Singling the Odd Ones Out: A Novelty Detection Approach to Find Defects in Infrastructure-as-Code

Stefano Dalla Palma
s.dalla.palma@uvt.nl
JADS, Tilburg University
The Netherlands

Majid Mohammadi
m.mohammadi1@tue.nl
JADS, Eindhoven University of Technology
The Netherlands

Dario Di Nucci
d.dinucci@uvt.nl
JADS, Tilburg University
The Netherlands

Damian A. Tamburri
d.a.tamburri@tue.nl
JADS, Eindhoven University of Technology
The Netherlands

ABSTRACT

Infrastructure-as-Code (IaC) is increasingly adopted. However, little is known about how to best maintain and evolve it. Previous studies focused on defining Machine-Learning models to predict defect-prone blueprints using supervised binary classification. This class of techniques uses both defective and non-defective instances in the training phase. Furthermore, the high imbalance between defective and non-defective samples makes the training more difficult and leads to unreliable classifiers. In this work, we tackle the defect-prediction problem from a different perspective using *novelty detection* and evaluate the performance of three techniques, namely OneClassSVM, LocalOutlierFactor, and IsolationForest, and compare their performance with a baseline RandomForest binary classifier. Such models are trained using only non-defective samples: defective data points are treated as *novelty* because the number of defective samples is too little compared to defective ones. We conduct an empirical study on an extremely-imbalanced dataset consisting of 85 real-world Ansible projects containing only small amounts of defective instances. We found that novelty detection techniques can recognize defects with a high level of precision and recall, an AUC-PR up to 0.86, and an MCC up to 0.31. We deem our results can influence the current trends in defect detection and put forward a new research path toward dealing with this problem.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; • **Computing methodologies → Semi-supervised learning settings**.

KEYWORDS

Infrastructure-as-Code, Novelty Detection, Defect Prediction

MaLTeSQuE '20, November 13, 2020, Virtual, USA

ACM ISBN 978-1-4503-8124-6/20/11…$15.00
https://doi.org/10.1145/3416505.3423563

ACM Reference Format:
Stefano Dalla Palma, Majid Mohammadi, Dario Di Nucci, and Damian A. Tamburri. 2020. Singling the Odd Ones Out: A Novelty Detection Approach to Find Defects in Infrastructure-as-Code. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE '20), November 13, 2020, Virtual, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3416505.3423563

1 INTRODUCTION

The DevOps principle for the automated configuration management of systems infrastructure, known as Infrastructure-as-Code (IaC), is emerging as the de-facto challenge for software maintenance and evolution of the coming years. IaC is used to address problems regarding the manual process of configuration management using automatic provision and configuration of infrastructural resources based on practices from software development through the definition of machine-readable files, *a.k.a. blueprints*. These practices, such as version control and automated testing, ensure consistent and repeatable routines for system provisioning and configuration changes [9]. While automatically managing configurations makes the process more efficient and repeatable, new problems can emerge: practitioners frequently change the infrastructure, which inadvertently introduces defects [15] whose impact might be significant. For example, the execution of a defective IaC blueprint erased the directories of about 270 Wikimedia users in 2017[1]

Because of its novelty, only a few studies have been proposed to address issues related to the quality of infrastructure code. The first steps in this direction focused on building supervised binary classifiers for predicting defect-prone blueprints to help DevOps engineers scheduling testing and maintenance activities, thereby saving resources and time. Typically, the process for building such classifiers consists of generating instances from software archives such as version control systems, in the form of software components (e.g., classes, methods, etc.). An instance is characterized using several metrics (a.k.a., features) extracted from those components, such as the number of lines of code, and is labeled as "defect-prone" or "defect-free". The labeled instances are used to build the ground truth for a machine learning classifier to learn the features that discriminate and predict defects in a specific component.

However, defect prediction models struggle with imbalanced datasets where the "defect-free" class is observed more frequently

[1] https://wikitech.wikimedia.org/wiki/Incident_documentation/20170118-Labs.

than the "defect-prone" class. Like in general-purpose defect prediction [4], IaC scripts represent a minority of the code artifacts present in software projects. Indeed, Jiang and Adams [7] observed that in open-source repositories, only a median of 11% of the files co-existing with source code files is IaC scripts. This lack of (defective) data is reflected and highlighted in the IaC scope, making the collection of defective instances and consequent training more challenging, leading to unreliable classifiers. These motivations lead to the following research question:

To what extent can we detect anomalous IaC blueprints with a machine learning approach using only information of ordinary blueprints?

