Machine learning techniques for code smells detection: an empirical experiment on a highly imbalanced setup

Frederico Caram Luiz
LAIS – Laboratory for Advanced Information Systems, FUMEC University
Belo Horizonte, Brazil
fredcaram@gmail.com

Bruno Rafael de Oliveira Rodrigues
LAIS – Laboratory for Advanced Information Systems, FUMEC University
Belo Horizonte, Brazil
brunorodriguesti@yahoo.com.br

Fernando Silva Parreiras
LAIS – Laboratory for Advanced Information Systems, FUMEC University
Belo Horizonte, Brazil
fernando.parreiras@fumec.br

ACM Reference Format:
Frederico Caram Luiz, Bruno Rafael de Oliveira Rodrigues, and Fernando Silva Parreiras. 2019. Machine learning techniques for code smells detection: an empirical experiment on a highly imbalanced setup. In *XV Brazilian Symposium on Information Systems (SBSI'19), May 20–24, 2019, Aracaju, Brazil.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3330204.3330275

1 INTRODUCTION

Code smells, also known as code bad smells, are "a surface indication that usually corresponds to a deeper problem in the system" [12]. Introduced by Fowler in 1999[11] where the author conceptualize each of them and also provide some guidance on refactoring them. Thereafter their impact on software maintainability and flexibility were targeted [28]. But even though their concepts are clearly defined, their identification is still subjective to the developer's interpretation [11]. Studies found that even among experienced developers working in the same application the existence of a given code smell may not be a consensus [7, 14]. The manual detection of code smells is time consuming, non-repeatable and does not scale so it could benefit from the usage of automated approaches for code smell identification [26]. Studies on the automated identification were proposed [6, 8, 25, 31], although they make the identification of code smells an easier task, they still fail to bring context, domain, size and design of the system to the identification [4].

Given this context, machine learning based techniques can bring more flexibility [21]. In Software Engineering, machine learning techniques have been used, for example, to predict software fault, effort estimation and defect prediction and so on [34]. There are tools [5] and techniques [31] for the code smell detection in literature, helping the developer to identify potential flaws that he or another developer may have missed initially. Unlike traditional approaches, machine learning can adapt through user feedback to identify code smells without the need to set default parameters such as size or other metric. Even though the usage of machine learning based techniques are recent when compared to static rules and metrics approaches and consequently has less studied and tested techniques, it is growing steadily. But we still lack comparable results that allows us to identify which one should be used for each smell, this is aggravated by the subjectivity of the code smells so that even for the same system the smells selected by the developers are very likely to differ from one another [30]. Experiments also rely on a positive/negative annotation [7, 10, 13, 29], what can slow down the development of further works since existing empirical works aim at identifying positive smells but do not include the negative ones [14, 26, 28, 30], so using a positive/unlabeled can make use of those existing works without the need to add new annotations.

In order to address these problems, this study aims to developing the state-of-art machine learning techniques in a standardized dataset using a positive/unlabeled setup in order to create a benchmark for future work. For that end, this research answer the following questions:

(1) How does the baseline models perform on the selected dataset?
(2) How the techniques recommended for imbalanced datasets perform when compared to the recommended techniques?
(3) How the best performing techniques behave in a unknown dataset?

To answer these questions, we developed an experiment based on the machine learning techniques for code smells identification that better performed in literature. We used a public dataset named landfill [30] composed of 2 databases that sums 1770 annotated smells spread across 8 different types of smells. The usage of an open and standardized data also contributes to the replication of the study and consequent comparison of the used methods. Since it only provides positive examples of annotated smells, this represents the machine learning problem as a positive/unlabeled problem. The main reason that led us to choose a positive/unlabeled setup instead of the more common positive/negative was that it allows us to work with datasets that only has positive annotations. Enabling the techniques to be used in a broader range of existing datasets, since datasets with only positive annotation are more abundant in the existing experiments. It is also more common to identify only positive annotations in the code source, since there is no practical application in identifying what is not a code smell. Until the date of this research, we did not find any work that used a positive/unlabeled setup to identify code smells. The study resulted in

SBSI'19, May 20–24, 2019, Aracaju, Brazil
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7237-4/19/05…$15.00
https://doi.org/10.1145/3330204.3330275

the development of a reproducible and open source smell identification model[1], implementing the state-of-art technique identified in the literature, as well as some technique proven to work better under this experiment setup. We found that Boosting and Ensemble models proved to work better for this experiment than the ones identified on literature for the used dataset. They also proved to generalize well for other datasets with different annotations.

