Exploration of the Feasibility and Applicability of Domain Adaptation in Machine Learning-Based Code Smell Detection

Peeradon Sukkasem and Chitsutha Soomlek(✉)

Department of Computer Science, College of Computing, Khon Kaen University, Khon Kaen, Thailand
peeradon_s@kkumail.com, chitsutha@kku.ac.th

Abstract. Machine learning-based code smell detection was introduced to mitigate the limitations of the heuristic-based approach and the subjectivity issues. Due to limited choices of the publicly available datasets, most of the machine learning-based classifiers were trained by the earlier versions of open-source projects that no longer represent the characteristics and properties of modern programming languages. Our experiments exhibit the feasibility and applicability of using a machine learning classifier well-trained on the earlier versions of open-source projects to classify four types of code smells, i.e., *god class*, *data class*, *feature envy*, and *long method*, in modern Java open-source projects without extensive feature engineering. The performance produced by the supervised machine learning algorithms was evaluated and compared. Particle swarm optimization and Bayesian optimization were adopted to enhance the performance of the machine learning classifiers, i.e., decision tree and random forest. The experimental results indicated that the machine learning-based code smell classifiers adapt poorly to the different target domain. The hyper-parameter optimization slightly improves the performance of the machine learning classifiers when classifying *feature envy*, *god class*, and *long method* in a modern Java project.

Keywords: Code smells · Machine learning · Hyper-parameter optimization

1 Introduction

A code smell indicates poor design or implementation choices made during software development [13]. Software maintenance and software evolution are inevitable. Software maintenance and software evolution ensure customer satisfaction and support new features due to advancement of technology. The software evolution activities unwittingly introduce code smells into a system [34]. Code smells cause unfavorable issues in a software system, particularly software maintainability and software quality. Although code smells do not affect the

C. Anutariya and M. M. Bonsangue (Eds.): DSAI 2023, CCIS 1942, pp. 74–87, 2023.
https://doi.org/10.1007/978-981-99-7969-1_6

functionality and outputs of a program, code smells can lead to future software failure and increase technical debt and maintenance cost [14,18,29]. To identify and eliminate a code smell, code refactoring is a well established solution. Code refactoring improves the design and quality of the code [13]. To support code refactoring, an effective code smell detection technique is required.

For decades, automated code analysis tools and researches often utilize the metrics-based approaches [2]. The metric-based approach suffers from subjectivity issues and is error prone due to the lack of common definitions of code smells. Programming languages are improved over time, and some code smells are language-specific. These problems cause the metric-based code smell detection fails to find the most suitable set of metrics and threshold values [15,28]. Finding metrics and calibrating the thresholds are exhaustive.

Alternatively, a machine learning-based approach trains the classifiers using software properties or attributes. Machine learning mitigates the subjectivity issues and provides reliable code smell detection results when a machine learning classifier is trained on high-quality data.

In recent years, machine learning-based code smell detection has been progressively studied [2,16]. The performance produced by a machine learning algorithm heavily relies on the quality of the training dataset.

It takes considerable time and effort to create a high-quality code smell dataset that is labeled by experienced software developers and experts. The widely used code smell datasets are of the `Qualitas Corpus` benchmark [37]. Many well-studied code smell datasets were built upon the `Qualitas Corpus` benchmark [21,41]. Although Di Nucci et al. [10] commented that the threats to the validity of their work are inherited from the threats related to the creation of the `Qualitas Corpus`, other studies report that the machine learning classifiers trained by those datasets achieve high performance results [1,9,25]. Considering the age of the `Qualitas Corpus` project and the modern programming paradigm, it is questionable that a machine learning-based code smell detection well-trained on the code smell dataset built upon the `Qualitas Corpus` benchmark would produce a comparable performance when identifying a code smell in a modern software project. To explore the feasibility and applicability of domain adaptation in machine learning-based code smell detection, we present the experiments with two goals:

(1) To evaluate whether the machine learning-based code smell classifiers trained over a dataset built upon the `Qualitas Corpus` benchmark can effectively detect code smells in a modern software project.
(2) To investigate whether hyper-parameter optimization (HPO) can improve the performance of the machine learning classifiers in the same settings as (1).

