EasyChair Preprint

№ 7717

On Transfer Learning in Code Smells Detection

Moabson Ramos, Rafael de Mello and Baldoino Fonseca

April 3, 2022

On Transfer Learning in Code Smells Detection

Moabson Ramos
Institute of Computing
Federal University of Alagoas
Alagoas, Brazil
amsr@ic.ufal.br

Rafael de Mello
Institute of Computing
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
rafaelmello@ic.ufrj.br

Baldoino Fonseca
Institute of Computing
Federal University of Alagoas
Alagoas, Brazil
baldoino@ic.ufal.br

***Abstract*—The incidence of code smells is often associated with software quality degradation. Several studies present the importance of detecting and tackling the incidence of smells in the source code. However, existing technologies to detect code smells are dependent on the programming language. Consequently, several programming languages are largely employed by the software community without proper technologies code smell detection. Our study addresses amplifying the availability of code smell detection technologies to different programming languages through transfer learning. For a first evaluation, we selected five programming languages among the ten most used languages according to *StackOverflow*: Java, C#, C++, Python, and JavaScript. The datasets for training and testing were obtained from selected open sources projects. Preliminary results indicate high levels of accuracy in detecting Complex Methods from other programming languages through transfer learning models, except for Python. This finding can help developers and researchers to apply the same code smell detection strategies in different programming languages. We also found that the particular behavior observed with Python is partially due to key structural differences in this programming language.**

***Index Terms*—code smells, detection, machine learning, deep learning**

I. Introduction

During software development, the incidence of code smells has been largely associated to the software quality degradation [11], [18]. In this way, it is important developers properly identifying and combating the incidence of smells in the source code [2], [3]. However, existing technologies for code smell detection are limited to few programming language, most of them addressing only Java. Besides, they lack on considering aspects addressing the specificity of the software system design and the subjectivity of the developers on perceiving the incidence of code smells [4], [5], [7], [8]. Consequently, there are several (and popular) programming languages largely used by the software community without proper technologies for detecting even the most known code smells.

Among the code smell detection technologies, there are several ones based on machine learning [1], [12], [15], which allows calibrating the detection model for certain projects according to the training samples employed. In general, the effectiveness of machine learning solutions for code smell detection has been showed promising and more reliable than traditional approaches, with highlight to deep learning. The growing development of machine learning-based solutions to solve Software Engineering problems follow a trend observed in different domains of knowledge. For instance, machine learning models have been widely employed for developing software solutions in many domains. Examples include applications for supporting medical diagnosis [10], building autonomous cars [9], and detection of fraudulent credit card transactions [6].

The development of conventional machine learning technologies is grounded on composing training datasets for building a trained model. This model is then assessed through testing datasets. However, in several cases it is required from machine learning approaches to be trained from similar models. It may happen, for instance, when developers from a particular programming language (e.g., Python) could not identify a smell detection tool available for this programming language. Alternatively, they would employ datasets of code smells detected in another programming language (e.g., Java) to learn from it. Transfer learning consists of using pre-trained models to solve specific tasks, to solve another related task as a way of reducing the efforts (i.e., data relabeling and computational resources) needed to solve it [14].

In this paper, we report one empirical study aiming at to investigate the feasibility of employing transfer learning for detecting the incidence of the Complex Method code smell among different programming languages. Through this study, we expect to contribute for extending the availability of code smell detection technologies through transfer learning, contributing with the software development community to identifying code smells in a larger and diverse samples of software projects. To our study, we composed curated training and evaluations datasets of from five popular programming languages: C++, C#, Java, Javascript and Python. Then we applied two machine learning algorithms with different combinations of these datasets.

The results revealed a trend for reaching high levels of effectiveness on using transfer learning among pairs of different programming languages for detecting Complex Method, except for the case in which the training dataset is composed of sourcecode written in Python. Our following analysis indicated that this behaviour was due some relevant structural differences that even accelerate the side effect of negative transfer [19] in higher sample sizes. Besides, we found that simple ML models based on a single perceptron are effective for identifying Complex Methods through transfer learning.

