multi-label training dataset characteristics. Some of the basic measures in single label dataset are attributes, instances, and labels. In addition to this, there are other measures added to the multi-label dataset (Tsoumakas and Katakis 2007). In the table, cardinality indicates the average number of active labels per instance. Dividing this measure by the number of labels in a dataset results in a dimensionless measure known as density. The two labels will have four label combinations (label sets) in our dataset. The mean imbalance ratio (mean IR) gives information about whether the dataset is imbalanced or not. Generally, Charte et al. (2015) in any multi-label dataset with a MeanIR value higher than 1.5 should be considered imbalanced. With this, the prepared multi-label dataset is well balanced because of the MeanIR value in our case is 1.07 which is less than the 1.5.

6.3 Co-occurrence analysis (market basket analysis)

To answer **RQ3:** In this section, we investigated the relation between two smells that can be done through the association (support, confidence) and correlation (lift) analysis.

Association analysis: How often the presence of long method implies the existence of feature envy and vice versa shown below. Support and confidence measures are used to evaluate the association among the smells.

Table 7 Statistics of multi-label dataset

Number of instances	Number of features	Number of labels	Number of label sets	Cardinality	Density	MeanIR
445	46	2	4	0.678	0.339	1.07

1. **Long method (LM) ⇒ Feature envy (FE):**

$$\text{Confidence} = \frac{\text{Support}(LM \cap FE)}{\text{Support}(LM)} = 100/162 = 61.7\%$$

2. **Feature envy (FE) ⇒ Long method (LM) :**

$$\text{Confidence} = \frac{\text{Support}(LM \cap FE)}{\text{Support}(FE)} = 100/140 = 71.4\%$$

In the above equations, support indicates how frequent a smell occurs together and confidence shows the number of times if-then patterns are formed.

Correlation analysis: Lift is a measure used to evaluate the correlation among the smells, i.e., how long method and feature envy are really related rather than coincidentally happening together.

$$\begin{aligned} Lift(LM \Rightarrow FE) &= \frac{\text{Support}(LM \cap FE)}{\text{Support}(LM) * \text{Support}(FE)} \\ &= \frac{(100/445)}{(162/445) * (140/445)} \\ &= \frac{0.22}{(0.36) * (0.31)} \\ &= 1.9 \end{aligned}$$

Lift (LM,FE) $>$ 1 (Sheikh et al. 2004) indicates that LM and FE are positively correlated, i.e., the occurrence of one implies the occurrence of another.

6.4 Performance improvement in existing datasets

To answer **RQ2**, we have removed 132 and 125 disparity instances of merged datasets of the long method (LM) and feature envy (FE), respectively. Now, the LM dataset has 708 instances among them 140 are positive (Smelly), and 568 are negative (non-smelly). In FE, the dataset has 715 instances among them 140 are positive, and 575 are negative. After that, we applied single label ML techniques (tree-based classifiers) on those datasets. The performance improved significantly on both the datasets which are shown in Tables 8 and 9. Earlier, the performance of the long method and feature envy datasets was on an average of 73% and 75% respectively using tree-based classifiers. After removing disparity instances in both the datasets, we got on an average of 95% and 98%, respectively. With this evidence, disparity in Di Nucci et al. (2018) had reduced the performance of the concerned code smell datasets.

Table 8 Long method results

Classifier	Accuracy	F-measure	ROC area
B-Random Forest	95.9%	96.0%	97.6%
Random Forest	*95.9%*	*96.0%*	*97.7%*
B-J48 UnPruned	95.4%	95.5%	97.1%
B-J48 Pruned	94.7%	94.8%	97.7%
J48 Unpruned	93.5%	93.5%	91.9%

Table 9 Feature envy results

Classifier	Accuracy	F-measure	ROC area
B-Random Forest	98.0%	98.0%	99.9%
Random Forest	98.1%	98.2%	99.9%
B-J48 UnPruned	99.0%	99.0%	98.7%
B-J48 Pruned	*99.1%*	*99.2%*	*99.3%*
J48 Unpruned	98.1%	98.2%	98.0%

Summary: Di Nucci et al. (2018), configured the datasets of Fontana et al. (2016b), to suit datasets for real use-case scenario's (reduced the metric distribution, added different type of smell instances) and applied ML classifiers on them. The authors achieved on an average 73%-75% accuracy for method-level datasets and claimed that, the results obtained by Fontana et al. (2016b) could not be generalized, thus contrasting the real effectiveness of ML code smell detection. But in this work, we have achieved 95%-98% of accuracy as similar as in Fontana et al. (2016b) after removal of disparity instances. From these results, we conclude that ML classifiers still gave good accuracy even for real use-case datasets.

