Comparison of Multi-Label Classification Algorithms for Code Smell Detection

Elife Ozturk Kiyak
The Graduate School of Natural and Applied Sciences
Dokuz Eylul University
Izmir, Turkey
elife.ozturk@ceng.deu.edu.tr

Derya Birant
Department of Computer Engineering
Dokuz Eylul University
Izmir, Turkey
derya@cs.deu.edu.tr

Kokten Ulas Birant
Department of Computer Engineering
Dokuz Eylul University
Izmir, Turkey
ulas@cs.deu.edu.tr

***Abstract*— Code smells in a source code shows the weakness of design or implementation. To detect code smells, several detection tools have been developed. However, these tools generally produce different results, since code smells are subjectively interpreted, informally defined and configured by the developers, domain-dependent and based on opinions and experiences. To cope with these issues, in this paper, we have used machine learning techniques, especially multi-label classification methods, to classify whether the given source code is affected with more than one code smells or not. We have conducted experiments on four code smell datasets and transformed them into two multi-label datasets (one for method level and the other one for class level). Two multi-label classification methods (Classifier Chains and Label Combination) and their ensemble models performed on the converted datasets using five different base classifiers. The results show that, as a base classifier, Random Forest algorithm performs better than Decision Tree, Naive Bayes, Support Vector Machine and Neural Network algorithms.**

Keywords*— *code smell detection, multi-label classification, software engineering, machine learning

I. Introduction

After deciding a software project is ready to release, it is essential for keeping the maintainability of the product for software developers. If the source code of the software has high complexity, it becomes difficult to modify it. The presence of a code smell indicates that the source code has poor design or implementation. For instance, the source code may have some duplicated methods, many lines of code, long parameter lists and unused codes. If there exists code smell in a source code, this means that a part of code needs to be refactoring which is one of the solutions to remove smells.

Detection of code smells in a source code is a significant research topic in software engineering. Although several studies [1] and tools [2] exist in the literature to detect code smells, there are some challenges to evaluate them. For instance, detection techniques produce different results since code smells are subjectively interpreted by the developers and also agreement between smell detector tools is low [2,3]. To overcome these problems, in this paper, we have used machine learning (ML) techniques to detect the presence of code smells.

In literature [4], some ML studies can only identify a single type of code smell in the source code. Furthermore, some studies can classify the code as smelly or non-smelly only. However, given code fragment may be affected from multiple smells. In other words, the source code can contain more than one code smell such as both large methods and dead codes. For this reason, in this study, we performed multi-label classification techniques on code smell datasets.

The main contributions of this study are two-fold: (i) it applies multi-label classification (MLC) technique for detecting both method-level and class-level code smells; (ii) it compares two MLC methods (Classifier Chains and Label Combination) and their ensemble models (BaggingML and EnsembleML) based on five classification algorithms: Decision Tree, Random Forest, Naive Bayes (NB), Support Vector Machine (SVM) and Neural Network (NN).

The remainder of the paper is organized as follows. Section 2 gives an overview of previous works related to multi-label classification and code smell detection. Section 3 explains methods and algorithms used in this study. Section 4 describes dataset preparation process. Section 5 presents experimental results. Section 6 provides concluding remarks and future directions.

II. Related Work

Over the last few years, multi-label classification has attracted interest in many areas such as text classification [5], image classification [6]. MLC is also encountered in the field of software engineering. For instance, Xia et al. [7] focused on crash reporting in a software system. A crash report may indicate that a failure is caused by more than one type of fault at the same time. Manually analyzing of these failures is time consuming. Hence, the authors proposed an algorithm named MLL-GA that combines different algorithms. Feng et al. [8] compared the multi-label and single-label classification on failure reports of software systems.

Many different code smell approaches exist in the literature. However, we concentrate on machine learning based approaches to detect code smells. Kreimer [9] proposed a method for identifying design errors and to make predictions with decision trees. Maneerat and Muenchaisri [10] collected datasets for evaluating 7 code smells and apply 7 machine learning algorithms for detection using 27 design model metrics extracted by a tool. In addition, Fontana and Zanoni [11] classified the code smells severity by using a ML method which can help software developers to grade the classes or methods. They used multinomial classification and regression methods.

In the aforementioned studies, code smell detection considered through single label. Guggulothu and Moiz [12] have used multi-label classification methods in this area. However, our study differs from it in three aspects: (i) we evaluated both method-level and class-level code smells, (ii) we also performed different ensemble-based models, and (iii) we used different ML techniques as base classifier such as NB, SVM, NN, instead of only tree-based techniques.