Identify applicable funding agency here. If none, delete this.

The high levels of effectiveness reached by our study also contrasts with the findings from [17] for the same evaluation dataset.

Section II presents related work on code smell detection and transfer learning. Section III describes the settings of our empirical study. Section IV presents the results of our study, reporting its main findings addressing each research question. Section V discuss the main threats to validity identified in our study. Finally, Section VI concludes the paper and indicates future work.

II. Related Work

Different methods are used for code smells detection, i.e., detecting code smells automatically. [17] divide them among detection methods based on: (i) metrics; (ii) rules/heuristics; (iii) history; (iv) optimization; and most recently, (v) machine learning. The abstract syntax tree (AST) is commonly used in metric-based methods for computing relevant metrics of the source code. Once the thresholds of these metrics are established, they are employed for detecting code smells. However, some code smells can not be detected only using metrics, such as rebellious hierarchy, missing abstraction, cyclic hierarchy, and empty catch block. For this purpose, heuristic-based methods are proposed. In such methods, simple rules or complex heuristics are defined according to the code smell type. History-based methods analyze the evolution of source code to spot code smells. The optimization-based methods are supported by optimization algorithms such as genetic algorithms.

Most recently, machine learning-based models emerged as a promising method for code smell detection. For training and composing these models, machine learning algorithms such as Support Vector Machine, Bayesian Belief Networks, and Logistic Regression have been employed. [15] study made a comparison between the use of the *DECOR* heuristic and a machine learning model based on the *Naive Bayes* algorithm. Both methods performed poorly. This study mentions the fact that the dataset has a wide variety of code smells and has real examples (i.e. manually validated) as a possible cause of low performance, indicating that the success of these methods can be related to the dataset and the results may be unsuitable for use in practice. On the other hand, the study by [12] evaluated the detection of 6 code smells by 7 machine learning algorithms. The results indicated that overall the models were able to reach their best accuracy with few samples, indicating to be a very promising method that deserves to be explored in depth.

Besides the more recent dissemination of machine learning models, there are several smell detection tools available. These tools are predominantly based on metrics and rules/heuristics. [13] compared four tools for code smells detection and presents supported programming languages, covered smells, advantages and disadvantages. For instance, the *InFunsion* is a commercial tool that supports the detection of 22 different types of code smells in Java, C, and C++ programs. *JDeodorant* is a open-source *plugin* for *Eclipse*[1], supporting the detection of four types of code smells in Java programs. All the tools analyzed by the authors support code smell detection in Java programs. Two of them support C/C++ programming languages, while only *PMD* can detect code smells in some other programming languages.

[16] investigated the feasibility of applying transfer learning based on deep learning models for code smells detection. Focusing on four types of smells (complex method, empty catch block, magic number, and multifaceted abstraction), the study investigated the transfer learning between datasets composed of source code written in distinct programming languages. These datasets were extracted from open-source projects at GitHub. The authors conducted the study in two steps: (i) evaluating the use of deep learning models for code smells detection in a C# and Java; and (ii) evaluating transfer learning using pre-trained models between C# and Java. For this study, two architectures of neural networks were selected, Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). The results indicate that is feasible to use deep learning for code smells detection and transfer learning for detecting code smells when training the model with C# samples and applying it in Java samples. However, both models obtained considerably lower results for detecting the multifaceted abstraction smell.

III. Study Design

Our study aims to evaluate the use of transfer learning to detect code smells in five programming languages. The study focuses on three particular aspects: effectiveness, i. e., how effective is transfer learning in correctly identifying or not a code smell into different programming languages; efficiency, i. e., the relation between the effectiveness of the transfer learning and the training effort required from developers to perform a proper transfer learning; and, influence, i. e., which programming language constructs are more influents on applying transfer learning. In this sense, we try to answer the following research questions:

- **RQ1: How effective is transfer learning to detect code smells?** *To answer this RQ:* we investigates the effectiveness of transfer learning to detect code smells into five programming languages. As a result, we expect to identify which programming languages are more proper to produce deep learning models that can be applied in projects with different languages.
- **RQ2: How influential are programming languages constructs to detect code smells by using transfer learning?** *To answer this RQ:* we analyze the influence of language constructs in the effectiveness of transfer learning to detect code smells. In particular, we investigate the influence of three constructs: preprocessor directives, switch statements, and conditional expressions. As a result, we hope to better understand the influence (positive or negative) of these constructs on model accuracy.