To these aims, we adopt the open-source Java projects from the `Qualitas Corpus` dataset and the labels from Fontana et al. [11,12] in our training process. A wide range of code metrics are extracted from the Java projects. The machine learning algorithms are trained to classify four types of code smells, i.e., *god class*, *data class*, *feature envy*, and *long method*. `MLCQ` dataset [23], which is

the most up-to-date industry-relevant code smell dataset at the time of writing, is used as the testing data. The performance results are evaluated and compared. Three HPO techniques of particle swarm optimization and Bayesian optimization are applied to improve the performance of the machine learning classifiers. The experimental results indicate that the classifiers produce poor performance results when the domain is shifted. HPO techniques hardly improve the recall scores of the classifiers. There is a slight difference between the baseline and optimized version in a large picture.

2 Related Work

Research on code smells is advancing and focusing on machine learning-based code smell detection [2] and using code smells as prediction factors [1,7,10,12, 21,25]. Recent research shows that machine learning-based code smell detection mitigates the subjectivity issues and produces promising results [35]. Considering that the machine learning algorithms should be trained on a reliable dataset labeled by experienced software developers, up-to-date and reproducible code smell datasets are limited [21]. Collecting a large number of software systems with the information needed and labeling code smells in the target source code is a laborious task. Zakeri-Nasrabadi et al. [41] did a survey on 45 code smell datasets and revealed that most of the existing datasets cover limited types of code smells. The datasets in the survey are imbalanced, lack of supporting severity levels, and are restricted to Java programming language. We argue that code smell datasets of other programming languages exist, e.g., [38,40,42], but there are smaller choices compared to Java. The work of Zakeri-Nasrabadi et al. [41] also confirms that the datasets proposed by Palomba et al. [26] and Madeyski et al. [23] are the most comprehensive dataset. The datasets from Fontana et al. [11,12] are mostly cited in research work. Fontana's datasets are also built from the Qualitas Corpus benchmark. Besides its popularity, the Qualitas Corpus benchmark is outdated. The projects in the dataset are primarily developed by Java 5. Fontana's datasets have been commented on several aspects such as the size, types of code smells, and the ratio between smelly and non-smelly samples which is not realistic compared to the nature of the code smell problem [10].

In 2020, Mandelski et al. introduced the `MLCQ` dataset [23], which contains code smells from the newer versions of open-source Java projects. It is larger in size, is labeled by experienced software developers, and is more comprehensive. Recent research also adopted the `MLCQ` dataset and achieved promising results, e.g., [20,24,35,39]. Each of them spent time on extensive metrics extraction and feature engineering. Therefore, domain adaptation in machine learning-based code smell detection would mitigate the issues. The closest work to ours is [31] proposed by Sharma et al. Unlike our research, their work focuses on training deep learning models from C# code samples and evaluating the models over Java code samples and vice-versa. Our research aims to train the machine learning classifiers on one programming language and seeks the possibility for the model trained on the previous version of the programming language to work on the newer version.

Another possibility to enhance a machine learning classifier is to apply HPO. The closest work to this research and to our previous work [36] is presented by Shen et al. [32]. The researchers trained the machine learning-based code smell classifiers on Fontana et al.'s datasets [11,12] and applied HPO to improve the performance. Four optimizers were applied to six machine learning classifiers to identify two types of code smells, i.e., *data class* and *feature envy*. They did not focus on domain shifts in the data distribution in their research. This research is a continuation of our previous work [36], in which we already employed one particle swarm optimizer and two Bayesian optimizers to the machine learning-based code smell classifiers. The three optimizers were applied to the machine learning classifiers to enhance the performance. The rest of this research article is brand new.

3 Research Methodology

3.1 Data Collection

The main goals of this research are to explore whether the machine learning-based code smell classifiers trained on a dataset built upon the `Qualitas Corpus` benchmark can effectively detect code smells in a modern software project and to improve the performance of the classifiers through HPO. The experiments require three data sources: software projects in the `Qualitas Corpus` benchmark, smell and non-smelly samples in the `Qualitas Corpus` benchmark, and smell and non-smelly samples in the projects built on the newer versions of Java.

The Qualitas Corpus Benchmark. The dataset was contributed by Tempero et al. in 2010 [37] and the latest version (release 20130901r) contains 112 open-source Java software systems. Later in 2013, 74 systems were selected to create a code smell dataset by Fontana et al. [12]. The `Qualitas Corpus` benchmark contains a total of 6,785,568 lines of code, 3,429 packages, 51,826 classes, and 404,316 methods. Fontana et al. also provide smell and non-smelly samples of *god class*, *data class*, *feature envy*, and *long method* in the dataset.