III. Methodology

A. Machine Learning Based Code Smell Detection

An overview of code smell detection utilizing machine learning is presented in Fig. 1. This process is based on two essential parts: dataset generation and machine learning application. In order to generate dataset, code smell definitions are specified by the developers and source codes in the system are analyzed to extract code metrics. Instances are labeled with manual validation or code smell advisors, or both. After that, several machine learning algorithms are performed and compared with each other to select the best.

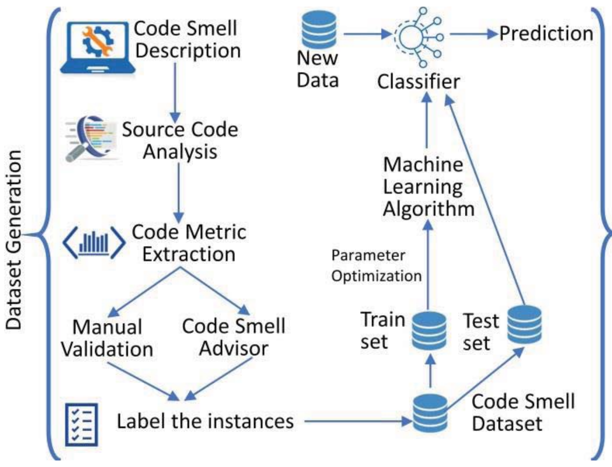


Fig. 1. An overview of code smells detection process.

B. Multi-Label Classification

Supervised classification algorithms predict the labels of new instances by learning pre-labeled instances. Classification problems can be categorized into three main groups according to output type: binary, multi-class, and multi-label. In *binary classification*, each example belongs to one of two classes (i.e. yes or no). In *multi-class classification*, there are more than two classes, but each instance can only be assigned one of them. However, in *multi-label classification* (MLC) each instance can belong to more than one class. Hence, MLC provides to learn from instances that are associated with a set of target labels.

In a code smell detection study, binary classification can be used to predict the existence of code smells (presence or absence). However, in multi-label classification, a source code element can be assigned to multiple smell types as shown in Fig. 2.

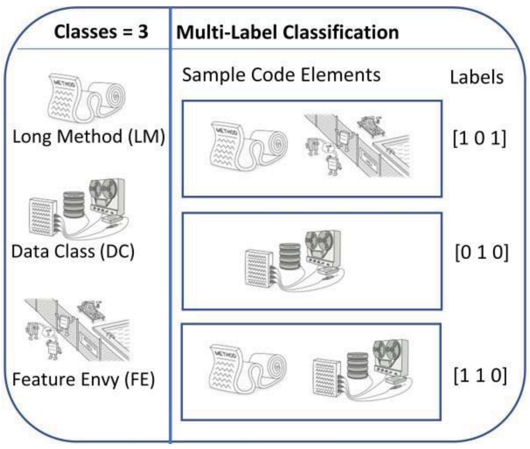


Fig. 2. An example of multi-label code smells detection.

C. Multi-label Classification Methods

Basically, there are two approaches that are widely used to solve MLC problems, namely: *algorithm adoption methods* (AAM) and *problem transformation methods* (PTM).

AAM involves modifying classifiers to make them multi-label capable. For instance, MLkNN is the multi-label version of k-nearest neighbors algorithm.

PTM converts multi-label problems to several single-label problems. After that, those single-label problems are solved using base classifiers. There are two main techniques that fall under PTM category:

1) *Binary relevance (BR) method:* It firstly converts a multi-label dataset to many binary datasets as the number of classes. After that, a standard binary classification algorithm is applied on them separately. The predictions from binary classifiers are joined to get the final outcome.

2) *Label Powerset (LP) method*: It firstly converts a multi-label dataset to multi-class dataset based on all unique label combinations. After that, any multi-class classification algorithm is applied to train.

In this study, Classifier Chains (CC) under BR category and Label Combination (LC) under LP category were applied on code smell datasets.

Classifier Chains (CC) [13]: It creates an order on *L*, a chain of labels, and trains a binary classifier for each label in this order. The difference between BR and CC is feature space: while BR considers original set, CC takes into account interdependencies between class labels.