[1] www.eclipse.org

- **RQ3: How efficient is transfer learning to detect code smells?** *To answer this RQ:* we intend to measure the necessary effort, i.e. number of samples, to model reach a good accuracy using transfer learning. As a result, we expected to obtain the *f-measure* of the models when trained with ascending number of samples and evaluated using the datasets of different programming languages.
- **RQ4: How complex should the deep learning models be for effectively detecting code smells?** *To answer this RQ:* we analyze the results of models of both architectures (*Perceptron* and *CNN*) when evaluated in the same programming language in which was trained, and by transfer learning. As a result, we intend to determine whether advantages of using different deep learning architectures for code smells detection.

A. Programming Languages

We selected five programming languages for our study: Java, C#, C++, JavaScript and Python. These programming languages figure out between the 10 most used ones according to the 2021 StackOverflow survey[2]. To observe the effect of transfer learning, we intentionally selected script-based programming languages and compiler-based ones. These languages present some of the specific characteristics. For instance, C++ and C# are the only programming languages analyzed in our study that offer the resource of *preprocessor directives*. Preprocessors are commonly employed for introducing variability at compilation time. Also, one can see that Python didn't offer a mechanism to control flow similar to *switch statements* until a more recent version (3.10). In this version, was introduced a feature called *Structural Pattern Matching* with support for pattern matching (at the moment is a preview feature in Java). Consequently, it was expected that most of the source code implemented in Python at the time of this study did not employ this feature. Another relevant difference from Python to the other programming languages investigated is the syntactical differences between conditional expressions.

B. Detection Rule for Complex Method

We decided to focus on code smells at method level. Therefore, the smell *Complex Method* was selected and in return as we mentioned, we include five programming languages and two neural network architectures. Also, the choice of this smell will make it possible to compare it with the study of [16] for the Java and C#. The datasets employed for training and evaluating the detection models were composed based on the rules and thresholds defined in the DesigniteJava[3] tool. If a certain method satisfy

$$CyclomaticComplexity \geq 8$$

then, it's classified has smelly or otherwise as non-smelly.

[2]https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language

[3]https://github.com/tushartushar/DesigniteJava

C. Projects Samples

We manually selected 30 open-source projects from GitHub[4] for each programming language analyzed in our study, i.e., 150 projects in total. We applied several quality criteria for selecting these projects, including recent repository activities, higher number of commits, stars and forks. For each programming language, these projects include frond-end applications, libraries, APIs and frameworks.

D. Metrics

We collected metrics from *Abstract Syntax Tree* (AST) of source code, such as number of switch statement, lambda expressions, nested class, nested function, if, for, while and others. A subset of these metrics were used to compute the *Cyclomatic Complexity* to then classify a method as Complex Method.

E. Data Collection

We found difficulties in employing open-source tools available for gathering the metrics needed from the code snippets selected for the five programming languages. The first difficulty was in finding a tool that covers all the programming languages of our study. The second was that the available tools often do not indicate the exact lines of interest of our study. For script-based languages like JavaScript and Python, there are few or no tools, some existing ones haven't been updated for years. To overcome this challenge, we built our own tool based on the Tree Sitter[5] library. With this tool, we could parse the source code of the selected projects for analyzing the Abstract Syntax Tree (AST), gathering the necessary metrics, and obtaining the desired code snippet for the experiment. Our tool also enabled to filter datasets according to the presence of certain language constructs.