This paper was organized according to the following structure: Section 2 presents works related to this project; Section 3 provides a background about the code smells and machine learning techniques necessary for this experiment; Section 4 addresses the methodology used in this work; Section 5 displays the results of the study; Section 6 discusses the results; Section 7 presents the threats posed to the validity of the study and finally Section 8 shows the conclusion and provides suggestions for future work.

2 RELATED WORK

Code smell is a recurring subject in researches in software engineering, even though machine learning can be used to reduce uncertainties, few works about machine learning and code smells are finding in literature. One of the first works using Machine Learning techniques were developed by Kreimer(2005)[22] which uses Decision Trees algorithms to identify Large Classes and Long Methods smells, Khomh (2009) [18], Kosker (2009) [20] and Khomh (2011) [19] relies on Bayesian approaches to identify anti-patterns, but while the first one uses a Bayesian Network approach, the second one uses a Naive Bayes and the last one uses "Goal, Question and Metric" in combination with the Bayesian approach. Fontana(2013) [10] uses the following WEKA algorithms: Support Vector Machines (SMO, LibSVM), Decision Trees (J48), Random Forest, Naive Bayes and JRip, for the identification of Long Method, Large Class and Feature Envy extracted from projects from Qualitas Corpus, this work is further evolved by Fontana(2016) [7] where it uses 16 different WEKA algorithms and also adds variations of these algorithms with boosting techniques. Fu (2015) [13] and Palomba(2015) [29] uses Association Rules approaches based on the code repository history, while the first focuses on Speculative Generality and Divergent Change smells the second focuses on Divergent Change, Feature Envy, Parallel Inheritance and Shotgun Surgery.

Our work differs from the existing works by tackling the following gaps: The first one is the lack of a common dataset that can be reused, in our work this is addressed by the use Landfill [30], an open dataset that contains 8 annotated different kinds of smells and can be reused by future experiments. Furthermore, we did setup the model as a positive/unlabeled problem, making it easier to be reused in existing empirical datasets, where only the positive annotations are known, but also accepts positive/negative inputs. We will also experiment using other kinds of techniques, that are more suited for heavily imbalanced data such as One Class Classification and Ensemble techniques and also techniques such as XGBBoost, CatBoost, LightGBM which were not adopted by previous study.

3 BACKGROUND

In this Section, we described the code smells and machine learning used in this work.

[1]https://github.com/lais-fumec/machine-learning-4-code-smells-detection

3.1 Code smells

Code smells represents design choices that may lead to a future degradation on maintainability, understandability and changeability of a given part of the code [11], but even though smells are hints that the code may present a design problem it does not necessarily indicates one [12]. Below the definition of each of the selected code smells are described, we will use Mantyla(2003) [24] taxonomy to classify each of them, but will also classify them by the code entity they affect and the quality aspects they affect as defined by Marinescu(2005) [27]:

- **Divergent Change:** represent classes that change together whenever another class changes. Classes which have this smell usually demonstrates a high coupling, it is classified as a change preventer class and affects inter-class interactions.
- **Feature Envy:** a method that is more interested in other properties of the classes than in the ones from its own class. This kind of smell affects the coupling, cohesion and encapsulation design aspects of the system, representing a problem in the abstract design of the system. It is classified as coupler smell and affects method/property entities.
- **Large Class:** a class that tries to do a load of things, having plenty of instance variables or methods. A class with this smell tends to present coupling, cohesion and complexity problem, affecting the maintainability and understandability of the class. It is classified as a bloatter smell and affects class entities.
- **Long Method:** a method that is so long that it is hard to understand, change or extend. It also increases the complexibility of the system. It is classified as a bloatter smell that affects method level entities.
- **Parallel Inheritance Hierarchies:** a situation where two parallel class hierarchies exist and are related. It affects the abstraction design of the system and it is classified as an object-orientation abuser and affects inter-class inheritance relationships.
- **Shotgun Surgery:** the shotgun surgery smell exists when changing a given class requires a consequent change on other classes that depends on it, classes with this smell usually has a low cohesion. It is classified as a change preventer class and affects inter-class interactions.

3.2 Machine Learning

Machine learning techniques can be categorized in three groups: supervised, unsupervised and semi-supervised, all of them take features as input, these features used may be categorized as continuous, categorical or binary, depending on their nature [21]. If instances are given with known labels (the human annotated correct output) then the learning is called supervised, otherwise, when the instances are unlabeled, it is unsupervised learning [15]. There is also a hybrid approach, which is the semi-supervised learning that uses both labeled and unlabeled data to perform an otherwise supervised learning or unsupervised learning task [35]. This work will focus on the supervised techniques, since we use labeled smells data and most of the experiments use this approach [5, 21]. During this section we will give a brief explanation of the techniques used on the experiment.