6.5 Performances of multi-label classification

To answer **RQ4**: Three problem transformation methods (BR, CC, LC) are used to transform multi-label training dataset into a set of binary or multi-class datasets. Then the top 5 tree-based classification techniques are used on the transformed dataset. The performances of those techniques are shown in Tables 10, 11, and 12, respectively. To evaluate these techniques, we have run them for ten iterations using 10-fold cross-validation. We measure average accuracy, hamming loss, and an exact match of those 100 iterations. In addition to this, we also listed the label-based measures of BR, CC, and LC methods respectively in Appendix, Tables 13, 14, and 15. The results of all other measures are available to download at https://github.com/thiru578/Multilabel-Dataset/blob/master/MLD_Results.csv

Table 10 Results of BR method using top 5 single label classifiers

BR (10-fold cross-validation run for 10 iterations)

Single label classifier	Example-based metrics		
	Accuracy (Jaccard index)	Hamming loss	Exact match
J48 Pruned	*96.3%*	*0.028*	*94.4%*
Random Forest	93.7%	0.044	91.9%
B-J48 pruned	94.6%	0.037	92.6%
B-J48 UnPruned	95%	0.035	93.1%
B-Random Forest	93.6%	0.044	91.8%

Table 11 Results of CC method using top 5 single label classifiers

CC (10-fold cross-validation run for 10 iterations)			
Single label classifier	Example-based metrics		
	Accuracy (Jaccard index)	Hamming loss	Exact match
J48 Pruned	96.7%	0.024	95.3%
Random Forest	96.8%	0.022	95.6%
B-J48 pruned	96.5%	0.023	95.3%
B-J48 UnPruned	97%	0.022	95.5%
B-Random Forest	*97%*	*0.021*	*95.8%*

From Tables 10, 11, and 12, we report that all top 5 classifiers are performing well under the three methods in all evaluation measures. In the tables, we have highlighted the best classifier of each method in italic font. When we compare the three methods, CC and LC are performing better than the BR in all three evaluation measures. The reason behind these results is CC and LC methods considered the correlation during the classification, which was not done by the BR. When we compare the CC and LC method, LC is performing slightly better over the CC method. From these results, we observed that the correlation among the smells can improve the classifier performance in detecting multiple smells.

7 Threats to validity

In this section, we have discussed the threats that might affect our empirical studies and how to mitigate them.

Threats to internal validity In MLD construction, there are 395 common instances in the considered datasets. These instances can be strongly distinguished as smelly or non-smelly and any ML techniques can easily categorize the two classes. This kind of dataset is not suitable in real-time scenario and it is similar to the limitation of Fontana et al. (2016b) datasets which are shown by Di Nucci et al. (2018). To mitigate this issue, we have added the 50 uncommon instances with their predicted labels to the MLD. The prediction labels can be assigned with the help of classifier models of long method and feature envy (Fontana

Table 12 Results of LC method using top 5 single label classifiers

LC (10-fold cross-validation run for 10 iterations)			
Single label classifier	Example-based metrics		
	Accuracy (Jaccard index)	Hamming loss	Exact match
J48 Pruned	96.7%	0.02	96%
Random Forest	96.7%	0.024	95.4%
B-J48 pruned	*97.5%*	*0.016*	*96.9%*
B-J48 UnPruned	97.2%	0.018	96.4%
B-Random Forest	96.7%	0.024	95.4%

et al. 2016b). The prediction procedure is discussed in Section 4.2.1, in step 2. After adding the uncommon instances, now the MLD is suitable for real time due to the reduced metric distribution.