F. Composing the Datasets

Table I presents the datasets used in our study. The first column describes the name of the dataset. The second column describes the goal of the dataset. The third column describes the programming languages of the code snippets that have this dataset. The last column describes the size of the dataset. We randomly extracted one training dataset (named *Train*) and two evaluation datasets (named *Test 1* and *Test 2*). We composed two additional datasets with only methods containing *preprocessor directives* for the programming languages offering this resource. We also composed additional datasets containing *conditional expression* and *switch statements*, the latter doesn't include Python. With these sets, we intend to observe the influence of certain constructs on the effectiveness of transfer learning.

The datasets employed for training and evaluating the detection models were composed based on the rules and thresholds defined in the DesigniteJava[6] tool. If a certain

[4]https://github.com

[5]https://tree-sitter.github.io

[6]https://github.com/tushartushar/DesigniteJava

method satisfies the detection rule, the tool spot this method as smelly. Otherwise, the tool spot the method as non-smelly.

TABLE I
DATASETS USED IN THE STUDY FOR TRAINING AND EVALUATION

Name	***Goal***	***Programming Languages***	***Total Size***
Train	Model Training	All	1000
Test 1	Overall Model Evaluation	All	500
Test 2	Model evaluation employed by [16]	Java and C#	2000
Conditional Expression	Model evaluation for methods containing conditional expressions	All	500
Preprocessor Directives	Model evaluation for methods containing preprocessor directives	C++ and C#	500
Switch Statements	Model evaluation for methods containing switch statements	Java, C#, C++ and JavaScript	500

G. Transfer Learning Models

We opted by first applying a simple *Perceptron* model to observe the effect of transfer learning in different programming languages. Our choice this model is due to as it is a single-layer neural network used for binary classification. In a second moment we employed models based on *Convolutional Neural Network*. The use of different architectures can we bring interesting insights about the use of more complex architectures when compared to simpler.

IV. Results and Discussion

We report in this section the study results by research question, discussing its main findings.

RQ1. How effective is transfer learning to detect code smells?

In RQ1, we applied the *Perceptron* model over the training dataset of each programming language. Then, for each trained model, we used the model to detect smells in the remaining datasets. For example, if we train a model in the dataset containing code snippets from Python projects, then we use this model to detect smells in the datasets containing JavaScript, C++, Java, and C#. Also for comparison purposes we evaluated the model in projects of the same programming language. Table II present the main results to support this discussion.

TABLE II
F-MEASURE OF THE PERCEPTRON MODELS WHEN EVALUATED USING TEST 1 DATASETS

	f-measure				
	Java	C#	C++	JavaScript	Python
Java	**98%**	96%	95%	93%	92%
C#	98%	**98%**	96%	93%	91%
C++	97%	97%	**96%**	94%	95%
JavaScript	96%	95%	95%	**96%**	93%
Python	75%	77%	88%	77%	**97%**

Table II describes the effectiveness of the model when trained in projects containing a specific programming language and, then, applied in projects of the same programming language and containing the other languages analyzed in our study. We observe a trend on reaching lower *f-measure* when training a model in Python and we applied on Java, C#, and JavaScript projects. We obtained a *f-measure* slightly higher when we applied this model on C++ projects. On the other hand, when we trained the model in Java, C#, C++, and JavaScript, we obtained an f-measure greater than 90%. This result indicates that the all models were effective when applied in the same programming language and through of transfer learning from Java, C#, C++, JavaScript but not when the model was trained in Python projects.

Summary of RQ1: Transfer learning between different programming languages is effective to detect Complex Methods except when the training dataset is composed of code snippets implemented in Python.

RQ2. How influential are programming language constructs to detect code smells by using transfer learning?

One possible explanation for the lower effectiveness reached by the Python training model (RQ1) is the common difference of the Python constructs when compared with the constructs of the other programming languages investigated. To check this hypothesis, we composed model evaluation datasets focused on code snippets composed of specific constructs (see Table I) which presence and composition considerably vary among the five programming languages investigated, i.e., *conditional expressions*, *switch statements* and *preprocessor directives*. The *f-measure* reached after applying these evaluation datasets is summarized in Table III.

Regarding conditional expressions, we observe that the differences among the programming languages do not influence the transfer learning process once the distributions of *f-measure* values reached are close to those found for the *Test 1* dataset. In this sense, it is important to note that all the programming languages investigated offer conditional expressions. Besides, despite the syntactical differences, their implementation is quite similar.