To address the research question, we tackle the problem of predicting defects in IaC from a different perspective using *novelty detection* and evaluate the performance of three techniques, namely OneClassSVM, LocalOutlierFactor, and IsolationForest, and compare their performance with a baseline RandomForest binary classifier. In this approach, models are trained using only non-defective samples because the number of defective samples is too little compared to the former. In contrast, defective data points are treated as *novelties*, namely data-points differing from the distribution of the training data that are considered "abnormal". We conduct an empirical study on an extremely-imbalanced dataset collected from 85 real-world Ansible projects containing only small amounts of defective instances. Our results show that novelty detection techniques can recognize defects with a high level of precision and recall, an Average Precision, *a.k.a.* Area Under the Curve Precision-Recall (AUC-PR) up to 0.86, and a Matthews Correlation Coefficient (MCC) up to 0.31. They bound to influence current trends in defect detection of IaC and put forward a new research path toward dealing with the problem.

Structure of the paper. The remaining of this paper is structured as follows: Section 2 introduces the concept of novelty detection and the techniques tested in this work; Section 3 describes the design of the empirical study; while Section 4 discusses the experiments results. Section 5 discusses the limitation of our work and how we mitigated them. Section 6 describes the related work on defect prediction of infrastructure code. Finally, Section 7 concludes the paper and outlines future works.

2 BACKGROUND

To the best of our knowledge, our work is the first studying the use of novelty detection methods for IaC defect prediction. The following section will introduce the topic and describe the selected techniques.

Defect Prediction and IaC. Before the appearance of standard tools to provision and configure cloud infrastructure declaratively, DevOps engineers used to write scripts in general-purpose, procedural languages, which execute a series of statements using control flow logic like if statements and while loops. Thus, combining the *what* and the *how*. Modern provisioning and configuration management tools, like Ansible, keep the *what* and *how* separate by using a declarative language, i.e., a set of statements that declare the result one wants.

tasks/main.yml

```
...  @@ -1,18 +1,18 @@
1      ---
2      # File: main.yml - Main tasks for Consul
3
4      - name: Install python dependencies
5        when:
6          - consul_install_dependencies | bool
7        block:
8          - name: Install netaddr dependency on controlling host (with --user)
9            pip:
10             name: netaddr
11             extra_args: --user
12           delegate_to: 127.0.0.1
13           become: false
14           run_once: true
-        when: is_virtualenv is not defined
15       +   when: is_virtualenv == '' or is_virtualenv == None
16
```

Figure 1: An example of failure-prone script.

Figure 1 depicts an IaC script in Ansible[2]. It declares tasks to install Python dependencies, which are grouped by a `block` section (line 7), and executed when given conditions apply (line 6). Each task provides some information, e.g., a name (line 8), a call to a module (line 9), parameters for the module (lines 10-11), conditions (line 15), etc[3]. However, this code does not include any logic about *how to* perform these tasks. Instead, the logic and error-handling are implemented in a separate module in any procedural or object-oriented language. In the example, the task at line 8 calls the module `pip` specifying the package to install (lines 9 and 10), while the logic to install the dependency is implemented in that Python module. As a result, the code is less verbose.

Despite this feature, errors might still occur at the infrastructure code level, e.g., build failures, crash due to access to un-existing resources, and wrong comparisons. Indeed, a wrong condition in Figure 1 (line 15, red) caused the system to fail. In this example, the *"playbook tries to run the task "Install netaddr dependency on controlling host (virtualenv)" when variable VIRTUAL_ENV is not set in VM. Then is_virtualenv is defaulted to " " what is treated as defined value. Then task without user arg is invoked, resulting in error caused by lack of permissions on VM."*[4] The fix (line 15, green) required checking the variable against empty string or None. Although somehow similar to GPL applications, these errors require ad-hoc metrics for defect prediction of IaC. These metrics can be extracted to analyze the IaC script and be used as features in machine learner models, as described in Section 3.