Label Combination (LC): It converts the multi-label problem into a single-label problem. Each label set is considered as atomic label in a single-label problem. Hence, the single-class label set illustrates all distinct label subsets in the original multi-label presentation.

D. Ensemble of Multi-label Classifiers

In this study, two ensemble learning techniques [14] were also employed to investigate their effects on multi-label classification of code smells.

1) *BaggingML* combines several multi-label classifiers in a voting system using Bootstrap AGGregatING (Bagging).

2) *EnsembleML* combines several multi-label classifiers by taking a subset of training set for each model (i.e. sampling without replacement).

E. Base Classifiers

In this study, firstly a multi-label classification problem is converted into single-label classification ones through a problem transformation strategy, after that a traditional classification algorithm is used as a base classifier to build models. Base learners used in this study are briefly described as follows.

C4.5 decision tree algorithm creates a tree to classify an instance of data. To select a splitting attribute, the algorithm utilizes a criterion such as information gain or Gini index.

Random Forest (RF) is an ensemble learning method and aims to increase the classification accuracy by generating multiple decision trees. Each tree is created by extracting samples from original dataset and evaluating different subsets of features at each node. Final decision is given by voting the predictions of all trees.

Naive Bayes (NB) is an algorithm based on Bayes Theorem which describes an equation that gives the probability of an event, based on prior knowledge of conditions. When classifying, it takes into account features independently of one another. The aim is to maximize the probability of the target class.

Support Vector Machine (SVM) is a supervised learning algorithm based on vector space. The goal is to construct an optimal hyperplane in a multidimensional space that maximizes the margin between different classes.

A *Neural Network* (NN) is an interconnected assembly of simple processing units that is made up of different layers such as input layer, hidden layer(s) and output layer. When the NN is trained, the weights of the interconnections are modified in order to decrease the error (difference between output and the corresponding input).

IV. Dataset

A. Dataset Description

In this study, two method-level datasets (long method and feature envy) and two class-level datasets (god class and data class) were used [15]. To construct these datasets, Fontana and Zanoni [16] used code smell detectors and metric extraction tools. They analyzed 74 open source java projects in the Qualitas Corpus. While method-level datasets have 82 features (software metrics), class-level datasets consist of 61 features. The datasets contain four types of code smells among the most frequent ones:

- Method-Level Datasets:
  1) *Long Method* (LM): It refers to method that has too many lines and requires too many parameters. Hence, it tends to be complex and difficult to understand.
  2) *Feature Envy* (FE): It refers to method that uses more data from other classes than its own class.
- Class-Level Datasets:
  3) *God Class* (GC): It refers to a class that serves as a central of the system and has many members and responsibilities.
  4) *Data Class* (DC): It refers to class that is used only for storing data, without providing any other functionality. This type of class only contains fields (data) and get/set methods for accessing them.

B. Multi-label Dataset Preparation

Data preparation process (merging single-labeled datasets to obtain a multi-labeled dataset) is illustrated in Fig. 3. Each dataset consists of many instances (I_1, I_2, I_3 … I_m), features (F_1, F_2, F_3 … F_n) and a class label LM, FE, DC or GC. Method-level datasets (LM and FE) were merged with each other and class level datasets (GC and DC) were combined together to construct two separate multi-label datasets (MLD). While merging, common elements on both datasets were added to MLD with their corresponding two class labels and the remaining elements were added into MLD by considering the other class label as non-smelly.

Method-level MLD includes 250 non-smelly and 190 smelly (LM, FE, or both) instances. Class-level MLD has also approximately similar number of instances and when concerned separately there are 140 instances affected by DC and GC.

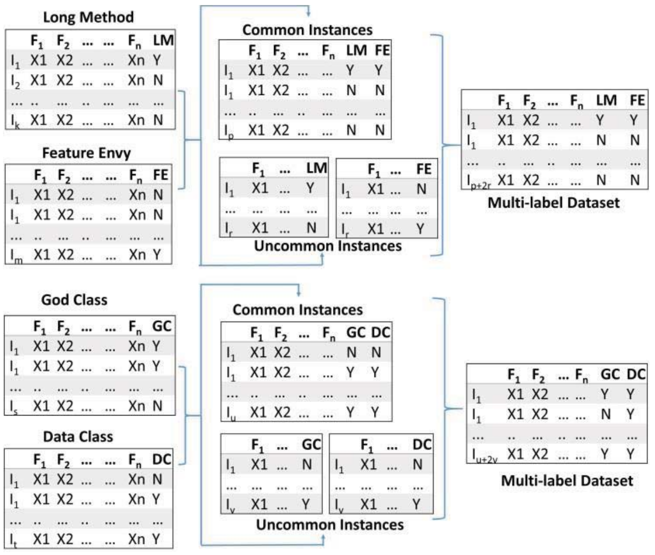


Fig. 3. Multi-labeled dataset preparation process.