In this research, a large set of code metrics were extracted by using code analysis tools, i.e., Design Features and Metrics for Java (DFMC4J) [1] and Understand tool by SciTools [30]. Code metrics exhibit software properties in various perspectives and can be used as independent variables to train machine learning models. The dependent variable is the label, which indicates whether a class or a method is a code smell or not. The label of a code sample is obtained from Fontana et al.'s code smell dataset. The major challenge is to match a code sample to its label and the corresponding code metrics due to the different file path formats used. There are 140 positive (smelly) instances and 280 negative (non-smelly) instances for each type of code smells in our training dataset.

The MLCQ Dataset. Madeyski and Lewowski [23] contribute an industry-relevant code smell dataset. The MLCQ dataset contains nearly 15,000 code samples together with severity levels (i.e., *critical*, *major*, *minor*, and *none*). These code samples were collected from Java open-source projects available on GitHub within the year 2019. 26 professional software developers reviewed and classified the code samples into four classes, i.e., *data class*, *blob* (a.k.a. *god class*), *feature envy*, and *long method*. A severity level is also given to a reviewed code sample. The *none* severity level indicates a non-smelly or negative sample. The MLCQ dataset contains 1057, 974, 454, and 806 positive instances for *data class*, *god class*, *feature envy*, and *long method*, respectively. For the negative instances, there are 2964, 3045, 2883, and 2556, respectively. This dataset is considered as the most comprehensive available code smell dataset in the field [41].

3.2 Data Preparation

When an instance in Fontana et al. contains a null code metric value, the instance is removed. In the case of the MLCQ dataset, several procedures were performed. Some Java projects are no longer available on the GitHub repository. In total, 518 projects from the MLCQ dataset were analyzed. Information relative to the reviewers, for instance, code sample id, reviewer id, and reviewing timestamp were also excluded from the dataset. Since there are multiple reviewers labeled on an instance, the majority vote was used to determine the positive and negative instances. Then, the Understand tool by SciTools [30] was employed to extract the code metric values. Due to the granularity difference between the code analysis tool and the dataset, there are instances with a null metric value. In this case, the instances were removed.

To test machine learning on unseen data, the machine learning model must be trained over a training set having identical properties as the testing set. We carefully studied the definitions of the code metrics provided by Fontana et al. and the Understand tool. Only identical code metrics were selected.

3.3 Experiments

Fontana et al.'s code smell dataset was used as a training set. The machine learning algorithms were tested over the MLCQ dataset. The experiments evaluated the performance metrics, e.g., precision, recall, and f-score. For each experiment, 10-fold cross validation was implemented with 10 repetitions. The experiments were repeated for 10 iterations to avoid randomness and then the results were averaged to conclude the experiments.

Selecting Machine Learning Algorithms. As mentioned earlier, this study is a continuation of our previous work [36]. Our previous work and other research in the same area confirm that decision tree [6] and random forest [5] produce the best performance in the context of code smell detection [35]. Both machine learning algorithms not only produce high performance results but also provide

the benefits of interpretability. Although we can include other machine learning algorithms, we chose to study decision tree and random forest to seek the feasibility and applicability of domain adaptation in machine learning-based code smell detection. The experiments can be repeated on other machine learning algorithms to explore more. Note that both machine learning algorithms were implemented by using Scikit-learn [27].

Selecting Hyper-parameter Optimization Techniques. This research selected hyper-parameter optimization (HPO) techniques based on the characteristics of the hyper-parameters of the machine learning algorithms. Considering the decision tree and random forest algorithm, the majority of the hyper-parameters are discrete. One of them is categorical. Decision tree and random forest possess the characteristic of a large hyper-parameter search space with multiple hyper-parameter types. From this perspective, Bayesian optimization using random forest (SMAC), Bayesian optimization using tree-structured parzen estimator (BO-TPE), and particle swarm optimization (PSO) are well-matched to the machine learning algorithms. The selected hyper-parameter and their configuration space are given in Table 1, which is inherited from our previous work [36]. Note that all HPO techniques use the same hyper-parameter configuration space and consider recall score as an objective function with 10 maximum optimization iterations.

