Static Test Flakiness Prediction

Valeria Pontillo
SeSa Lab — Department of Computer Science, University of Salerno
Fisciano, Italy
vpontillo@unisa.it

ABSTRACT

The problem of flakiness occurs when a test case is non-deterministic and exhibits both a passing and failing behavior when run against the same code. Over the last years, the software engineering research community has been working toward defining approaches for detecting and addressing test flakiness, but most of these approaches suffer from scalability issues. Recently, this limitation has been targeted through machine learning solutions that could predict flaky tests using various features, both static and dynamic. Unfortunately, the proposed solutions involve features that could be costly to compute. In this paper, I perform a step forward and predict test flakiness only using statically computable metrics. I conducted an experiment on 18 Java projects coming from the FlakeFlagger dataset. First, I statistically assess the differences between flaky and non-flaky tests in terms of 25 static metrics in an individual and combined way. Then, I experimented with a machine learning approach that predicts flakiness based on the previously evaluated factors. The results show that static features can be used to characterize flaky tests: this is especially true for metrics and smells connected to source code complexity. In addition, this new static approach has performance comparable to the machine learning models already in the literature in terms of F-Measure.

1 INTRODUCTION AND MOTIVATION

During regression testing, the term flaky test is used to describe a test case that shows a non-deterministic behavior when run against the same code. According to the literature, this unexpected behavior does not only make the test result unreliable but (1) may hide real defects and be hard to reproduce [19]; (2) increase testing costs, as developers, invest time debugging failures that are not real [16]; (3) can reduce the overall developer’s confidence on test cases, potentially leading to neglect real defects [9]. The consequences of test flakiness have been made more and more popular by practitioners and companies worldwide (e.g., [10, 21]), who all called for automated mechanisms to detect and predict them. The software engineering research community has been contributing with empirical investigations aiming at eliciting the causes of flakiness [9, 17-20] as well as with the definition of techniques for detecting and addressing them [4, 8, 29, 31]. Despite the efforts made by the researchers, the proposed solutions for both detection and prediction of flakiness are not always optimal since these solutions suffer from poor scalability or involve features that are costly to calculate. It seems necessary to identify alternative techniques for detecting and predicting flaky tests.

2 RELATED WORK

The discussion concerns only the seminal papers on the topic that have inspired this work. From the perspective of flakiness detection, researchers devised alternatives like DeFlaker [4], that works at commit-level and relies on the differential code coverage extracted from the analysis of test execution from a commit to another. Moreover, the use of machine learning approaches has been proposed to predict the presence of a flaky tests. Pinto et al. [24] and further replications [6, 12] exploited the test code dictionary to discriminate the presence of potential flakiness. More recently, Alshammari et al. [1] devised a supervised learning model that, using a mixture of code and coverage metrics, can predict flaky tests with an accuracy up to 86%. While these previous research efforts have shown promising results, they all involve steps that might deteriorate the scalability of the proposed techniques. More particularly, the techniques proposed by Bell et al. [4], and Alshammari et al. [1] require the computation of dynamic features, while the approach by Pinto et al. [24] relies on natural language processing, which is known to be costly as the corpus of the text to analyze increases in size [3]. Recently Pontillo et al. [25] aimed at conducting a feasibility study to assess whether a static prediction of test flakiness would be possible, i.e., whether we could identify likely flaky test cases only based on their design. In particular, the authors analyzed the iDFlakies dataset, and investigated the differences between flaky and non-flaky tests in terms of 25 test and production code metrics and smells. The results achieved by this work indicated the feasibility of devising a static approach to flaky tests prediction.

3 PROPOSED SOLUTION

I replicated the work proposed by Pontillo et al. [25] on the FlakeFlagger dataset, in order to increase the generalizability of the results. This analysis was conducted on a new dataset of 9,785 test cases, including 670 flaky tests. After this initial replication, which showed statistically significant differences between flaky and non-flaky sets for metrics connected to code complexity and assertion, I built on top of the replication to devise a prediction model that

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE ’22 Companion, May 21–29, 2022, Pittsburgh, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9223-5/22/05...
https://doi.org/10.1145/3510454.3522680