Novelty Detection. Novelty detection is the task of classifying test data that differ in some respect from the available data during training [13]. This may be seen as "one-class classification", in which a model is constructed to describe "normal" training data

[2]From https://github.com/ansible-community/ansible-consul/commit/91efd4c95d03f269ab626a319d5f3e4058abca8b
[3]For the sake of space, we only reported a piece of the script. For the complete version of the script see footnote 2
[4]From https://github.com/ansible-community/ansible-consul/issues/161

(for example "defect-**free**" blueprints). The novelty detection approach is commonly used for anomaly detection. The goal is to detect abnormal or unusual observations and when the quantity of available "abnormal" data (for example "defect-prone" blueprints) is insufficient to construct explicit models for non-normal classes. This condition is particularly true in the early stages of the software development life-cycle, where there is little or no information about defective code. In novelty detection, models are typically constructed with numerous IaC blueprints indicative of normal functional behavior. Previously unseen patterns are then tested by comparing them with the normality model, often resulting in a novelty score. The score is typically compared to a decision threshold, learned by the model during training, and the test data are then deemed to be "abnormal" if the threshold is exceeded [13].

Techniques for Novelty Detection. Although there exist several methods that have been shown to perform well on different data, novelty detection is an extremely challenging task, for which there is no single best model. The success depends not only on the type of method used but also on the statistical properties of data handled [8]. For this reason, in this study, we relied on three widely-used novelty detection methods, described below.

OneClassSVM is a one-class classifier that requires to choose a kernel (e.g., the *rbf* kernel) and a regularization parameter to define a frontier. The latter represents the classifier's margin, also known as the probability of finding a new, but regular, observation outside the frontier.

IsolationForest isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. Since a tree structure can represent recursive partitioning, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node. Once averaged over a forest of such random trees, the path length becomes a measure of normality. When a forest of random trees collectively produces shorter path lengths for particular samples, they are highly likely anomalies.

Finally, LocalOutlierFactor computes a score (called local outlier factor) reflecting the degree of abnormality of the observations. The idea is to detect abnormal data points by measuring their local density deviation and compare it to their k-nearest neighbors: a normal instance is expected to have a local density similar to that of its neighbors. In contrast, abnormal data are expected to have a much smaller local density.

3 DESIGN OF THE STUDY

This section describes the dataset and the pipeline to construct and validate the models to investigate the role of novelty detection for the prediction of defective IaC blueprints.

Dataset. Only a handful of IaC datasets are publicly available and ready to use for defect prediction. Therefore, for the sake of convenience, we focused on the RADON Defect Prediction Tool Validation Dataset available on Zenodo [11]. The dataset is the first large publicly-accessible of its kind: it provides over 180k observations of defect-prone and defect-free IaC blueprints collected from 85 open-source GitHub repositories based on the Ansible language. The ratio of the number of defective scripts to the total instances is approximately 6% on average. Thus, relevant for this study. Each observation in the dataset represents the snapshot of a given IaC file collected at each project release and contains the value for 108 IaC-oriented (46), delta (46), and process metrics (16). The former capture the structural properties of the infrastructure source code. They were proposed by Dalla Palma et al. [3], and include eight traditional source code metrics that are applicable to IaC, such as the number of *lines of code*, *comments*, and *conditions*; 14 metrics introduced by Rahman et al. [15] and validated in the context of defect prediction for Puppet and ported to Ansible, such as the number of *commands*, *module parameters*, and *file access*; and 24 metrics related to best and bad practices in Ansible, such as the number of *deprecated modules*, *suspicious comments*, *tasks with unique name*, and more. The second capture the amount of change in a file between two successive releases, and were collected for each IaC-oriented metric. The latter capture aspects of the development process rather than aspects about the code itself, and include metrics such as the number of developers that changed a file, the total number of added and removed lines, and the number of files committed together.[5] It is worth noting that, to ensure independence among observations, we sampled observations such that for each project, we analyzed only its latest release.