Table 1. List of selected hyper-parameters with their characteristics and configuration space.

Hyper-parameter	Characteristic	Range
criterion	Categorical	'gini', 'entropy'
max_depth	Discrete	[5, 50]
min_samples_split	Discrete	[2, 11]
min_samples_leaf	Discrete	[1, 11]
max_features	Discrete	[1, 64]
n_estimators	Discrete	[100, 300]

Bayesian optimization (BO) [33] is an iterative algorithm learning from previously found information. The algorithm consists of two main components: a probabilistic surrogate model and an acquisition function. BO works by building a surrogate model of the objective function, detecting the optimal on the surrogate model, and applying the optimal configuration to the real objective function. The surrogate model is updated with the new results and these procedures are repeated until the optimal solution is found, or the maximum number of iterations is reached. BO can use tree-parzen estimator (TPE) [3] and random forest (SMAC) [17] to support multiple types of hyper-parameters. Therefore, SMAC and BO-TYPE are included in this research.

Particle swarm optimization (PSO) [19] is an optimization technique inspired by biological theories. The process of PSO starts by randomly initializing a group of samples called particles. These particles explore the search space to search for the optimal point. Then, the obtained information is shared within a group. PSO algorithm is one of the most popularly used algorithms in metaheuristic research due to its ease of implementation, flexibility, and high performance. PSO also supports multiple types of hyper-parameters.

Several well-recognized Python libraries were employed to implement HPO. Hyperopt [4] (v.0.2.7) was used to implement BO-TPE. SMAC3 library [22] (v.1.4.0), which is the only Python library providing BO with random forest surrogate model at the time of writing, was adopted to implement SMAC. For PSO, Optunity [8] (v.1.1.1) was used to implement the algorithm.

4 Results and Discussion

Table 2 shows the ratio of instances used for training and testing per code smell. These two datasets have two major differences. First, Fontana et al.'s code smell dataset contains 420 instances, which is four times smaller than the `MLCQ` dataset for all types of code smells. Second, the positive and negative ratio of the testing set is heavily imbalanced. Obviously, for *feature envy*, there are only 3% of smelly instances in the testing set, while the training set contains more than 30%. For the rest code smells, the testing set contains approximately 20% less number of smelly instances.

Table 2. No. of training and testing instances, No. of negative and positive instance and the ratio of negative and positive per code smell.

Code smell	Dataset type	Dataset	No. of instance	Negative ratio	Positive ratio
Data class	training set	Fontana et al	420	0.67	0.33
	testing set	MLCQ	2154	0.87	0.13
Feature envy	training set	Fontana et al	420	0.67	0.33
	testing set	MLCQ	2035	0.97	0.03
God class	training set	Fontana et al	420	0.67	0.33
	testing set	MLCQ	2122	0.89	0.11
Long method	training set	Fontana et al	420	0.67	0.33
	testing set	MLCQ	2080	0.88	0.12

In the experiments, the performance evaluation results obtained from the validation set and the testing set are compared in three aspects, i.e., precision, recall, and f-score. When a classifier achieves a relatively close or higher prediction score than a validation score, the machine learning classifier is general enough to identify code smells on a software project developed by a newer version of Java. In other words, it is feasible and applicable for the machine learning-based code smell detection to adapt when there is a change in the data

distribution. Otherwise, a new dataset containing the characteristics of modern Java programming language is required. Other strategies used to gain a better model adaptation are also needed. Figures 1–3 illustrates the evaluation results.

In terms of recall, in most cases, the performance of the machine learning classifiers was extremely decreased, except for *feature envy* as shown in Fig. 1. The recall produced by decision tree and random forest were dropped by $\approx 30\%$ and $\approx 60\%$ when classifying *data class* and *god class*, respectively. In the case of *long method*, both decision tree and random forest hardly predicted correct results on the target domain. Unlike *feature envy*, the decision tree classifier achieved slightly lower recall ($\approx$ 5%) on the testing set. The random forest classifier produced $\approx 16\%$ lower recall over the testing set than the validation set.

Fig. 1. Recall produced by decision tree (left) and random forest (right)

Figure 2 shows the precision results for code smell detection. It is clear that in most cases the classifiers achieved lower precision when facing the new target domain, except for *long method*. In this case, both algorithms achieved relatively close prediction and validation precision scores. Considering the machine learning algorithm, random forest classifiers outperformed decision tree in all cases. The random forest also achieved the highest precision at 100% when detecting *long method* in the `MLCQ` dataset.