V. Experimental Studies

In this study, experiments were conducted on MEKA software package. We used default parameters of the algorithms in each experiment. To test the performances of algorithms, we applied 10-fold cross validation technique. Experiments were performed by using 2.7 GHz quad core processor and 8GB RAM.

A. Evaluation Metrics

MLC evaluation metrics can be classified into two fundamental groups: *example based* and *label based* [12]. In our work, we used example based measures since label based measures would fail to address the correlations among different classes [12].

To measure the performances of MLC approaches, we used five metrics: accuracy, F1, AUROC, hamming score and exact match ratio. Assume that dataset D includes m multi-label instances denoted by (x_i, Y_i) where $1 \le i \le m$, and $x_i \in X$, $Y_i \in Y = \{0, 1\}^k$, with a label set L. Let h be a multi-label classifier and $Z_i = h(x_i) = \{0, 1\}^k$ be the set of label memberships predicted by h for the example x_i.

1) *Accuracy:* The ratio of correctly predicted labels by the number of labels for each instance.

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

2) *F1*: The harmonic mean of the precision and recall.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2}$$

3) *AUROC*: The area under the ROC curve.

4) *Hamming Score*: The accuracy for each label to correctly predicted, normalized over the number of labels and instances.

$$Hamming\ Score = 1 - \frac{1}{m}\sum_{i=1}^{m}\frac{|Y_i \, \Delta \, Z_i|}{|L|} \quad (3)$$

5) Exact Match Ratio: The percentage of the predicted label set that is identical to the actual label set,

$$Exact\ Match\ Ratio = \frac{1}{m}\sum_{i=1}^{m} I(Y_i = Z_i) \quad (4)$$

where I is the indicator function.

B. *Experimental Results*

We have conducted experiments on two multi-label code smell datasets (method-level and class-level). Two multi-label classification methods (CC and LC), and their ensemble models, BaggingML (BML) and EnsembleML (EML), performed on the datasets using five different base classifiers: C4.5, RF, NB, SVM and NN.

While Table I and Table II shows the performances of those techniques on method-level dataset, Table III and Table IV shows the results for class-level dataset. In all cases, the best results were obtained by the RF algorithm (93.6% accuracy for method-level and 96.9% for class-level). Hence, the algorithm achieves more successful results for class-level, compared with method-level. However, according to the experimental results, it is possible to say that ensemble models (BML and EML) don't affect the accuracy so much. The results also indicate that the CC method generally performs slight over the LC method.

TABLE I. PERFORMANCES OF CLASSIFIER CHAINS METHOD AND ITS ENSEMBLE MODELS ON METHOD-LEVEL DATASET

	Binary Relevance (Method-Level Code Smells Dataset)														
Base Classifiers	**Accuracy (%) (Jaccard Index)**			**F1**			**AUROC**			**Hamming Score (%)**			**Exact Match (%)**		
	CC	**BML_{CC}**	**EML_{CC}**	**CC**	**BML_{CC}**	**EML_{CC}**	**CC**	**BML_{CC}**	**EML_{CC}**	**CC**	**BML_{CC}**	**EML_{CC}**	**CC**	**BML_{CC}**	**EML_{CC}**
C4.5	89.6	**92.8**	92.2	0.875	**0.924**	0.917	0.910	0.969	**0.975**	92.2	**94.9**	94.5	85.4	**90.3**	89.2
RF	**93.6**	**92.8**	**92.9**	**0.928**	0.923	**0.919**	**0.954**	**0.970**	0.970	**95.3**	**94.9**	**94.7**	**91.0**	**90.3**	**89.9**
NB	79.0	80.0	79.9	0.758	0.771	0.765	0.823	0.884	0.892	84.9	85.3	85.1	73.7	74.4	74.6
SVM	89.3	89.8	89.7	0.886	0.891	0.888	0.919	0.959	0.947	92.7	93.0	92.8	85.8	86.5	86.5
NN	89.0	90.1	89.4	0.879	0.893	0.884	0.912	0.959	0.964	92.4	93.1	92.6	85.4	87.0	85.6