Similarly, we could not observe significant differences in the models' effectiveness when applying evaluation datasets composed only by code snippets with preprocessor directives, despite C++ and C# being the only programming languages offering this resource. One possible explanation for this finding is that these directives are commonly composed of simple instructions that would barely contribute to the complexity of the methods.

On the other hand, we observed considerably low *f-measure* values for the Python model evaluated through datasets composed only by code snippets with switch statements. Besides the already mentioned lack of this resource in Python (until a recent version), one can see that the misuse of switch statements is a common root cause of complex methods. Therefore, the inability to recognize this resource may lead to several false negatives.

TABLE III
F-MEASURE OF THE PERCEPTRON MODELS WHEN EVALUATED USING CONSTRUCTS DATASETS

	f-measure										
	Conditional Expressions					**Switch Statements**				**Preprocessor Directives**	
	Java	C#	C++	JavaScript	Python	Java	C#	C++	JavaScript	C#	C++
Java	**93%**	95%	93%	93%	90%	**88%**	87%	88%	89%	93%	90%
C#	93%	**93%**	93%	91%	90%	89%	**88%**	87%	91%	**93%**	89%
C++	94%	94%	**93%**	87%	90%	89%	87%	**85%**	83%	93%	**89%**
JavaScript	91%	93%	90%	**93%**	87%	88%	89%	87%	**90%**	92%	87%
Python	85%	84%	91%	86%	**89%**	48%	53%	47%	34%	78%	86%

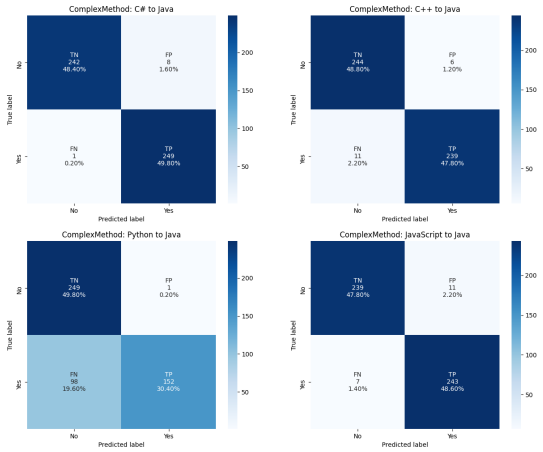

Fig. 1. Confusion Matrix: *Perceptron* model trained using the dataset of Python when it was evaluated using *Test 1* dataset of Java

To further investigate the low *f-measure* scores found for the *Python* model, we looked in depth the confusion matrix (i. e., *true-positive*, *true-negative*, *false-positive*, and *false-negative*) of the models trained in Python, C#, C++, and JavaScript projects for detecting code smells in Java projects (Test1), as depicted in Figure 1. Despite the *true-negative* values are quite similar, one can see that the *false-positive* and *true-positive* values of the model trained in *Python* are slightly lower than those reached by the other models. Consequently, we also observe that the *false-negative* score of the model trained in *Python* projects is higher than the other models. While we obtained 19.6% of *false-negatives* for the model trained in *Python*, the models trained in the other programming languages reached less than 2.2%.

Summary of RQ2: Except for the Python model, the effectiveness of transfer learning between different programming languages for detecting complex methods is not influenced by the incidence of specific constructs in the code snippets evaluated. The *Python* model reached a high percentage of *false-negatives* for detecting complex methods in code snippets from other programming languages. The lack of switch statements in Python and the common influence of this resource over the methods' complexity tend to exacerbate this behaviour when the complex methods are composed of switch statements.

RQ3: How efficient is transfer learning to detect code smells?

So far, we had evaluated models trained with 1,000 samples from each programming language. However, it is important to analyze the extent to which the results found are influenced by this sample size. For this purpose, we opted by focusing on the original training set of the Python programming language, for which evaluations were less favorable and more variable. We randomly composed exponentially lower subsamples from the Python training dataset. Then, we evaluated how the model derived from each subsample behaved for detecting complex methods in all programming languages investigated using the *Test 1* evaluation datasets.