In the case of f-score (see Fig. 3), domain shifting in code smell datasets makes a substantial difference in the validation and prediction scores. When facing a different target domain, the random forest slightly outperformed the decision tree and achieved the best f-score at 42.7% for *god class* classification.

Our experiments confirm that the machine learning-based code smell classifiers trained on a dataset built upon the `Qualitas Corpus` benchmark adapt poorly to domain shifting. The classifiers achieved lower performance results when they were trying to detect code smells in a modern software project. To improve the performance, HPO algorithms were applied to the machine learning models. The experimental results are summarized in Table 3–5.

In most of the cases, the optimized random forest classifiers produced higher recalls than their default settings. Table 3 indicates that random forest with PSO can achieve a better recall by $\approx 5\%$ when classifying *feature envy*. For *long method*, PSO assists both algorithms to achieve slightly higher recalls. The

random forest classifier applying SMAC can also achieve a slightly better recall when identifying *god class*.

Considering the precision (see Table 4), decision tree with SMAC produced higher precision results when classifying *god class* and *feature envy*. In case of the optimized random forest classifier, SMAC is the only algorithm that can assist the random forest classifier to achieve a slightly better result when classifying *feature envy*.

Table 5 compares the f-score obtained from the classifiers with the default configurations the proposed HPO techniques. SMAC can boost the performance by $\approx 1\%$ and $\approx 2\%$ for *god class* and *feature envy*, respectively. PSO can slightly improve both decision tree and random forest when classifying *long method*. However, none of the classifiers can exceed 45% of the f-score.

Table 3. Recall produced by the classifiers with default configuration and the proposed HPO techniques.

Code smell	Decision tree				Random forest			
	Default	BO-TPE	SMAC	PSO	Default	BO-TPE	SMAC	PSO
Data Class	0.51021	**0.51831**	0.46761	0.50106	**0.52359**	0.48486	0.50915	0.49225
Feature Envy	**0.57188**	0.55625	0.54375	0.53906	0.52188	0.56094	0.55313	**0.57188**
God Class	**0.33562**	0.33176	0.28884	0.30043	0.32275	0.33691	**0.33777**	0.33605
Long Method	0.13008	0.13008	0.13008	**0.13252**	0.13618	0.13049	0.13374	**0.13699**

We can conclude that the machine learning-based code smell classifiers trained over a dataset built upon the `Qualitas Corpus` benchmark can not produce desirable results when identifying code smells in a modern software project. The advancement of programming languages leads to changes in software properties and various aspects of software, which affect the performance of the classifiers. The experimental results confirm that the machine learning-based code smell classifiers adapt poorly to domain shifting. Although the hyper-parameter optimization techniques can slightly improve the prediction results in some cases, it is insufficient for the machine learning-based code smell classifiers

Fig. 2. Precision produced by decision tree (left) and random forest (right)

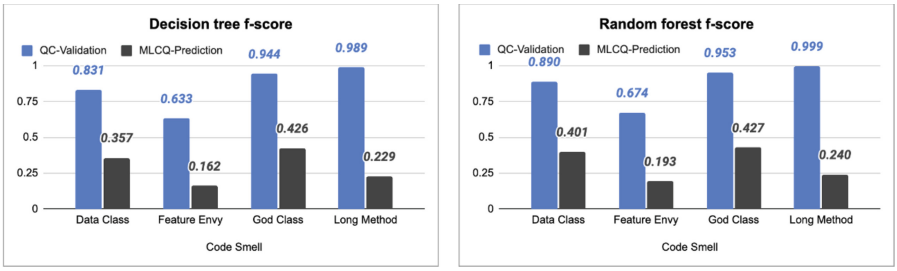

Fig. 3. F-score produced by decision tree (left) and random forest (right)

Table 4. Precision produced by the classifiers with default configuration and the proposed HPO techniques.