Threats to external validity With respect to the generalizability of the findings (external validity), we used two code smell datasets which are constructed from 74 open source Java projects for our experimentation. However, we can not declare that our results can be generalized to other coding languages, practitioners, and industrial practice. Future replications of this study are necessary to confirm the generalizability of our findings.

8 Conclusion and future directions

In this paper, two method-level code smells have been detected using multi-label classification approaches. The mechanisms proposed in the literature detect a single type of code smells for the same element (class or method). The main contributions of this paper can be summarized as follows:

– Addressed the disparity instances in the existing method-level datasets and improved the ML classifier performances from 73–75% to more than 95–98% by their removal. From these results, we conclude that ML classifiers still gave good accuracy even for real use-case scenario datasets. This work opens a new perspective for code smell detection called multi-label classification (MLC).
– For MLC, we have constructed a multi-label dataset (MLD) by considering method-level smell datasets from the single type detectors. We experimented with three multi-label classification methods (BR, CC, LC) on the constructed dataset. The main advantage of these methods (CC and LC) is that it takes the smell correlation (70% of confidence found in MLD) during the classification and produces higher accuracy than the BR method. The BR method does not consider the correlation and it classifies each decision label (code smell) independently by splitting the labels. With consideration of correlation, LC and CC method gave on an average of 96% and 95.5% of classifier performance for the exact match evaluation measure. Without consideration of the correlation, the BR method gave on an average of 92.7% classifier performance.

Though the proposed approach detected two code smells, it may not be limited to two. The methods of MLC also consider a negative correlation among the smells. For example, God class and data class smells can not co-occur (negative correlation) for the same class. In the future, we want to investigate negative correlation smells with the proposed approach . In addition to it, our findings have important implications for future research communities: (1) After the detection of the code smells, it was analyzed which smell has to be refactored first so as to reduce developer effort as different smell order requires varied effort. (2) Identify (or prioritize) the critical code elements for refactoring, based on the number of code smells an element can be detected, i.e., an element affected with more design problems (code smells) is considered the highest priority for refactoring. (3) In code smell detection, the number of code elements affected by the code smell in the software systems is reduced. For example, Palomba et al. (2018) have shown that the affect of God class code smell is less than 1% of the total number of classes in a system. This results in imbalanced dataset for ML classification. Thus, to handle those imbalanced datasets some of the multi-label classification methods like RAKEL (Tsoumakas et al. 2011) and PS (Read et al. 2008) are used. Apart from these methods, there are some other techniques such as SMOTE which is used in Charte et al. (2015) to handle the imbalance datasets. In our study, the multi-label dataset (MLD) is well balanced because we have considered only two code smells for the experimentation. As the code smells increases, the MLD gets imbalanced.

The datasets, which are free from disparity instances, are available at https://github.com/thiru578/Datasets-LM-FE.

The multi-label dataset is available for download at https://github.com/thiru578/Multilabel-Dataset.

Appendix

Table 13 Results of BR method using top 5 single label classifers

BR (10-fold cross-validation run for 10 iterations)						
Single label classifier	Label-based metrics					
	Micro averaging			Macro averaging		
	Micro precision	Micro recall	F1-Micro	Macro precision	Macro recall	F1-Macro
J48 Pruned	95.8%	95.9%	95.8%	95.8%	95.9%	95.6%
Random Forest	94.8%	93.1%	93.5%	94.7%	92.8%	93.1%
B-J48 pruned	95.8%	93.6%	94.5%	95.7%	93.3%	94%
B-J48 UnPruned	96.1%	94.1%	94.9%	96.2%	93.8%	94.4%
B-Random Forest	94.8%	93%	93.4%	94.6%	92.7%	93%

Table 14 Results of CC method using top 5 single label classifers

CC (10-fold cross-validation run for 10 iterations)						
Single label classifier	Label-based metrics					
	Micro averaging			Macro averaging		
	Micro precision	Micro recall	F1-Micro	Macro precision	Macro recall	F1-Macro
J48 Pruned	95.9%	97.1%	96.5%	95.8%	97.1%	96.3%
Random Forest	95.4%	97.9%	96.6%	95.2%	97.9%	96.5%
B-J48 pruned	96.2%	96.6%	96.4%	96.2%	96.6%	96.3%
B-J48 UnPruned	96%	97.5%	96.7%	95.8%	97.4%	96.5%
B-Random Forest	95.7%	97.9%	96.8%	95.6%	97.9%	96.6%