- **Association Rules:** is a rules-based technique that aims at identifying relationship between variables that exists in the dataset.
- **Boosting Techniques:** uses the combination of a set of weak learners to build a stronger one, with lower-bias and variance. Some of the most prominent boosting techniques currently are XGBoost [1], LightGBM [16] and CatBoost [2] which were recently developed and are known to be used by the winners of multiple Kaggle contexts.
- **Decision Trees:** classify instances by sorting them based on feature values and splitting them into branches, each branch represents the value thresholds the contained nodes can assume and each node represents a feature.
- **Logistic Regression:** fits a sigmoid function in a linear regression model for binary classification, the function determines the classification probability of each instance and based on a probability threshold determines if it is classified as positive or negative.
- **Naive Bayes:** uses Bayesian Statistics to establish the probability between one unobserved node and a chain of children observed nodes. It assumes an independent relationship between child nodes and their parent.
- **Nearest Neighbors:** also known as k-nearest neighbors is a lazy-learning algorithm algorithm that classifies the items based on their position and distance in a hyper-plane.
- **Neural Networks:** a multi-layer neural network consists of large number of units (neurons) joined together in a pattern of connections, each connection has a weight established by the model, and each node contains an activation function that determines the node value. It usually has one output node for each class and are usually defined by a sigmoid function.
- **Random Forest:** is a tree-based ensemble technique, which uses a bagging of trees models, built using only a subset of the features, the average of those trees is taken to calculate for the features prediction.

4 EXPERIMENT SETUP

In order to compare the performance of the machine learning techniques for code smells identification, an empirical experimental was developed. Empirical Experiments uses empirical studies to build and produce a theory or a model and demonstrate that it is usable from a practical perspective [33]. We conducted a controlled environment, using the same database, validated using 10 times repeated 10-fold cross validations (except for Shotgun Surgery and Divergent Change, where the number of positive classes only allows for 5-folds) in order to keep the same rigor as in the original experiments. The sets are exactly the same for each dataset, so that the only variables that changes during the experiment are the machine learning techniques. Adjustments, such as feature-engineering (Association Rules values extraction) and extraction (such as over/undersampling and class weights adjustments) may have to be done in the sets to be usable with the diverse techniques, but they ought be done in a way that do not affect the outcome of the project.

For the development of the experiment we used the Python language as it is one of the most used languages for Machine Learning purpose, also having a huge ecosystem and community. We used MongoDB as our database since it is a flexible document oriented database. For data handling we used the popular Pandas and Numpy python frameworks, for the basic Machine Learning and Ensemble models the Sklearn and Scipy frameworks were used, while for the boosting models we used the respective XGBBoost, LightGBM and CatBoost packages and Mlxtend for the Apriori implementation. Considering that Sklearn does not have pruned J48 and a J-Rip implementations we used the R language to reproduce them and used the Rpy2 package to make it communicate with our pipeline. To help in the definition of the scope we defined a goal, according to the baselines defined by Wohlin(2012)[32]: Analyze the Machine Learning Techniques for Code Smells identification for the purpose of evaluation with respect to their effectiveness and efficiency from the point of view of a researcher in the context of a standardized environment that can be replicated and compared by future experiments.

4.1 The smells dataset

For the purpose of replicability, we used a public dataset named Landfill, since it constitutes the largest collection of manually validated smells publicly available [30]. It is composed of two databases, each contains around 20 different projects with annotated smells, summing up to 1770 annotated smells spread across eight different types. Since this work is focused on objected-oriented smells we discard two test-related smells: Eager Test and General Fixture. What leaves 6 smells remaining: Divergent Change, Feature Envy, Large Class, Long Method, Parallel Inheritance and Shotgun Surgery.

We downloaded the project snapshot for each project contained in the dataset and used the Metrics Reloaded [23] Idea IntelliJ plugin to download the Metrics required for each technique, for the metrics that were not supported by the plugin, we used the tool used in the given experiment. For the Association Rules smells, we used the change history provided by the same authors of the Landfill dataset that used them on the history mining study [29], and extracted the Association Rules from them. As the experiment was setup to be a positive/unlabeled problem, we merged the whole metrics data with the annotated smell, the ones that matched were classified as positive, while the remaining metrics were classified as Unlabeled.