Code smell	Decision tree				Random forest			
	Default	BO-TPE	SMAC	PSO	Default	BO-TPE	SMAC	PSO
Data Class	**0.27473**	0.26738	0.25785	0.26127	**0.32547**	0.29867	0.30598	0.29844
Feature Envy	0.09473	0.10510	**0.10873**	0.10813	0.11839	0.11704	**0.11954**	0.11581
God Class	0.58377	0.60510	**0.62915**	0.62012	**0.63201**	0.61174	0.61159	0.61148
Long Method	**0.96970**	0.96970	0.96970	0.96541	**1.00000**	0.99108	1.00000	0.97702

Table 5. F-score produced by the classifiers with default configuration and the proposed HPO techniques.

Code smell	Decision tree				Random forest			
	Default	BO-TPE	SMAC	PSO	Default	BO-TPE	SMAC	PSO
Data Class	**0.35702**	0.35082	0.32947	0.34188	**0.40136**	0.36941	0.38173	0.37137
Feature Envy	0.16246	0.17634	**0.18072**	0.17950	0.19298	0.19349	**0.19647**	0.19239
God Class	**0.42616**	0.42559	0.39214	0.40391	0.42726	0.43438	**0.43516**	0.43364
Long Method	0.22939	0.22939	0.22939	**0.23293**	0.23969	0.23059	0.23586	**0.24014**

to work on a different target domain. There is no significant difference in the performance results produced by the default-configured and optimized classifiers. The concluded experimental results already grant the answers to both of the main goals of this research.

Threats to Validity. Note that there are subjectivity issues in both code metrics and code smells. Although we selected the same set of metrics from both the Fontana et al.'s code smell dataset and the `MLCQ` dataset, there are insignificant differences in the metric definitions due to different code analysis tools used. The selected code metrics might not represent the characteristics of the source code and the code smells. Moreover, the code smell instances of the two datasets were labeled by disparate groups of software developers. The Fontana et al.'s code smell dataset was labelled by master students whose background and experience are inconclusive. In case of the `MLCQ` dataset, the instances were labelled

by a group of experienced professional software developers. The researchers also include the background experience and survey results of the developers.

5 Conclusions

This research explores the feasibility and applicability of domain adaptation in machine learning-based code smell detection. The experiments ran supervised machine learning classifiers well-trained on the earlier versions of open-source Java projects to identify code smells in the modern Java open-source projects without carrying out extensive feature engineering. Three hyper-parameter optimization techniques, i.e., Bayesian optimization using random forest (SMAC), Bayesian optimization using tree-structured parzen estimator (BO-TPE), and particle swarm optimization (PSO), were also applied to the machine learning-based code smell classifiers to improve their performance. The experiments measured precision, recall, and f-score.

Fontana et al.'s code smell dataset, which is a dataset built upon the `Qualitas Corpus` benchmark, was employed as a training set. The `MLCQ` dataset, which is a dataset built upon the newer-version Java projects, was used as a test set. Most of the open-source Java projects in the `Qualitas Corpus` benchmark are of Java 5, which is much older than Java versions used in the projects included in the `MLCQ` dataset.

The experimental results indicate that the machine learning classifiers trained over Fontana et al.'s code smell dataset incompetently detect four types of code smells in the `MLCQ` dataset. In other words, the classifiers adapt poorly to domain shifting in code smell detection problems. Although the hyper-parameter optimization techniques were adopted, the performance results of the enhanced machine learning classifiers are smell-specific. For example, random forest with PSO produced a higher recall and f-score when classifying *long method*. Decision tree with SMAC achieved higher precision results when classifying *god class* and *feature envy*.

While the interest in using machine learning-based techniques for code smell detection is inclined, creating more choices of high quality training datasets is a non-trivial and exhaustive task. Programming languages are also evolved overtime. It would be impractical to continue laboriously working on code smell dataset construction to keep up with the advancement of programming languages. This research opens a new opportunity for using a more sophisticated domain adaption technique to mitigate the limitations and supports the research community to mature more studies in the fields. To support the advancement in the research community, we made the dataset and source code publicly available at https://github.com/Peeradon06/Domain-Adaptation-ML-Based-Code-Smell-Detection.

Acknowledgements. This research was partially supported by the Department of Computer Science, College of Computing and the Graduate School, Khon Kaen University, Khon Kaen, Thailand.