Novelty Detection for Defect Prediction. The novelty detector for defect prediction implemented in this work relies on *scikit-learn* [12], a Python framework to build the pipeline to pre-process the dataset, select the features, train and validate the machine learning models, and use them to identify unseen abnormal instances. The pipeline applies different configurations in terms of hyper-parameters for each step[6], as described below:

(1) *Data normalization.* In this step, numeric features were scaled individually in the range [0, 1].
(2) *Feature selection.* Although the dataset was originally intended for defect prediction, not all features may be significant for the task. We used the `SelectKBest` feature selection algorithm implemented in *scikit-learn* to remove all but the k highest scoring features, i.e., those with the highest values for a given scoring function, in our case a chi-squared statistic. This method requires the training data to contain only non-negative features, a condition ensured by Step 1.
(3) *Classification.* The normalized selected attributes and the learning algorithms described in Section 2 were used to build the learner. Before testing it, the original test data were normalized in the same way as the training data, and the dimensionality was reduced to the same subset of features. After comparing the predicted value and the actual value of the test data, the performance of one *step* of validation was obtained.

The final output consists of a *json* file that reports information for each validation step.

[5] A description of the process metrics in this dataset can be found at https://pydriller.readthedocs.io/en/latest/processmetrics.html
[6] The replication package, along with hyper-parameters, is reported in the online appendix available at https://github.com/stefanodallapalma/MALTESQUE2020-IaC-Novelty-Detection

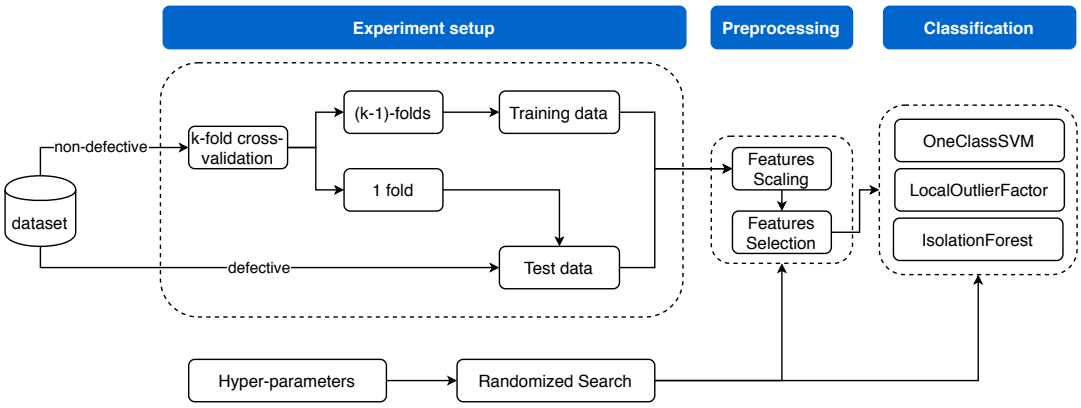

Figure 2: The research flow of defects detection with the novelty detection approach.

Model Validation. The model selection was guided by a randomized search on the models' parameters through a *10-folds* cross-validation, as depicted in Figure 2. More specifically, the model was fit on training data consisting only of observations describing normal behavior (i.e., defect-free) and then used to evaluate new observations. Any new observations differing from the considered classifier's training data were classified as abnormal (i.e., defect-prone).

To evaluate the model performance, we relied on the following evaluation measures, as we believe they are the most suitable for imbalanced datasets: Precision, Recall, AUC-PR [1, 5], and MCC [2]. The performance was analyzed and reported in terms of mean and standard deviation.

4 RESULTS OF THE STUDY

Since previous work [16] has demonstrated the appropriateness of binary classification and showed that RandomForest can achieve significantly good performance for IaC defect prediction in the scope of Puppet code, we compared the result of novelty detection techniques with those of a RandomForest binary classifier as a baseline.

Table 1 shows the experimental results in terms of AUC-PR, MCC, precision, and recall. The tested novelty detection methods perform effectively and consistently in an extremely-imbalanced dataset. The average PR area ranges from 0.82 for the worst-performing technique, namely OneClassSVM, to 0.86 for the best-performing techniques, namely LocalOutlierFactor and IsolationForest. The latter is also the learning method with the lowest standard deviation: and thus yields the most stable results regardless of the selected metrics, even though the three methods have similar standard deviations for the evaluated measures in most cases. Likewise, the average MCC ranges from 0.27 for the worst-performing techniques, namely OneClassSVM and LocalOutlierFactor, to 0.31 for the best-performing technique, namely IsolationForest. All the classifiers have equal, low standard deviation (i.e., 0.06), as can be observed from Table 1. The three classifiers' median prediction performance is 0.86 and 0.27 for AUC-PR and MCC. Although the MCC can be interpreted as the Pearson correlation coefficient, a value around 0.30 could represent a "weak" concordance between predictions and observations. We believe that if considered along with the high AUC-PR, this result indicates a high concordance between predictions and observations.