4.2 Code Smells detection strategy

A comprehensive number of strategies for the detection of code smells on the source code is present on the literature, as the focus of this work is on Machine Learning techniques, we will focus on the best strategies found on literature using machine learning techniques for each of the covered smells.

- **Divergent Change and Shotgun Surgery:** the top performing article for this smell uses the change history in the code repository to identify how often the classes changes together with others. An Association Rules (Apriori) technique was used to identify the support, confidence and lift in the classes that presents changes on the same commits as described on the experiment done by [29].

- **Feature Envy, Long Method and Large Class:** the dominant article on this subject in the literature [7] uses the tool JCodeOdor to extract the metrics [9].
- **Parallel Inheritance Hierarchies:** similar to the Divergent Change and Shotgun Surgery detection, this smell is also detected using the code repository change history and uses Association Rules (Apriori) to identify the support, confidence and lift. But instead of checking for classes that presents changes in the same commits it checks for superclasses that changes together when a subclass is added. For the other techniques, we kept the Association Rules variables (confidence, support and lift) as features, since they provide valuable information, but instead of using a hard threshold we fit them to the model.

4.3 Evaluated models

Based on the smells supported by the Dataset we selected the best performing technique for each of them based on the literature, the smells and the respective associated technique can be seem on the table 1.

Smell	Techniques	Reference
Divergent Change	Association Rules	[29]
Feature Envy	Boosted JRip	[7]
Large Class	Naive Bayes	[7]
Long Method	Boosted J-48 Pruned	[7]
Parallel Inheritance	Association Rules	[29]
Shotgun Surgery	Association Rules	[29]

Table 1: Best performing technique for each smell

For each of the smells we also developed models recommended by the literature for highly imbalanced datasets such as: One class classifiers (Such as One class SVM or Local Outlier Detection), Boosting (Random Forest, XGBBoost, LightGBM, CatBoost) and Ensemble (multiple ML techniques with a soft voting classifier) [17], a weight adjustment proposed by Elkan(2008)[3] and a under/oversampling technique. Since Random Forest was recommended by the article and is also a Boosting model, it fits into two categories, so when it appears as recommended by the Article we will classify it as Article recommendation instead of as a boosting model.

5 RESULTS

In this Section we present the results of the empirical experiment. Firstly, some descriptive statistics of the used dataset are demonstrated, after the results of the literature selected models are showed and finally the results of techniques recommended for positive/unlabeled learning and compare with the baseline results are presented.

5.1 Overview

To understand how the smells are distributed the dataset, we present a descriptive statistics on table 2. It is possible to notice in this table that the dataset is highly imbalanced, the smells represents less than 1% of the set for most of the smells. The smell ratio also variate a lot from project to project, as demonstrated by the Deviation, given that for every project we have a deviation that surpasses 100% its ratio. This difference may occur due to the differences in the quality of the smell, but also due to the project size or the developers knowledge and experience in the given project [26].

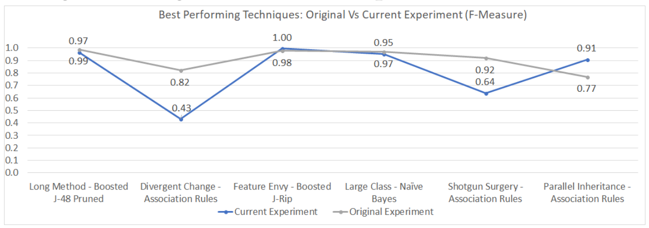
Table 2: Basic descriptive stats from the smells

Smell	Projects	Smells By Project	Unlabeled By Project	Ratio By Project	Std. Dev
Large Class	36	6.86	989.22	0.0120	0.0218
Long Method	19	21.58	753.26	0.0270	0.0361
Feature Envy	27	4.67	2,659.96	0.0059	0.0089
Parallel Inheritance	7	2.71	645.00	0.0173	0.0306
Divergent Change	9	1.11	1,203.78	0.0024	0.0024
Shotgun Surgery	5	2.80	1,498.80	0.0036	0.0056

5.2 How does the baseline models perform on the selected dataset?

We replicated the best performing studies found in the literature for each smell in the dataset we prepared for this experiment, the results of the replicated results were then compared to the results reported on the original experiment as can be seem on figure 1. For most of the studied smells, we were able to achieve a similar performance, performing slightly worst on Long Method and Large Class and better at Feature Envy and Parallel Inheritance. But for the Shotgun Surgery and Divergent Change we had divergence, with a significant impact on the algorithm performance. Since we used the same method and steps as described on the original experiment, we believe that this difference may be due to some missing information from the original experiment or due to the influence of the random seed over the Association Rules and Apriori Method.