TABLE II. PERFORMANCES OF LABEL COMBINATION METHOD AND ITS ENSEMBLE MODELS ON METHOD-LEVEL DATASET

	Label Powerset (Method-Level Code Smells Dataset)														
Base Classifiers	**Accuracy (%)**			**F1**			**AUROC**			**Hamming Score (%)**			**Exact Match (%)**		
	LC	**BML_{LC}**	**EML_{LC}**	**LC**	**BML_{LC}**	**EML_{LC}**	**LC**	**BML_{LC}**	**EML_{LC}**	**LC**	**BML_{LC}**	**EML_{LC}**	**LC**	**BML_{LC}**	**EML_{LC}**
C4.5	89.0	92.5	92.5	0.880	0.916	0.917	0.912	0.966	0.967	92.5	94.4	94.5	85.2	89.4	89.4
RF	**93.3**	**92.7**	**93.3**	**0.920**	**0.919**	**0.921**	**0.950**	**0.967**	**0.970**	**94.7**	**94.7**	**94.8**	**90.1**	**89.9**	**90.3**
NB	83.3	83.8	84.5	0.784	0.806	0.805	0.839	0.921	0.925	86.9	87.4	87.8	77.1	77.8	78.4
SVM	87.5	87.0	87.6	0.861	0.859	0.863	0.899	0.954	0.949	91.2	91.1	91.3	83.6	83.1	83.8
NN	87.5	87.8	88.0	0.858	0.865	0.866	0.896	0.955	0.954	91.1	91.5	91.5	83.1	83.6	84.0

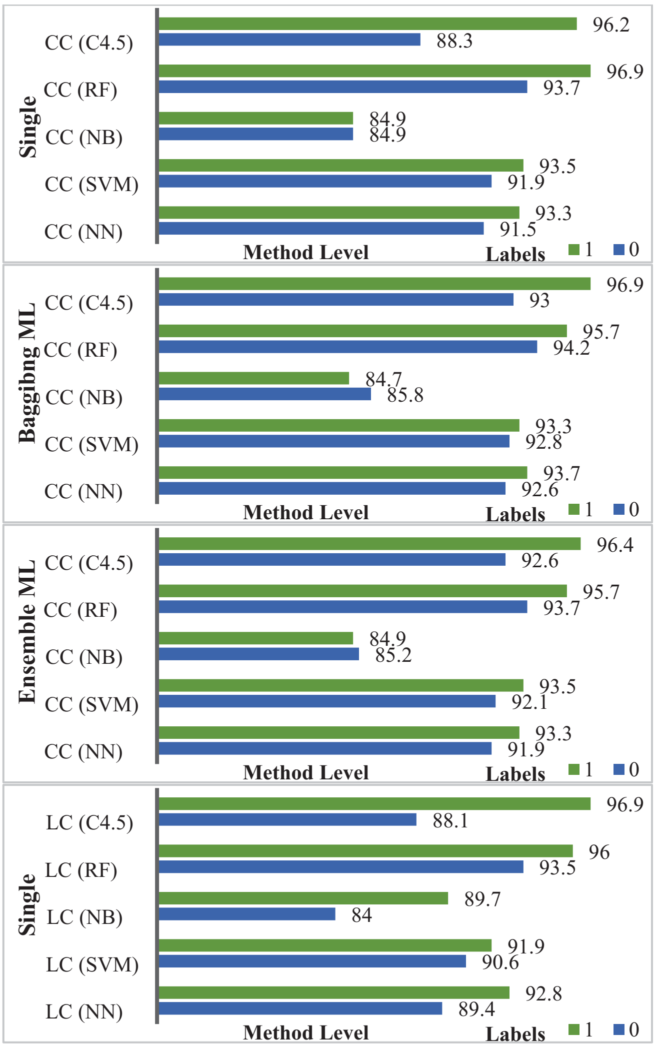
TABLE III. PERFORMANCES OF CLASSIFIER CHAINS METHOD AND ITS ENSEMBLE MODELS ON CLASS-LEVEL DATASET