It is worth noting that all the analyzed novelty techniques strongly out-perform the baseline binary classifier for Ansible code (see the last row of Table 1). As the data come from multiple projects with different characteristics, we suggest that the performance gap is mainly due to how the techniques deal with the data. However, as investigating the differences between the techniques is not in this work's scope, we reserve such an analysis as future work.

5 THREATS TO VALIDITY

At the best of our knowledge, this work is the first of its kind for Infrastructure-as-Code. As such, there may be limitations that we did not or partially considered. Therefore, this section describes the threats that can affect the validity of our study.

Threats to Construct Validity. The data could not be representative of the problem at hand. Nevertheless, we mitigated these threats by using a dataset explicitly created and validated for Defect Prediction of infrastructure code. The projects present in the original dataset were carefully selected based on ten criteria to identify active, IaC-related, and well-engineered software projects stemming from the properties defined by Munaiah et al. [10].[7] The observations were then labeled as defect-prone or defect-free considering the developers' intent where possible (i.e., by analyzing issues and commit messages) and using a state-of-the-art strategy to identify defective scripts (such as the SZZ algorithm). Finally, the metrics were implemented following their documentation and adopting a test-driven approach to make sure the measurements comply with the intended behavior[8].

Threats to Internal Validity. The metrics' choice might significantly influence detection. This threat was mitigated by the comprehensive and complementary set of metrics considered in the original dataset, consisting of metrics related to the infrastructure code's structural characteristics and development process.

[7] A description of each criterion can be found on the dataset page on Zenodo.
[8] The tool used to extract the metrics can be found at https://github.com/radon-h2020/radon-ansible-metrics

Table 1: Performance of the novelty detection methods for IaC defect prediction.

Method	Precision		Recall		AUC-PR		MCC	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
OneClassSVM	0.84	0.01	**0.77**	0.06	0.82	0.02	0.27	0.06
LocalOutlierFactor	**0.86**	0.01	0.70	0.06	**0.86**	0.02	0.27	0.06
IsolationForest	**0.86**	0.01	**0.77**	0.06	**0.86**	0.01	**0.32**	0.06
RandomForest	0.08	0.17	0.01	0.02	0.09	0.04	0.02	0.06

Threats to External Validity. We analyzed 85 Ansible-based projects from different application domains and with different characteristics. Still, we do not know whether and how our results generalize on systems from different ecosystems (e.g., Chef, Puppet). However, we believe that most of the used metrics can be extracted from similar IaC languages. Therefore, we hope to corroborate our findings by analyzing such ecosystems as part of our research agenda.

Threats to Conclusion Validity. The metrics we used to assess our approach are widely adopted in evaluating the performance of novelty detection techniques and in imbalanced setups. To avoid overfitting and sampling bias, we exploited the k-fold cross-validation to evaluate the novelty detection techniques.

6 RELATED WORK

The concept of *anomalous* blueprint is strictly related to code defects and smells. Sharma et al. [17] looked for code smells in the source code of 4,621 Puppet open source repositories. As a result, they proposed a catalog of 13 *implementation* and 11 *design* configuration smells. They observed that design smells showed higher average co-occurrence than the implementation smells: one wrong or non-optimal design decision introduces many quality issues in the future. Rahman et al. [14] presented a catalog of seven security smells in IaC that were extracted from qualitative analysis of Puppet scripts in open source repositories. The identified smells comprise (i) granting admin privileges by default, (ii) empty passwords, (iii) hard-coded secrets, (iv) invalid IP address binding, (v) suspicious comments (such as 'TODO' or 'FIXME'), (vi) use of HTTP without TLS, and (vii) use of weak cryptography algorithms. The work is limited to Puppet scripts, and some smells are not generalizable to other languages or tools.