Figure 1: Original x Current Experiment (F-Measure)

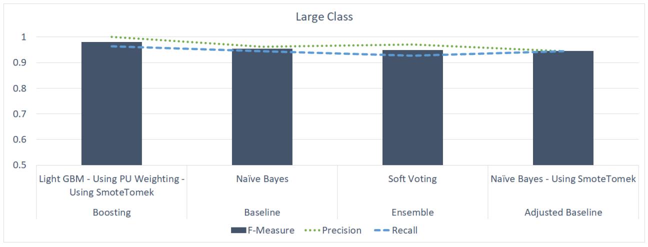


5.3 How the techniques recommended for imbalanced datasets perform when compared to the recommended techniques?

For each of the studied smells we tested a set of approaches for highly imbalanced data: One class classifiers (Such as One class SVM or Local Outlier Detection), Boosting (Random Forest, XGBBoost, LightGBM, CatBoost) and Ensemble (multiple ML techniques with a soft voting classifier). We also tried them with and without under/oversampling combination technique and the positive/unlabeled weight adjustment. For the charts below we selected only the best performing techniques of each category.

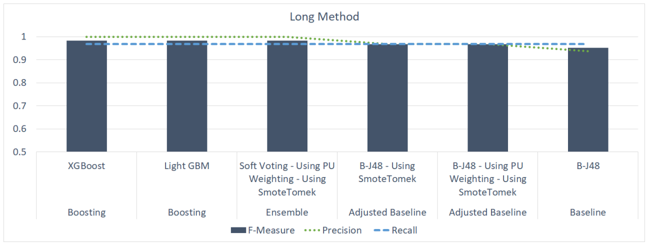
Regarding the Large Class smell, the best performing technique was the LightGBM, it outperformed the recommended technique by 0.02 (1.6%), it presented a better performance in both precision and recall. The usage of under/oversampling technique and the weight adjustment resulted in worst performance than the recommended technique by 0.01 (1%), caused by a loss in the recall, while the Ensemble model also had a small loss of 0.02 (5%), with a better precision than the baseline, but a slightly worst recall. The isolation forest classifier performed below the baseline by 0.004 (0.4%), with a slightly better precision but a worst recall. The chart with the Large Class results can be seem on figure 2.

Figure 2: Large class results



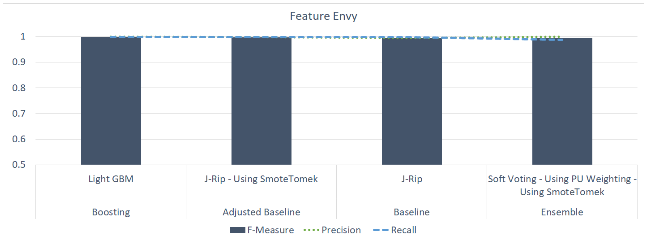
The long method smell presented a small variance on the results. The best performing techniques were the LightGBM boosting technique, that presented the same results as the Ensemble technique, it outperformed the recommended technique by 0.02 (1.6%), having a better performance in both precision and recall. The usage of under/oversampling technique and the wight adjustment resulted in a slightly better performance than the recommended technique by 0.005 (0.6%), caused by a gain in the precision that compensated the worst recall. As shown in the figure 3.

Figure 3: Long method results



The feature envy also had just a small variance when compared to the Article recommended technique, the best performing technique was the LightGBM which performed 0.005 better than the baseline (0.5%), using recommended by the baseline but using PU weighting adjustments also improved the performance by 0.002 (0.2%) with a better recall but worst precision than it, the Ensemble technique did not perform well though, scoring 0.001 (0.1%) with a precision better than the baseline but a worst recall. As can be seem on figure 4.

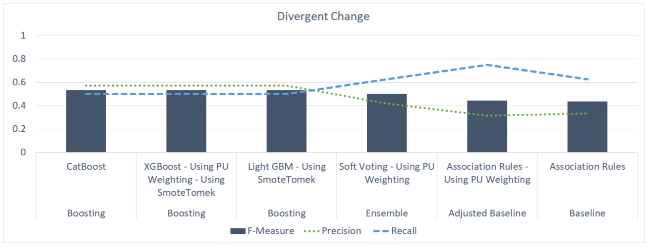
Figure 4: Feature Envy results



For the next three smells: Divergent Change, Shotgun Surgery and Parallel Inheritance, given that in the original experiment they used a hard threshold to define the smell, for the calculation of the adjusted model with PU weighting and under/oversampling techniques we used these technique for recalculating the threshold.