	Binary Relevance (Class-Level Code Smells Dataset)														
Base Classifiers	**Accuracy (%)**			**F1**			**AUROC**			**Hamming Score (%)**			**Exact Match**		
	CC	**BML_{CC}**	**EML_{CC}**	**CC**	**BML_{CC}**	**EML_{CC}**	**CC**	**BML_{CC}**	**EML_{CC}**	**CC**	**BML_{CC}**	**EML_{CC}**	**CC**	**BML_{CC}**	**EML_{CC}**
C4.5	94.6	**96.2**	95.5	0.958	**0.970**	0.965	0.974	**0.993**	**0.991**	97.3	**98.1**	97.8	94.6	**96.2**	95.5
RF	**96.9**	**96.2**	**96.2**	**0.975**	**0.970**	**0.970**	**0.982**	0.991	0.989	**98.4**	**98.1**	**98.1**	**96.9**	**96.2**	**96.2**
NB	72.1	74.9	71.7	0.808	0.822	0.806	0.878	0.942	0.933	85.2	86.7	85.0	71.8	74.5	71.4
SVM	91.9	91.9	91.9	0.935	0.935	0.935	0.950	0.971	0.958	96.0	96.0	96.0	91.9	91.9	91.9
NN	90.8	91.5	91.9	0.923	0.928	0.930	0.949	0.986	0.983	95.1	95.4	95.5	90.6	91.3	91.7

TABLE IV. PERFORMANCES OF LABEL COMBINATION METHOD AND ITS ENSEMBLE MODELS ON CLASS-LEVEL DATASET

	Label Powerset (Class-Level Code Smells Dataset)														
Base Classifiers	**Accuracy (%)**			**F1**			**AUROC**			**Hamming Score (%)**			**Exact Match (%)**		
	LC	**BML$_{LC}$**	**EML$_{LC}$**	**LC**	**BML$_{LC}$**	**EML$_{LC}$**	**LC**	**BML$_{LC}$**	**EML$_{LC}$**	**LC**	**BML$_{LC}$**	**EML$_{LC}$**	**LC**	**BML$_{LC}$**	**EML$_{LC}$**
C4.5	95.1	95.7	96.2	0.961	0.966	0.968	0.972	**0.992**	**0.990**	97.5	97.9	98.0	95.1	95.7	96.2
RF	**96.0**	**96.6**	**96.4**	**0.968**	**0.974**	**0.972**	**0.979**	0.989	**0.990**	**98.0**	**98.3**	**98.2**	**96.0**	**96.6**	**96.4**
NB	88.4	88.0	87.7	0.902	0.898	0.893	0.936	0.969	0.959	93.6	93.4	93.1	88.4	87.9	87.7
SVM	91.7	91.9	91.9	0.932	0.936	0.932	0.949	0.976	0.964	95.7	96.0	95.7	91.7	91.9	91.9
NN	90.4	92.6	92.2	0.922	0.937	0.936	0.947	0.984	0.978	95.1	96.0	95.9	90.4	92.6	92.2

Fig. 4 and Fig. 5 show model accuracies (%) per label for method-level and class-level datasets respectively. According to the results, it is possible to say that the constructed models generally make more accurate predictions for label 1. The graphs also show that the algorithms achieve higher classification accuracies (for each label) for class-level code smell dataset, compared with method-level one.