References

1. Arcelli Fontana, F., Mäntylä, M.V., Zanoni, M., Marino, A.: Comparing and experimenting machine learning techniques for code smell detection. Empir. Softw. Eng. **21**(3), 1143–1191 (2015). https://doi.org/10.1007/s10664-015-9378-4
2. Azeem, M.I., Palomba, F., Shi, L., Wang, Q.: Machine learning techniques for code smell detection: a systematic literature review and meta-analysis. Inf. Softw. Technol. **108**, 115–138 (2019). https://doi.org/10.1016/j.infsof.2018.12.009
3. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Advances in Neural Information Processing Systems, vol. 24. Curran Associates, Inc. (2011)
4. Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., Cox, D.D.: Hyperopt: a python library for model selection and hyperparameter optimization. Comput. Sci. Discov. **8**(1), 014008 (2015). https://doi.org/10.1088/1749-4699/8/1/014008
5. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324
6. Breiman, L.: Classification and Regression Trees. Routledge, New York (2017). https://doi.org/10.1201/9781315139470
7. Caram, F.L., Rodrigues, B.R.D.O., Campanelli, A.S., Parreiras, F.S.: Machine learning techniques for code smells detection: a systematic mapping study. Int. J. Software Eng. Knowl. Eng. **29**, 285–316 (2019). https://doi.org/10.1142/S021819401950013X
8. Claesen, M., Simm, J., Popovic, D., Moreau, Y., De Moor, B.: Easy hyperparameter search using optunity. http://arxiv.org/abs/1412.1114. Accessed 15 Jan 2023
9. Dewangan, S., Rao, R.S., Mishra, A., Gupta, M.: A novel approach for code smell detection: an empirical study. IEEE Access **9**, 162869–162883 (2021)
10. Di Nucci, D., Palomba, F., Tamburri, D.A., Serebrenik, A., De Lucia, A.: 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 (2018). https://doi.org/10.1109/SANER.2018.8330266
11. Fontana, F.A., Zanoni, M.: Code smell severity classification using machine learning techniques. Knowl.-Based Syst. **128**, 43–58 (2017)
12. Fontana, F.A., Zanoni, M., Marino, A., Mantyla, M.V.: Code smell detection: towards a machine learning-based approach. In: 2013 IEEE International Conference on Software Maintenance, pp. 396–399. IEEE (2013). https://doi.org/10.1109/ICSM.2013.56, http://ieeexplore.ieee.org/document/6676916/
13. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston, MA, USA (1999)
14. Hall, T., Zhang, M., Bowes, D., Sun, Y.: Some code smells have a significant but small effect on faults. ACM Trans. Softw. Eng. Methodol. **23**(4), 1–39 (2014). https://doi.org/10.1145/2629648
15. Haque, M.S., Carver, J., Atkison, T.: Causes, impacts, and detection approaches of code smell: a survey. In: Proceedings of the ACMSE 2018 Conference, pp. 1–8. ACM, Richmond Kentucky (2018). https://doi.org/10.1145/3190645.3190697
16. Hasantha, C.: A systematic review of code smell detection approaches. J. Adv. Softw. Eng. Test. **4**(1), 1–9 (2021). https://doi.org/10.5281/zenodo.4738772
17. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) Learning and Intelligent