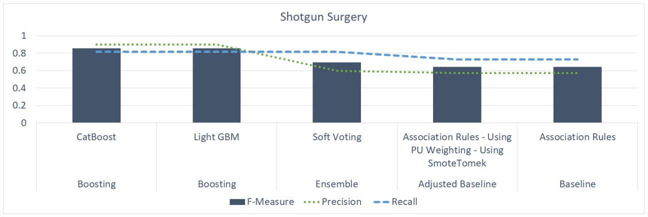
For divergent change smell the LightGBM, XGBoost and Catboost models demonstrated the best performance with a score 0.09 (21%) above the Baseline, it presented a better precision, but a smaller recall. While the Ensemble technique performed 0.06 (14%) better than the baseline with a better recall but worst precision when compared to the boosting techniques. The adjusted model had just a small effect on the results. It is important to draw attention to the fact that none of these models were able to achieve a result similar to the one achieved in the original experiment.

Figure 5: Divergent Change results



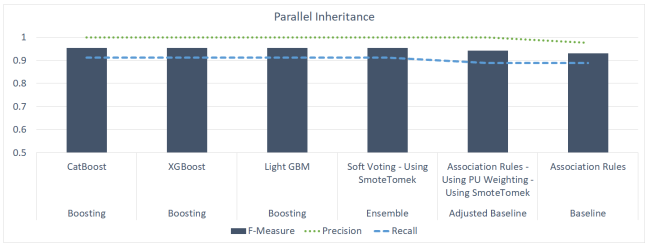
The shotgun surgery best performing models were also Boosting Techniques based, with Catboost and LightGBM presenting the same performance, scoring 0.21 (34%) above the baseline, with a loss on recall but a huge gain on precision. It was followed by Ensemble, with a gain of 0.05 (8%). The adjusted model had the same performance as the original model. Even though the techniques didn't present a performance as good as the one reported in the original experiment, they were able to close the gap significantly.

Figure 6: Shotgun Surgery results



When it comes to Parallel Inheritance CatBoost, XgbBoost and LightGBM achieved the same performances, outperforming the original technique by 0.045 (5%), presenting a better precision and recall. It was followed by the Ensemble technique, which performed 0.033 (3.7%) better than the baseline for this smell, using a weight adjusting technique also demonstrated to improve the performance, improving the baseline by 0.02(1%).

Figure 7: Parallel Inheritance results



For all the smells the One Class Classification Techniques performed poorly, even though it achieved a high recall it wasn't able to obtain a relevant precision, since it searches for outliers it is probable that the algorithms could not find a clear boundary between the smells and the unlabeled set, since the difference in the metrics is subtle, bringing the results to a lower score.

The positive/unlabeled weighting adjustment helped improving the score in 1/2 of the highest scoring techniques, while the under/oversampling technique helped it in 1/3 of the top techniques. They also were able to improve the baseline techniques performance in 4/6 of the techniques, on the other 2 it performed the same. From these 4 that achieved a better performance, 2 used a combination of the weight adjustment and under/oversampling, 1 used only the weight adjustment and 1 used only the under/oversampling.

5.4 How the best performing techniques behave in a unknown dataset?

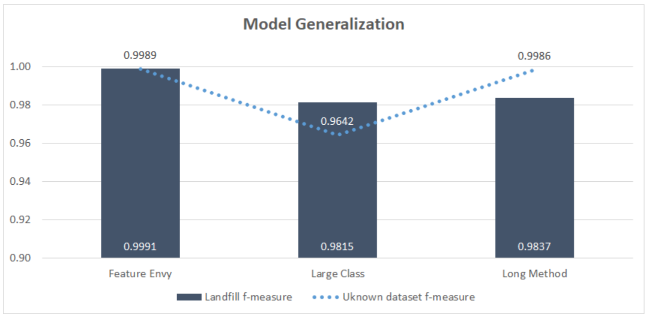
In order to assess the generalization capability of the best performing techniques, we executed it against the original experiment datasets. For the Feature Envy, Large Class and Long Method smells the models proved to generalize very well. With the Long Method model performing even better (1.51% improvement) in the unknown dataset than in the Landfill, while the feature envy had almost the same performance (0.02% variation) and the Long method performed slightly worst (1.76% worst). The comparisons can be seem on figure 8.