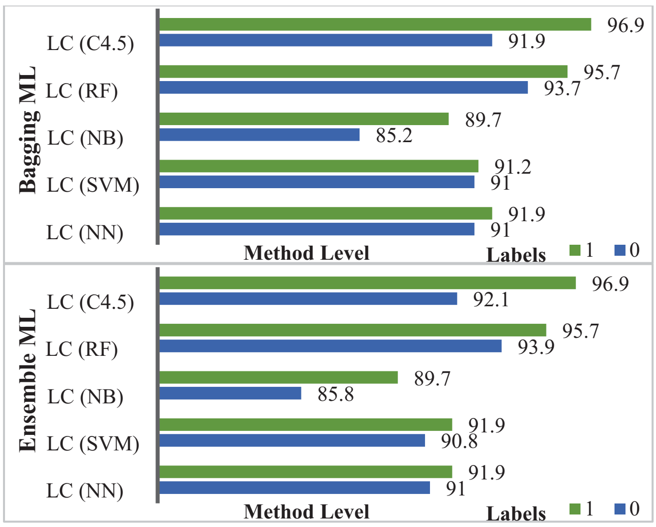
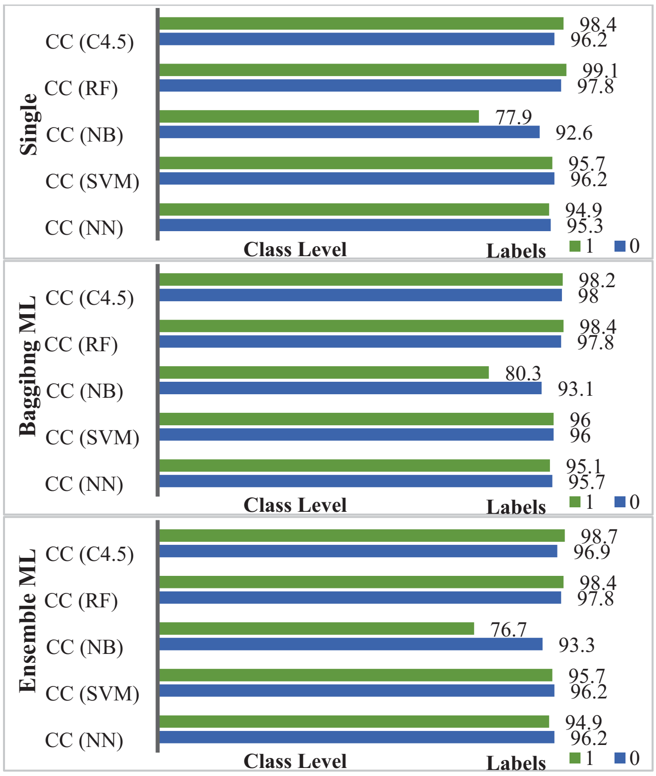


Fig. 4. Model accuracies per label for method-level code smells.



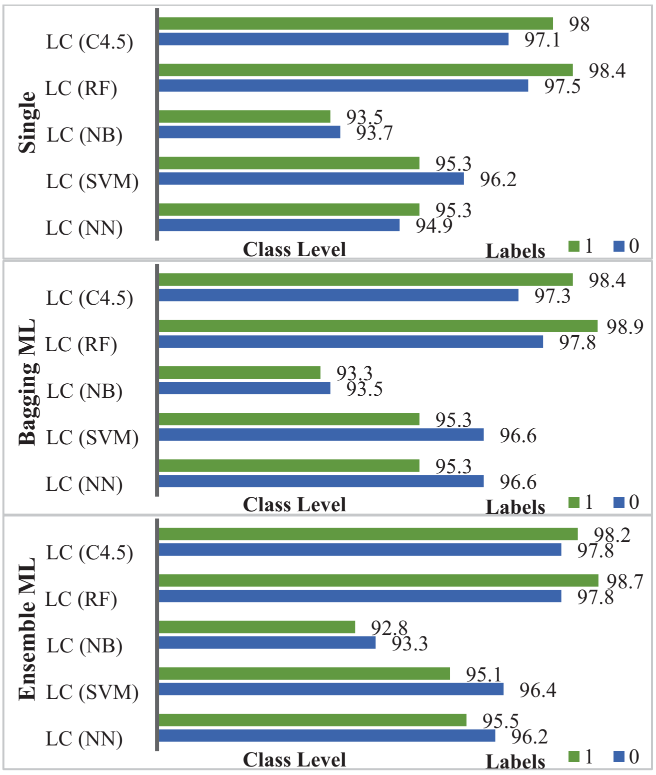


Fig. 5. Model accuracies per label for class-level code smells.

Build times of the classifiers are presented in Table V and Table VI for method-level and class-level datasets respectively. According to the results, NN had a very long build time. Ensemble classifiers (BaggingML and EnsembleML) need more execution time than single-models, since they construct more than one model.

TABLE V. BUILD TIME OF CLASSIFIERS FOR METHOD-LEVEL CODE SMELLS DATASET

	Execution Times (sec.)					
Base Classifiers	**Binary Relevance (Method-Level Dataset)**			**Label Powerset (Method-Level Dataset)**		
	CC	**BML_{CC}**	**EML_{CC}**	**LC**	**BML_{LC}**	**EML_{LC}**
C4.5	0.058	0.125	0.098	0.016	0.081	0.076
RF	0.212	0.856	0.942	0.103	0.586	0.672
NB	0.014	0.058	0.045	0.005	0.02	0.019
SVM	0.087	0.223	0.211	0.031	0.234	0.256
NN	21.213	137.238	146.995	11.325	72.486	76.241