1The iDFlakies dataset: https://sites.google.com/view/flakytestdataset/home
2The FlakeFlagger dataset: https://zenodo.org/record/4450723#.YXetWprP2Uk.
could identify flaky tests only considering the design of test cases.
For this study, I evaluated Decision Trees [11], Naive Bayes [30], Multilayer Perceptron [28], and Support Vector Machine [23] as basic classifiers. Additionally, I also considered two ensemble techniques such as Ada Boost [27], and Random Forest [13]. In terms of training, I had to deal with the flaky test problem being unbalanced. The number of flaky test instances represented the 6.8% of the total amount of test cases in the dataset. Hence, before running the models, I applied the Synthetic Minority Oversampling Technique, a.k.a SMOTE [7], to balance the data. I employed a ten-fold cross-validation [5, 14], applying it on both individual projects and considering all projects as a unique dataset. The first necessary step is related to the feature engineering process, that is, the identification of the relevant metrics to use as predictors. While the statistical exercise conducted as the first step provided indications on which features are more connected to test flakiness, it does not necessarily provide insights into the predictive power of the considered metrics [2]. I was interested in assessing the value of the metrics as features of a machine learner more precisely. Hence, I performed a further step ahead by quantifying the predictive power of each metric in terms of information gain [26].

Table 1: List of features with information gain (IG). I have chosen to include features with IG > 0.

<table>
<thead>
<tr>
<th>Features</th>
<th>FlakeFlagger dataset</th>
<th>IG</th>
<th>Random Forest</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>0.1287</td>
<td>0.0253</td>
<td></td>
</tr>
<tr>
<td>Halstead Vocabulary</td>
<td>0.1239</td>
<td>0.0186</td>
<td></td>
</tr>
<tr>
<td>Halstead Length</td>
<td>0.1117</td>
<td>0.0178</td>
<td></td>
</tr>
<tr>
<td>LCOM2</td>
<td>0.1063</td>
<td>0.0152</td>
<td></td>
</tr>
<tr>
<td>WMC</td>
<td>0.0998</td>
<td>0.0101</td>
<td></td>
</tr>
<tr>
<td>MPC</td>
<td>0.0978</td>
<td>0.0061</td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>0.0787</td>
<td>0.0061</td>
<td></td>
</tr>
<tr>
<td>Halstead Volume</td>
<td>0.0558</td>
<td>0.0021</td>
<td></td>
</tr>
<tr>
<td>Spaghetti Code</td>
<td>0.0355</td>
<td>0.0017</td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>0.0305</td>
<td>0.0015</td>
<td></td>
</tr>
</tbody>
</table>

4 PRELIMINARY RESULTS
To verify the presence of possible statistically significant differences between the different machine learning algorithms, I exploited the Nemenyi test [22] for statistical significance and analyzed its results by mean on MCM (Multiple Comparison with the best) plots [15]. The results, obtained with the nemenyi function available in R toolkit,¹ have shown that the best classifier is Random Forest.

Table 1 reports the outcome of the feature engineering process, showing the information gain (IG) obtained when building the model. We can observe that there are 20 features with an IG > 0, and the higher values are related to production and test code complexity measures. Other features with a high IG are Eager Test, Mystery Guest, and Spaghetti Code, meaning that the presence of design flaws, either in production or test code, might provide indications of test flakiness. Perhaps more interestingly, the assert-related features have lower predictive power for what one could have reasonably expected from previous work [25] and my preliminary analysis, in which the assert-related features are the most statistically significant between the two sets. This result seems to suggest that a high number of (undocumented) assertions is connected to test flakiness but not enough to clearly enable its prediction.

Table 2: Results of the Random Forest classifier for the dataset in terms of True Positives, True Negatives, False Positives, False Negatives, Precision, Recall, and F-Measures. The last row ("Total") reports the results when considering all projects as a unique dataset.

4.1 PRELIMINARY RESULTS

To verify the presence of possible statistically significant differences between the different machine learning algorithms, I exploited the Nemenyi test [22] for statistical significance and analyzed its results by mean on MCM (Multiple Comparison with the best) plots [15]. The results, obtained with the nemenyi function available in R toolkit,¹ have shown that the best classifier is Random Forest.