It was not possible to execute this same assessment for the Divergent Change, Shotgun Surgery and Parallel Inheritance smells, since the original experiment dataset was used to compose the Landfill dataset, so executing the models against it would not prove anything, since the original experiment dataset is included in the Landfill one.

6 DISCUSSION

This study tried to create a reproducible experiment recreating the best performing machine learning techniques learning applied for code smells identification found in the literature. The study

Figure 8: Model generalization: Landfill dataset f-measure x unknown dataset f-measure



also added some techniques recommended for highly imbalanced datasets, that characterizes the setup of this experiment.

Most of the replicated techniques performed closely on this experiment when compared to the original experiment they were performed, except for the Divergent Change and Shotgun Surgery smells. We believe that their poorer performance is a result of having fewer annotated examples than the other smells, what reduces their capability to generalize to different scenarios and make it more likely to overfit. While the original experiments used positive/negative annotated code, ours used a positive/unlabeled approach, what can makes it easier to be used on existing experiments, where the researchers only have access to positive annotated examples. This setup makes it harder to achieve a high precision without hurting the recall and vice-versa due to the high imbalance between the number of positive annotations and the unknown part.

Even though most of the models performed as well on the replicated experiment as they did on the original experiment, the Parallel Inheritance replication was able to greatly surpass the result of the replicated techniques. One possible reason for it is that the combination of its simple model with a decent number of annotations make it better for generalization. On the other hand the Shotgun Surgery and Divergent Change models performed much worst than on the original experiment, what can be partially explained by the smaller number of annotations, what makes it harder to generalize well. We also found that the smells which relied heavily on metrics performed better on the replicated experiment than the ones which relied on the change history, this may happen due to the hard thresholds used on the Association Rules extracted for the change history, which made the original experiments less flexible.

The models which performed the best for most of the smells were the LightGBM based models, XGBBoost and CatBoost also had a good performance in all the techniques, both of them are from the Boosting algorithms category, and it was expected that they would present a good performance and generalization capability on highly imbalanced datasets. Another model that performed well for most of the smells are the Ensemble based models, which did not performed as good as the Boosting techniques but performed very closely to them.

Most of the best performing technique did not found the weight adjustments and SmoteTomek techniques to be useful for improving their performance, since most of them were boosting algorithms, the bagging in combination with the weak learners may have the same effect. The One Class Classification techniques were not able to reach any significant performance for any of the smells, even though it was able to reach high recall rates, it failed to reach a good recall. Since it relies heavily on outlier detection the poor performance demonstrates that the boundaries between what is and what is not a smell can't be clearly defined only by the used features, it may also rely on some adjustable probability.

7 THREATS TO VALIDITY

We have selected the best performing techniques based on the literature, but since the compared techniques used different datasets and even the studies that use the same project can use different annotations to train the data, consequently it may decrease the reliability of the performance comparisons. This threat is increased by publication bias as the researchers tend to release only positive results, avoiding the negative ones. In order to mitigate this risk we also tested our techniques against the original datasets to ensure that our models can perform well under data that was unknown to it.

Another possible threat is that some snapshots used by the dataset does not exist anymore, in order to reduce this risk we used snapshots taken around the same time of the ones proposed in the original dataset and manually re-validated it. There were are also some annotations that failed to merge with our dataset due to non-standard nomenclature, what can reduce the annotation size and consequently impact the performance.

8 CONCLUSIONS

This study replicated the best performing machine learning techniques for code smell identification to create a benchmarking in a standardized annotated dataset for the evaluation and posterior comparison of the techniques. We also used a different setup from the original experiment, while the original ones used a positive/negative annotation with the rates between positive and negative varying between 1/3 and 1/2 smells per non smells, ours used a positive/unlabeled setup, which used the positive annotations from the Dataset and the whole code as the unknown set, consequently our rates varied between 0.3% to 2%, we believe that a positive/unlabeled setup can be more useful for the techniques application, since many experiments have only the positive annotations.

We found that most of the techniques close to the result they obtained in the original experiment. But some of the Association Rules techniques could not perform as good as they back then. Using techniques that are best fit for this kind of imbalanced data proved to improve the performance under this setup but also can perform and generalize well on positive/negative scenarios. Among the studied smells we found that Boosting models were best fit for this kind of setup, performing better on all studied smells.

We believe that the smells identification techniques could benefit from using a positive/unlabeled setup since it would eliminate the need for Negative annotations, making it easier to acquire a proper dataset, it would also avoid a possible selection bias, since the developers may tend to get more extreme negative examples from the code. Future experiments can also benefit from the developed experiment to run benchmarks where they can check their techniques performance against proven techniques without needing manual labor to annotate smells.