TABLE VI. BUILD TIME OF CLASSIFIERS FOR CLASS-LEVEL CODE SMELLS DATASET

	Execution Times (sec.)					
Base Classifiers	**Binary Relevance (Class Level)**			**Label Powerset (Class Level)**		
	CC	**BML_{CC}**	**EML_{CC}**	**LC**	**BML_{LC}**	**EML_{LC}**
C4.5	0.009	0.056	0.059	0.008	0.044	0.047
RF	0.092	0.565	0.628	0.064	0.404	0.454
NB	0.005	0.033	0.033	0.002	0.017	0.016
SVM	0.031	0.103	0.111	0.014	0.126	0.104
NN	12.53	80.589	83.931	6.519	44.579	44.918

VI. CONCLUSION AND FUTURE WORK

In a software project, a source code may have more than one type of code smell. Hence, instead of single-label classification, multi-label classification (MLC) is considered in this study. We applied MLC technique for detecting both method-level and class-level code smells. We compared two MLC methods (Classifier Chains and Label Combination) and their ensemble models (BaggingML and EnsembleML) based on five classification algorithms: Decision Tree, Random Forest, Naive Bayes, Support Vector Machine and Neural Network. The results show that, as a base classifier, the RF algorithm performs better than others.

As a future work, multi-label classification algorithms can be applied on other software product or process data, i.e. to predict multiple software defect types.

REFERENCES

[1] W. Kessentini, M. Kessentini, H. Sahraoui, S. Bechikh, and A. Ouni, “A cooperative parallel search-based software engineering approach for code-smells detection”, IEEE Transactions on Software Engineering, vol. 40, issue 9, 2014, pp. 841-861.

[2] F. A. Fontana, P. Braione, and M. Zanoni, “Automatic detection of bad smells in code: An experimental assessment”, Journal of Object Technology, vol. 11, issue 2, 2012, pp. 1-38.

[3] T. Paiva, A. Damasceno, E. Figueiredo, and C. Sant’Anna, “On the evaluation of code smells and detection tools”, Journal of Software Engineering Research and Development, vol. 5, issue 1, 2017.

[4] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, “Machine learning techniques for code smell detection: A systematic literature review and meta-analysis”, Information and Software Technology, vol. 108, April 2019, pp. 115-138.

[5] W. C. Chang, H. F. Yu, K. Zhong, Y. Yang, and I. Dhillon, “A modular deep learning approach for extreme multi-label text classification” arXiv preprint arXiv:1905.02331, (2019).

[6] C. Li, C. Liu, L. Duan, P. Gao, K. Zheng, “Reconstruction regularized deep metric learning for multi-label image classification”, IEEE transactions on neural networks and learning systems, in press, 2019.

[7] X. Xia, Y. Feng, D. Lo, Z. Chen, and X. Wang, “Towards more accurate multi-label software behavior learning”, In 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, 2014, pp. 134-143.

[8] Y. Feng, J. Jones, Z. Chen, and C. Fang, “An empirical study on software failure Ccassification with multi-label and problem-transformation techniques”, In 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), 2018, pp. 320-330.

[9] J. Kreimer, “Adaptive detection of design flaws”, Electronic Notes in Theoretical Computer Science, vol. 141, issue 4, 2005, pp. 117-136.

[10] N. Maneerat, P. Muenchaisri, “Bad-smell prediction from software design model using machine learning techniques”, Computer Science and Software Engineering (JCSSE), 2011 Eighth International Joint Conference on, IEEE, 2011, pp. 331-336.

[11] F. A. Fontana, and M. Zanoni, “Code smell severity classification using machine learning techniques”, Knowledge Based Systems, vol. 128, 2017, pp. 43-58.

[12] T. Guggulothu and S.A. Moiz, “Code smell detection using multilabel classification approach” arXiv preprint arXiv:1902.03222 (2019).

[13] R. Senge, J. J. del Coz, E. Hüllermeier, “Rectifying classifier chains for multi-label classification”, arXiv preprint arXiv:1906.02915 (2019).

[14] A. G. de Sá, A. A. Freitas, G. L. Pappa, “Multi-label classification search space in the MEKA software” arXiv preprint arXiv:1811.11353 (2018).

[15] Code smell detection datasets, http://www.essere.disco.unimib.it/, Accessed on September 2019.

[16] F. A. Fontana, M. V. Mantyla, M. Zanoni, A. Marino, “Comparing and experimenting machine learning techniques for code smell detection”, Empirical Software Engineering, vol. 21, no 3, 2016.