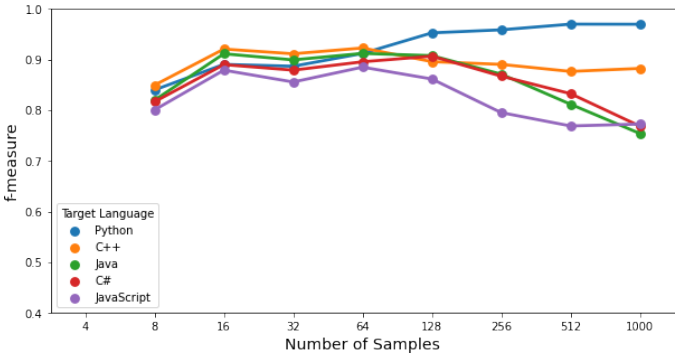

Fig. 2. F-measure of the *Perceptron* model trained with Python subsamples when it was evaluated using the *Test 1* datasets

In Figure 2 presents the *f-measure* scores reached for each model by programming language. The sample sizes appear in the *x-axis*, and the corresponding *f-measure* scores can be observed in the *y-axis*. One can see that the models derived from the Python training datasets present a trend of increasing effectiveness as we increase the sample size for evaluating the dataset in the same programming language. The effectiveness of the Python models for the C++ evaluation dataset presents a slight decrease until 512 samples. Surprisingly, the effectiveness of the same models for the remaining programming languages presents a clear trend of decreasing for models trained with more than 64 samples.

Summary of RQ3: As the sample size of the training dataset increase, the resulting model in Python also increases its efficacy for evaluating code snippets in Python. On the other hand, we observed an inverse behavior in models resulting from training sets higher than 64 code snippets on evaluating code snippets from other programming languages.

RQ4: How complex should the deep learning models be to detect code smells effectively?

Models using a relatively simple neural network architecture have achieved a good classification level through transfer learning except the model trained in Python projects. To check if choose of neural network architecture should be influence on classification, we also use *Convolutional Neural Network* another popular architecture used on tasks like classification of texts, images, and others.

We compared the results with those already obtained from the model using *Perceptron*. The main results about this question are presented in Tables IV, V and Figure 3.

TABLE IV
F-MEASURE OF THE CNN MODELS WHEN EVALUATED USING TEST 1 DATASETS

	f-measure				
	Java	C#	C++	JavaScript	Python
Java	**98%**	95%	95%	94%	94%
C#	98%	**98%**	96%	94%	92%
C++	97%	97%	**96%**	93%	96%
JavaScript	97%	97%	94%	**97%**	91%
Python	77%	78%	88%	82%	**98%**

Comparing the results between Table II and Table IV we observe that the models of both architectures also were able to identify the code smell well in projects of the same language, maintaining a high *f-measure*. On transfer learning the results of Table IV also shows that the *f-measure* of model trained in Python projects remain almost unchanged with slightly increases.

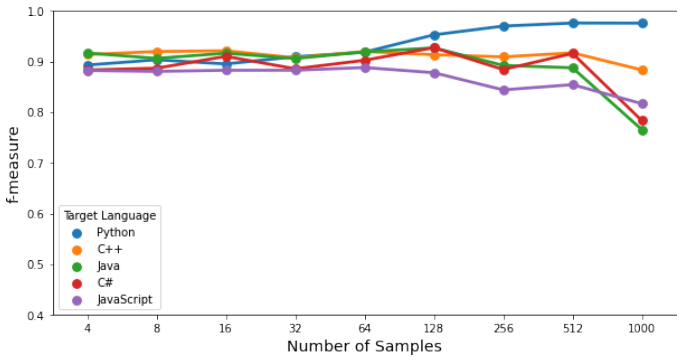

Fig. 3. F-measure of the *CNN* model trained with Python subsamples when it was evaluated using the *Test 1* datasets

Figure 3 presents the results of *CNN* model trained in Python projects, with only 4 samples the model reach an *f-measure* close to 90% but the behavior is similar to results of the *Perceptron* models presented in Figure 2.