REFERENCES

[1] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[2] A. V. Dorogush, V. Ershov, and A. Gulin. Catboost: gradient boosting with categorical features support, 2017.

[3] C. Elkan and K. Noto. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 213–220. ACM, 2008.

[4] V. Ferme, A. Marino, and F. A. Fontana. Is it a Real Code Smell to be Removed or not? In *International Workshop on Refactoring & Testing (RefTest) 2013*, Wien, Austria, 2013.

[5] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo. A review-based comparative study of bad smell detection tools. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering - EASE '16*, pages 1–12, Limerick, Ireland, 2016. ACM.

[6] M. Fokaefs, N. Tsantalis, and A. Chatzigeorgiou. Jdeodorant: Identification and removal of feature envy bad smells. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 519–520, Paris, France, 2007. IEEE.

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

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

[9] F. A. Fontana, V. Ferme, and M. Zanoni. Towards assessing software architecture quality by exploiting code smell relations. In *Proceedings of the Second International Workshop on Software Architecture and Metrics*, pages 1–7. IEEE Press, 2015.

[10] F. A. Fontana, M. Zanoni, A. Marino, and M. V. Mäntylä. Code smell detection: Towards a machine learning-based approach. In *IEEE International Conference on Software Maintenance, ICSM*, pages 396–399, Eindhoven, The Netherlands, 2013.

[11] M. Fowler and K. Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[12] M. Fownler. Codesmell. https://martinfowler.com/bliki/CodeSmell.html, Feb. 2016. (Accessed on 01/14/2018).

[13] S. Fu and B. Shen. Code Bad Smell Detection through Evolutionary Data Mining. In *International Symposium on Empirical Software Engineering and Measurement*, pages 41–49, 2015.

[14] M. Hozano, A. Garcia, B. Fonseca, and E. Costa. Are you smelling it? investigating how similar developers detect code smells. *Information and Software Technology*, 93:130–146, 2018.

[15] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[16] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3149–3157, 2017.

[17] S. S. Khan and M. G. Madden. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(3):345–374, 2014.

[18] F. Khomh, S. Vaucher, Y. G. Gueheneuc, and H. Sahraoui. A Bayesian Approach for the Detection of Code and Design Smells. In *Quality Software, 2009. QSIC '09. 9th International Conference on*, pages 305–314, Jeju, Korea, 2009.

[19] F. Khomh, S. Vaucher, Y. G. Guéhéneuc, and H. Sahraoui. BDTEX: A GQM-based Bayesian approach for the detection of antipatterns. *Journal of Systems and Software*, 84(4):559–572, 2011.

[20] Y. Kosker, B. Turhan, and A. Bener. An expert system for determining candidate software classes for refactoring. *Expert Systems with Applications*, 36(6):10000–10003, 2009.

[21] S. B. Kotsiantis. Supervised Machine Learning : A Review of Classification Techniques. *Informatica, An International Journal of Computing and Informatics*, 3176(31):249–268, 2007.

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

[23] B. Leijdekkers. Metricsreloaded, 2017.

[24] M. Mantyla, J. Vanhanen, and C. Lassenius. A taxonomy and an initial empirical study of bad smells in code. *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, OCTOBER:381–384, 2003.

[25] M. Mantyla, J. Vanhanen, and C. Lassenius. Bad smells: humans as code critics. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pages 399–408, Chicago, Illinois, USA, 2004. IEEE.

[26] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *IEEE International Conference on Software Maintenance, ICSM*, pages 350–359,

Chicago, Illinois, USA, 2004. IEEE.

[27] R. Marinescu. Measurement and quality in object-oriented design. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 701–704. IEEE, 2005.

[28] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka. The Evolution and Impact of Code Smells : A Case Study of Two Open Source Systems What are code smells ? In *Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement*, April, pages 390–400, Orlando, FL, USA, 2009. IEEE Computer Society.

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

[30] F. Palomba, D. D. Nucci, M. Tufano, G. Bavota, R. Oliveto, D. Poshyvanyk, and A. D. Lucia. Landfill : an Open Dataset of Code Smells with Public Evaluation. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, pages 482–485, Florence, Italy, 2015. IEEE.

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

[32] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[33] M. V. Zelkowitz and D. R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23–31, 1998.

[34] D. Zhang and J. J. Tsai. Machine learning and software engineering. *Software Quality Journal*, 11(2):87–119, Jun 2003.

[35] X. Zhu and A. B. Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.