From a defect prediction perspective, the closest papers were recently proposed by Rahman et al. [15, 16]. The former work exploited textual features of IaC to train a RandomForest binary classifier for defective Puppet scripts, able to achieve between 0.71 and 0.74 in terms of F-Measure. The latter identified 10 source code properties that correlate with defective IaC scripts through qualitative analysis. Using the identified properties, the authors trained different binary classifiers to construct defect prediction models in Puppet code's scope and observed that the DecisionTree and RandomForest classifiers achieved promising results, with a precision up to 0.78, and recall up 0.67.

Our work is complementary to those mentioned above. First, although validated using code defects rather than smells, our work represents the first step toward studying the impact of novelty detection technique on predicting anomalous IaC blueprints, regardless of they contain defects or code smells (i.e., antipatterns). Second, using the information concerning blueprints representative of good behavior solely has the advantage of saving resources and time that are often required to catch defective scripts to train a binary classifier. Ideally, classifying scripts as abnormal only based on "normal" scripts. Finally, we focus on Ansible and not on Puppet, as this is the most spread configuration management technology in the industry to date [6].

7 CONCLUSION

The study's goal was to evaluate novelty detection techniques to predict defective infrastructure code using only information about "normal" samples to evaluate whether and which of the considered novelty detection techniques can achieve high detection performance. Our results show that novelty detection for IaC defect prediction can achieve high precision and recall, with an AUC-PR up to 0.86, and an MCC up to 0.31. We deem our results to put forward a new research path toward dealing with this problem. Researchers should exploit different approaches from the traditional supervised binary classification for defect prediction in Infrastructure-as-Code scripts. Practitioners may find it beneficial to use approaches that consider only "defect-free" samples for training their models. Indeed, novelty detection can potentially reduce the effort to collect relevant information that is challenging to obtain, such as defect-prone scripts in the early stages of infrastructure development.

As future work we plan to (i) examine more imbalanced learning methods to corroborate these results; (ii) analyze more in-depth the comparison with binary classifiers; and (iii) investigate how novelty detection generalizes on software defect datasets from different configuration management languages (e.g., Puppet and Chef).

ACKNOWLEDGMENTS

This work is supported by the European Commission grant no. 825040 (RADON H2020).

REFERENCES

[1] Kendrick Boyd, Kevin H Eng, and C David Page. 2013. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 451–466.

[2] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21, 1 (2020), 6.

[3] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian Andrew Tamburri. 2020. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software* 170 (2020), 110726. https://doi.org/10.1016/j.jss.2020.110726

[4] Rudolf Ferenc, Zoltán Tóth, Gergely Ladányi, István Siket, and Tibor Gyimóthy. 2018. A public unified bug dataset for Java. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*. 12–21.

[5] Mark Goadrich, Louis Oliphant, and Jude Shavlik. 2006. Gleaner: Creating ensembles of first-order clauses to improve recall-precision curves. *Machine Learning* 64, 1-3 (2006), 231–261.

[6] Michele Guerriero, Martin Garriga, Damian A Tamburri, and Fabio Palomba. 2019. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 580–589.

[7] Yujuan Jiang and Bram Adams. 2015. Co-evolution of infrastructure and source code-an empirical study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 45–55.

[8] Markos Markou and Sameer Singh. 2003. Novelty detection: a review—part 1: statistical approaches. *Signal processing* 83, 12 (2003), 2481–2497.

[9] Kief Morris. 2016. *Infrastructure as Code*. O'Reilly Media. https://doi.org/10.1007/978-1-4302-4570-4_9

[10] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empirical Software Engineering* 22, 6 (2017), 3219–3253.

[11] Stefano Dalla Palma. 2020. *Defect Prediction Tool Validation Dataset 1.* https://doi.org/10.5281/zenodo.3906023

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[13] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. 2014. A review of novelty detection. *Signal Processing* 99 (2014), 215–249.

[14] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The Seven Sins: Security Smells in Infrastructure as Code Scripts. In *Proceedings of the 41st International Conference on Software Engineering*. 164–175.

[15] Akond Rahman and Laurie Williams. 2018. Characterizing defective configuration scripts used for continuous deployment. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 34–45.

[16] Akond Rahman and Laurie Williams. 2019. Source code properties of defective infrastructure as code scripts. *Information and Software Technology* 112 (2019), 148–163.

[17] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does your configuration code smell?. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 189–200.