TABLE V
COMPARISON BETWEEN TRANSFER LEARNING RESULTS PRESENTED BY SHARMA AND THOSE OBTAINED THROUGH OUR EXPERIMENTS USING THE TEST 2 DATASETS

	f-measure					
	Our results		Sharma results			
	Perceptron Java	CNN Java	CNN-1D Java	CNN-2D Java	RNN Java	AE Java
C#	98%	98%	44%	50%	40%	48%
C++	96%	97%	-	-	-	-
JavaScript	98%	97%	-	-	-	-
Python	76%	81%	-	-	-	-

As shown in Table V, we also obtained lower *f-measure* values when applying the Python model over the *Test 2* evaluation datasets composed by [16] for the Java and C# languages.

Summary of RQ4: The model using the CNN architecture reach an *f-measure* close to 90% with only 4 examples in projects of all languages but compared to simpler model don't presents significantly advantages.

V. THREATS TO VALIDITY

One common threat to validity on studies based on mining software repositories address the representativeness of the software projects and the code snippets analysed. To mitigate this threat, we applied a set of rigorous criteria for composing our datasets. We selected relevant releases from popular open source projects addressing different technologies and domains for each programming language. We also reused the datasets from previous work [16], for the programming languages available.

In the particular case of the training datasets, we recognize that the automated detection of code smells is prone to error. Indeed this is common threat to vality in large-scale studies addressing machine learning solutions in code smell detection. However, it is important to note that Complex Method is a popular type of code smell, which detection is covered by several tool with similar rules. Once our study is focused in this type of code smell, we made effort to optimize its detection by applying combined detection rules.

Some strcutural differences in the programming langauges may vary according to evolution of each language. These differences may significantly influence in the transfer learning results. In the particular context of our study, we verified that the main constructs analysed are estable in the last versions of the five programming languages investigated. The only execption was to the recent resource for implementing switch case of Python (match). In this sense, we certified that the releases of the Python projects analysed did not employd this resource.

Another threat to validity address the limitations on the parsing tools employed. We need to perform our own parsing

solution for Python and JavaScript. For this purpose, we made effort on identifying and reusing stable technologies as background for implementing these solutions. We also tested our solution over several code elements from both programming languages before running the study.

VI. Conclusions

The purpose of the present work was extend the code smells detection technologies through transfer learning between different programming languages. We decided to focus on Complex Method, because this smell appears in all languages selected for our experiment, some of them without proper technologies to detect it. In this sense, was necessary to obtain pre-trained models based on Perceptron and Convolutional Neural Network (CNN), able to detect this smell in Java, C#, C++ and JavaScript. Then these models were evaluated in different programming languages by applying transfer learning using the datasets for general evaluation and with some constructs of these languages such as *conditional expressions*, *preprocessor directives* and *switch statements*.

The results showed that transfer learning is a effective way to detect code smells in different programming languages. Our pre-trained models were able to identify Complex Method in different languages, except when the models were trained with datasets containing code snippets of Python projects. The Python models got an low f-measure on transfer learning, unlike the models of other languages. This difference with the other models became more accentuated when evaluated in the datasets containing *switch statements*, indicating that programming language constructs can be affect the effectiveness of the model. Also, the models built using different architectures didn't present significant differences in our results.

The contributions were not limited to the results of transfer learning, as a consequence a tool was developed for mining and labeling code snippets of five programming languages

Our findings indicate that is feasible to detect code smells in different programming languages from a pre-trained model of a language. Helping the software community in identifying these smells in several programming languages.

As future work, we intend to (i) extend the investigation about the influence of programming languages constructs on transfer learning, (ii) review the technical literature for identifying additional code smell types to include in the research, and (iii) include on our experiment others pre-trained models and compare the results.

Acknowledgment

This research was supported by CNPq 152179/2020-8.

References

[1] Muhammad Ilyas Azeem, Fabio Palomba, Lin Shi, and Qing Wang. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology*, 108:115–138, 2019.