Table 15 Results of LC method using top 5 single label classifers

LC (10-fold cross-validation run for 10 iterations)

Single label classifier	Label-based metrics					
	Micro averaging			Macro averaging		
	Micro precision	Micro recall	F1-Micro	Macro precision	Macro recall	F1-Macro
J48 Pruned	96.9%	97.8%	97%	96.7%	96.9%	96.7%
Random Forest	95.1%	97.2%	96.4%	95%	97.8%	96.3%
B-J48 pruned	97.9%	97.5%	97.6%	97.9%	97.3%	97.6%
B-J48 UnPruned	97.8%	96.8%	97.3%	97.8%	96.6%	97.1%
B-Random Forest	95.3%	97.6%	96.4%	95.1%	97.6%	96.3%

References

Abdelmoez, W., Kosba, E., Iesa, A.F. (2014). Risk-based code smells detection tool. In *The international conference on computing technology and information management (ICCTIM2014)* (pp. 148–159): The Society of Digital Information and Wireless Communication.

Amorim, L., Costa, E., Antunes, N., Fonseca, B., Ribeiro, M. (2015). Experience report: evaluating the effectiveness of decision trees for detecting code smells. In *2015 IEEE 26th international symposium on software reliability engineering (ISSRE)* (pp. 261–269): IEEE.

Azeem, M.I., Palomba, F., Shi, L., Wang, Q. (2019). Machine learning techniques for code smell detectio: a systematic literature review and meta-analysis. Information and Software Technology.

Booch, G. (1980). *Object-oriented analysis and design*. Addison-Wesley.

Boutell, M.R., Luo, J., Shen, X., Brown, C.M. (2004). Learning multi-label scene classification. *Pattern Recognition*, *37*(9), 1757–1771.

Bowes, D., Randall, D., Hall, T. (2013). The inconsistent measurement of message chains. In *2013 4th International workshop on emerging trends in software metrics (WETSoM)* (pp. 62–68): IEEE.

Charte, F., Rivera, A.J., del Jesus, M.J., Herrera, F. (2015). Addressing imbalance in multilabel classification: measures and random resampling algorithms. *Neurocomputing*, *163*, 3–16.

Ciupke, O. (1999). Automatic detection of design problems in object-oriented reengineering. In *Technology of object-oriented languages and systems, 1999. TOOLS 30 Proceedings* (pp. 18–32): IEEE.

Di Nucci, D., Palomba, F., Tamburri, D.A., Serebrenik, A., De Lucia, A. (2018). Detecting code smells using machine learning techniques: are we there yet? In *2018 IEEE 25th International conference on software analysis, evolution and reengineering SANER* (pp. 612–621): IEEE.

Ferme, V. (2013). Jcodeodor: a software quality advisor through design flaws detection. Master's thesis University of Milano-Bicocca, Milano, Italy.

Fontana, F.A., & Zanoni, M. (2017). Code smell severity classification using machine learning techniques. *Knowledge-Based Systems*, *128*, 43–58.

Fontana, F.A., Braione, P., Zanoni, M. (2012). Automatic detection of bad smells in code: an experimental assessment. *Journal of Object Technology*, *11*(2), 5–1.

Fontana, F.A., Dietrich, J., Walter, B., Yamashita, A., Zanoni, M. (2016a). Antipattern and code smell false positives: preliminary conceptualization and classification. In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, (Vol. 1 pp. 609–613): IEEE.

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

Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (1999). Refactoring: improving the design of existing programs.

Godbole, S., & Sarawagi, S. (2004). Discriminative methods for multi-labeled classification. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 22–30): Springer.

Guo, Y., & Gu, S. (2011). Multi-label classification using conditional dependency networks. In *IJCAI Proceedings-international joint conference on artificial intelligence*, (Vol. 22 p. 1300).

Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S. (2011). Developing fault-prediction models: what the research can show industry. *IEEE Software*, *28*(6), 96–99.

Kessentini, W., Kessentini, M., Sahraoui, H., Bechikh, S., Ouni, A. (2014). A cooperative parallel search-based software engineering approach for code-smells detection. *IEEE Transactions on Software Engineering*, *40*(9), 841–861.

Khomh, F., Vaucher, S., Guéhéneuc, Y.G., Sahraoui, H. (2009). A Bayesian approach for the detection of code and design smells. In *9th International conference on quality software, 2009. QSIC'09* (pp. 305–314): IEEE.

Khomh, F., Vaucher, S., Guéhéneuc, Y.G., Sahraoui, H. (2011). Bdtex: a gqm-based Bayesian approach for the detection of antipatterns. *Journal of Systems and Software*, *84*(4), 559–572.

Kreimer, J. (2005). Adaptive detection of design flaws. *Electronic Notes in Theoretical Computer Science*, *141*(4), 117–136.

Liu, H., Guo, X., Shao, W. (2013). Monitor-based instant software refactoring. *IEEE Transactions on Software Engineering*, 1.

Maiga, A., Ali, N., Bhattacharya, N., Sabané, A., Guéhéneuc, Y.G., Antoniol, G., Aïmeur, E. (2012). Support vector machines for anti-pattern detection. In *2012 Proceedings of the 27th IEEE/ACM international conference on automated software engineering (ASE)* (pp. 278–281): IEEE.

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

Marinescu, R. (2002). Measurement and quality in objectoriented design. *IEEE International Conference on Software Maintenance*.

Marinescu, R. (2004). Detection strategies: metrics-based rules for detecting design flaws. In *20th IEEE International conference on software maintenance, 2004. Proceedings* (pp. 350–359): IEEE.

Marinescu, R. (2005). Measurement and quality in object-oriented design. In *Proceedings of the 21st IEEE international conference on software maintenance, 2005. ICSM'05* (pp. 701–704): IEEE.

Moha, N., Gueheneuc, Y.G., Duchien, A.F., et al. (2010a). Decor: a method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering (TSE)*, *36*(1), 20–36.

Moha, N., Guéhéneuc, Y.G., Le Meur, A.F., Duchien, L., Tiberghien, A. (2010b). From a domain analysis to the specification and detection of code and design smells. *Formal Aspects of Computing*, *22*(3-4), 345–361.

Murphy-Hill, E., & Black, A.P. (2010). An interactive ambient visualization for code smells. In *Proceedings of the 5th international symposium on software visualization* (pp. 5–14): ACM.

Nongpong, K. (2012). Integrating "code smells" detection with refactoring tool support. Thesis, University of Wisconsin-Milwaukee.

Opdyke, W.F. (1992). *Refactoring: a program restructuring aid in designing object-oriented application frameworks PhD thesis*. PhD thesis: University of Illinois at Urbana-Champaign.

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., Poshyvanyk, D. (2013). Detecting bad smells in source code using change history information. In *Proceedings of the 28th IEEE/ACM international conference on automated software engineering* (pp. 268–278): IEEE Press.

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., Poshyvanyk, D., De Lucia, A. (2015). Mining version histories for detecting code smells. *IEEE Transactions on Software Engineering*, *41*(5), 462–489.

Palomba, F., Oliveto, R., De Lucia, A. (2017). Investigating code smell co-occurrences using association rule learning: a replicated study. In *IEEE Workshop on machine learning techniques for software quality evaluation (MaLTeSQuE)* (pp. 8–13): IEEE.

Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., De Lucia, A. (2018). On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering*, *23*(3), 1188–1221.

Pecorelli, F., Di Nucci, D., De Roover, C., De Lucia, A. (2019a). On the role of data balancing for machine learning-based code smell detection. In *Proceedings of the 3rd ACM SIGSOFT international workshop on machine learning techniques for software quality evaluation* (pp. 19–24): ACM.

Pecorelli, F., Palomba, F., Di Nucci, D., De Lucia, A. (2019b). Comparing heuristic and machine learning approaches for metric-based code smell detection. In *Proceedings of the 27th international conference on program comprehension* (pp. 93–104): IEEE Press.