Optimization, pp. 507–523. Springer, Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40
18. Kaur, A.: A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes. Arch. Comput. Methods Eng. **27**(4), 1267–1296 (2020). https://doi.org/10.1007/s11831-019-09348-6
19. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN 1995 - International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE, Perth, WA, Australia (1995). https://doi.org/10.1109/ICNN.1995.488968
20. Kovačević, A., et al.: Automatic detection of long method and god class code smells through neural source code embeddings. Expert Syst. Appl. **204**, 117607 (2022)
21. Lewowski, T., Madeyski, L.: How far are we from reproducible research on code smell detection? a systematic literature review. Inf. Softw. Technol. **144**, 106783 (2022). https://doi.org/10.1016/j.infsof.2021.106783, https://linkinghub.elsevier.com/retrieve/pii/S095058492100224X
22. Lindauer, M., et al.: SMAC3: A versatile bayesian optimization package for hyperparameter optimization. J. Mach. Learn. Res. **23**, 1–9 (2021). https://doi.org/10.48550/ARXIV.2109.09831
23. Madeyski, L., Lewowski, T.: MLCQ: industry-relevant code smell data set. In: Proceedings of the Evaluation and Assessment in Software Engineering, pp. 342–347. ACM (2020). https://doi.org/10.1145/3383219.3383264
24. Madeyski, L., Lewowski, T.: Detecting code smells using industry-relevant data. Inf. Softw. Technol. **155**, 107112 (2023). https://doi.org/10.1016/j.infsof.2022.107112, https://linkinghub.elsevier.com/retrieve/pii/S095058492200221X
25. Mhawish, M.Y., Gupta, M.: Predicting code smells and analysis of predictions: using machine learning techniques and software metrics. J. Comput. Sci. Technol. **35**(6), 1428–1445 (2020). https://doi.org/10.1007/s11390-020-0323-7
26. Palomba, F., Bavota, G., Penta, M.D., Fasano, F., Oliveto, R., Lucia, A.D.: On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. Empir. Softw. Eng. **23**(3), 1188–1221 (2017). https://doi.org/10.1007/s10664-017-9535-z
27. Pedregosa, F., Varoquaux, G., Gramfort, A., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
28. Pereira dos Reis, J., Brito e Abreu, F., de Figueiredo Carneiro, G., Anslow, C.: Code smells detection and visualization: a systematic literature review. Arch. Comput. Methods Eng. **29**(1), 47–94 (2021). https://doi.org/10.1007/s11831-021-09566-x
29. Santos, J.A.M., Rocha-Junior, J.B., Prates, L.C.L., Nascimento, R.S.D., Freitas, M.F., Mendonça, M.G.D.: A systematic review on the code smell effect. J. Syst. Softw. **144**, 450–477 (2018). https://doi.org/10.1016/j.jss.2018.07.035
30. Understand by Scitools. https://www.scitools.com/. Accessed 31 May 2023)
31. Sharma, T., Efstathiou, V., Louridas, P., Spinellis, D.: Code smell detection by deep direct-learning and transfer-learning. J. Syst. Softw. **176**, 110936 (2021). https://doi.org/10.1016/j.jss.2021.110936, https://www.sciencedirect.com/science/article/pii/S0164121221000339
32. Shen, L., Liu, W., Chen, X., Gu, Q., Liu, X.: Improving machine learning-based code smell detection via hyper-parameter optimization. In: 2020 27th Asia-Pacific Software Engineering Conference, pp. 276–285. Singapore, Singapore (2020)
33. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems, vol. 25. Curran Associates, Inc. (2012)

34. Sobrinho, E.V.D.P., De Lucia, A., Maia, M.D.A.: A systematic literature review on bad smells-5w's: which, when, what, who, where. IEEE Trans. Softw. Eng. **47**(1), 17–66 (2021). https://doi.org/10.1109/TSE.2018.2880977
35. Soomlek, C., van Rijn, J.N., Bonsangue, M.M.: Automatic human-like detection of code smells. In: Soares, C., Torgo, L. (eds.) DS 2021. LNCS (LNAI), vol. 12986, pp. 19–28. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88942-5_2
36. Sukkasem, P., Soomlek, C.: Enhance machine learning-based code smell detection through hyper-parameter optimization. In: 20th International Joint Conference on Computer Science and Software Engineering (JCSSE). IEEE (2023)
37. Tempero, E., et al.: The qualitas corpus: a curated collection of java code for empirical studies. In: 2010 Asia Pacific Software Engineering Conference, pp. 336–345. IEEE (2010). https://doi.org/10.1109/APSEC.2010.46, http://ieeexplore.ieee.org/document/5693210/
38. Vatanapakorn, N., Soomlek, C., Seresangtakul, P.: Python code smell detection using machine learning. In: 2022 26th International Computer Science and Engineering Conference (ICSEC), pp. 128–133. IEEE (2022)
39. Virmajoki, J., Knutas, A., Kasurinen, J.: Detecting code smells with AI: a prototype study. In: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 1393–1398 (2022). https://doi.org/10.23919/MIPRO55190.2022.9803727
40. Wang, T., Golubev, Y., Smirnov, O., Li, J., Bryksin, T., Ahmed, I.: Pynose: a test smell detector for python. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 593–605. IEEE (2021)
41. Zakeri-Nasrabadi, M., Parsa, S., Esmaili, E., Palomba, F.: A systematic literature review on the code smells datasets and validation mechanisms. ACM Comput. Surv. **55**, 1–48 (2023). https://doi.org/10.1145/3596908
42. Zhang, H., Cruz, L., Van Deursen, A.: Code smells for machine learning applications. In: Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI, pp. 217–228 (2022)