Table 1 reports the outcome of the feature engineering process, showing the information gain (IG) obtained when building the model. We can observe that there are 20 features with an IG > 0, and the higher values are related to production and test code complexity measures. Other features with a high IG are Eager Test, Mystery Guest, and Spaghetti Code, meaning that the presence of design flaws, either in production or test code, might provide indications of test flakiness. Perhaps more interestingly, the assert-related features have lower predictive power for what one could have reasonably expected from previous work [25] and my preliminary analysis, in which the assert-related features are the most statistically significant between the two sets. This result seems to suggest that a high number of (undocumented) assertions is connected to test flakiness but not enough to clearly enable its prediction.

Table 2: Results of the Random Forest classifier for the dataset in terms of True Positives, True Negatives, False Positives, False Negatives, Precision, Recall, and F-Measures. The last row ("Total") reports the results when considering all projects as a unique dataset.

4.2 PRELIMINARY RESULTS

To verify the presence of possible statistically significant differences between the different machine learning algorithms, I exploited the Nemenyi test [22] for statistical significance and analyzed its results by mean on MCM (Multiple Comparison with the best) plots [15]. The results, obtained with the nemenyi function available in R toolkit,¹ have shown that the best classifier is Random Forest.

Table 1 reports the outcome of the feature engineering process, showing the information gain (IG) obtained when building the model. We can observe that there are 20 features with an IG > 0, and the higher values are related to production and test code complexity measures. Other features with a high IG are Eager Test, Mystery Guest, and Spaghetti Code, meaning that the presence of design flaws, either in production or test code, might provide indications of test flakiness. Perhaps more interestingly, the assert-related features have lower predictive power for what one could have reasonably expected from previous work [25] and my preliminary analysis, in which the assert-related features are the most statistically significant between the two sets. This result seems to suggest that a high number of (undocumented) assertions is connected to test flakiness but not enough to clearly enable its prediction.

Table 2: Results of the Random Forest classifier for the dataset in terms of True Positives, True Negatives, False Positives, False Negatives, Precision, Recall, and F-Measures. The last row ("Total") reports the results when considering all projects as a unique dataset.

To verify the presence of possible statistically significant differences between the different machine learning algorithms, I exploited the Nemenyi test [22] for statistical significance and analyzed its results by mean on MCM (Multiple Comparison with the best) plots [15]. The results, obtained with the nemenyi function available in R toolkit,¹ have shown that the best classifier is Random Forest.

Table 1 reports the outcome of the feature engineering process, showing the information gain (IG) obtained when building the model. We can observe that there are 20 features with an IG > 0, and the higher values are related to production and test code complexity measures. Other features with a high IG are Eager Test, Mystery Guest, and Spaghetti Code, meaning that the presence of design flaws, either in production or test code, might provide indications of test flakiness. Perhaps more interestingly, the assert-related features have lower predictive power for what one could have reasonably expected from previous work [25] and my preliminary analysis, in which the assert-related features are the most statistically significant between the two sets. This result seems to suggest that a high number of (undocumented) assertions is connected to test flakiness but not enough to clearly enable its prediction.

Table 2: Results of the Random Forest classifier for the dataset in terms of True Positives, True Negatives, False Positives, False Negatives, Precision, Recall, and F-Measures. The last row ("Total") reports the results when considering all projects as a unique dataset.

To verify the presence of possible statistically significant differences between the different machine learning algorithms, I exploited the Nemenyi test [22] for statistical significance and analyzed its results by mean on MCM (Multiple Comparison with the best) plots [15]. The results, obtained with the nemenyi function available in R toolkit,¹ have shown that the best classifier is Random Forest.

Table 1 reports the outcome of the feature engineering process, showing the information gain (IG) obtained when building the model. We can observe that there are 20 features with an IG > 0, and the higher values are related to production and test code complexity measures. Other features with a high IG are Eager Test, Mystery Guest, and Spaghetti Code, meaning that the presence of design flaws, either in production or test code, might provide indications of test flakiness. Perhaps more interestingly, the assert-related features have lower predictive power for what one could have reasonably expected from previous work [25] and my preliminary analysis, in which the assert-related features are the most statistically significant between the two sets. This result seems to suggest that a high number of (undocumented) assertions is connected to test flakiness but not enough to clearly enable its prediction.

Table 2: Results of the Random Forest classifier for the dataset in terms of True Positives, True Negatives, False Positives, False Negatives, Precision, Recall, and F-Measures. The last row ("Total") reports the results when considering all projects as a unique dataset.

¹https://www.r-project.org/
REFERENCES