Rao, A.A., & Reddy, K.N. (2007). Detecting bad smells in object oriented design using design change propagation probability matrix 1.

Rasool, G., & Arshad, Z. (2015). A review of code smell mining techniques. *Journal of Software: Evolution and Process*, *27*(11), 867–895.

Read, J., Pfahringer, B., Holmes, G. (2008). Multi-label classification using ensembles of pruned sets. In *2008 Eighth IEEE international conference on data mining* (pp. 995–1000): IEEE.

Read, J., Pfahringer, B., Holmes, G., Frank, E. (2011). Classifier chains for multi-label classification. *Machine Learning*, *85*(3), 333.

Read, J., Reutemann, P., Pfahringer, B., Holmes, G. (2016). Meka: a multi-label/multi-target extension to weka. *The Journal of Machine Learning Research*, *17*(1), 667–671.

Sheikh, L.M., Tanveer, B., Hamdani, M. (2004). Interesting measures for mining association rules. In *8th International multitopic conference, 2004. Proceedings of INMIC 2004* (pp. 641–644): IEEE.

Sorower, M.S. (2010). A literature survey on algorithms for multi-label learning. Oregon State University, Corvallis, p. 18.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., Noble, J. (2010). The qualitas corpus: a curated collection of java code for empirical studies. In *Software engineering conference (APSEC), 2010 17th Asia Pacific* (pp. 336–345): IEEE.

Travassos, G., Shull, F., Fredericks, M., Basili, V.R. (1999). Detecting defects in object-oriented designs: using reading techniques to increase software quality. In *ACM sigplan notices*, (Vol. 34 pp. 47–56): ACM.

Tsantalis, N., & Chatzigeorgiou, A. (2009). Identification of move method refactoring opportunities. *IEEE Transactions on Software Engineering*, *35*(3), 347–367.

Tsoumakas, G., & Katakis, I. (2007). Multi-label classification: an overview. *International Journal of Data Warehousing and Mining (IJDWM)*, *3*(3), 1–13.

Tsoumakas, G., Katakis, I., Vlahavas, I. (2011). Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, *23*(7), 1079–1089.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., Poshyvanyk, D. (2017). When and why your code starts to smell bad (and whether the smells go away). *IEEE Transactions on Software Engineering*, *43*(11), 1063–1088.

Wang, X., Dang, Y., Zhang, L., Zhang, D., Lan, E., Mei, H. (2012). Can i clone this piece of code here? In *Proceedings of the 27th IEEE/ACM international conference on automated software engineering* (pp. 170–179): ACM.

White, M., Tufano, M., Vendome, C., Poshyvanyk, D. (2016). Deep learning code fragments for code clone detection. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering* (pp. 87–98): ACM.

Yang, J., Hotta, K., Higo, Y., Igaki, H., Kusumoto, S. (2015). Classification model for code clones based on machine learning. *Empirical Software Engineering*, *20*(4), 1095–1125.

Zaidi, M.A., & Colomo-Palacios, R. (2019). Code smells enabled by artificial intelligence: a systematic mapping. In *International conference on computational science and its applications* (pp. 418–427): Springer.

Zhang, M.-L., & Zhou, Z.-H. (2013). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *26*(8), 1819–1837.

Thirupathi Guggulothu is working as a Research Scholar in the area of Software Engineering, University of Hyderabad and done his post graduation in the stream of Computer Science, University of Hyderabad. He has worked as a Research Associate on the project titled "implementation of A5/1 attack" sponsored by DLRL in the University of Hyderabad. His research interests include software evolution, software maintenance, and machine learning. He qualified UGC NET and TS SET in 2017.

Salman Abdul Moiz is working as an Professor in School of Computer & Information Sciences at the University of Hyderabad. He worked as Professor and Head CSE at GITAM University, Hyderabad Campus. He has previously worked as Research Scientist at Centre for Development of Advanced Computing, Bangalore. He is a member of IEEE, ACM, IE, and EWB. His research interests include software engineering, software re-usability, mobile databases, and e-Learning.