[2] Ana Carla Bibiano, Wesley K. G. Assunção, Daniel Coutinho, Kleber Santos, Vinícius Soares, Rohit Gheyi, Alessandro Garcia, Baldoino Fonseca, Márcio Ribeiro, Daniel Oliveira, Caio Barbosa, João Lucas Marques, and Anderson Oliveira. Look ahead! revealing complete composite refactorings and their smelliness effects. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 298–308, 2021.

[3] Ana Carla Bibiano, Eduardo Fernandes, Daniel Oliveira, Alessandro Garcia, Marcos Kalinowski, Baldoino Fonseca, Roberto Oliveira, Anderson Oliveira, and Diego Cedrim. A quantitative study on characteristics and effect of batch refactoring on code smells. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11, 2019.

[4] Rafael de Mello, Anderson Uchôa, Roberto Oliveira, Willian Oizumi, Jairo Souza, Kleyson Mendes, Daniel Oliveira, Baldoino Fonseca, and Alessandro Garcia. Do research and practice of code smell identification walk together? a social representations analysis. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6. IEEE, 2019.

[5] Rafael Maiani de Mello, Roberto Oliveira, and Alessandro Garcia. On the influence of human factors for identifying code smells: A multi-trial empirical study. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 68–77. IEEE, 2017.

[6] S. Dhankhad, E. Mohammed, and B. Far. Supervised machine learning algorithms for credit card fraudulent transaction detection: A comparative study. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 122–125, July 2018.

[7] Mario Hozano, Alessandro Garcia, Nuno Antunes, Baldoino Fonseca, and Evandro Costa. Smells are sensitive to developers! on the efficiency of (un)guided customized detection. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 110–120, 2017.

[8] Mário Hozano, Alessandro Garcia, Baldoino Fonseca, and Evandro Costa. Are you smelling it? investigating how similar developers detect code smells. *Information and Software Technology*, 93:130–146, 2018.

[9] R. Hussain and S. Zeadally. Autonomous cars: Research results, issues, and future challenges. *IEEE Communications Surveys Tutorials*, 21(2):1275–1313, Secondquarter 2019.

[10] Konstantina Kourou, Themis P. Exarchos, Konstantinos P. Exarchos, Michalis V. Karamouzis, and Dimitrios I. Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and Structural Biotechnology Journal*, 13:8 – 17, 2015.

[11] Rodrigo Lima, Jairo Souza, Baldoino Fonseca, Leopoldo Teixeira, Rohit Gheyi, Márcio Ribeiro, Alessandro Garcia, and Rafael de Mello. Understanding and detecting harmful code. 2020.

[12] Daniel Oliveira, Wesley K G Assunção, Leonardo Souza, and Alessandro Garcia. Applying Machine Learning to Customized Smell Detection : A Multi-Project Study Omited due to blind review. (Ml).

[13] Thanis Paiva, Amanda Damasceno, Eduardo Figueiredo, and Cláudio Sant'Anna. On the evaluation of code smells and detection tools. *Journal of Software Engineering Research and Development*, 5(1):1–28, 2017.

[14] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[15] Fabiano Pecorelli, Fabio Palomba, Dario Di Nucci, and Andrea De Lucia. Comparing heuristic and machine learning approaches for metric-based code smell detection. *IEEE International Conference on Program Comprehension*, 2019-May:93–104, 2019.

[16] Tushar Sharma, Vasiliki Efstathiou, Panos Louridas, and Diomidis Spinellis. Code smell detection by deep direct-learning and transfer-learning, 2021.

[17] Tushar Sharma and Diomidis Spinellis. A survey on software smells. *Journal of Systems and Software*, 138:158–173, 2018.

[18] Anderson Uchôa, Caio Barbosa, Willian Oizumi, Publio Blenílio, Rafael Lima, Alessandro Garcia, and Carla Bezerra. How does modern code review impact software design degradation? an in-depth empirical study. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 511–522. IEEE, 2020.

[19] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11293–11302, 2019.