An Evaluation of Machine Learning Methods for Predicting Flaky Tests

Azeem Ahmad[a], Ola Leifler[a] and Kristian Sandahl[a]

[a]Linköping University, 581 83 Linköping, Sweden

Abstract

In this paper we have investigated as a means of prevention the feasibility of using machine learning (ML) classifiers for flaky test prediction in project written with `Python`. This study compares the predictive accuracy of the three machine learning classifiers (Naive Bayes, Support Vector Machines, and Random Forests) with each other. We compared our findings with the earlier investigation of similar ML classifiers for projects written in `Java`. Authors in this study investigated if test smells are good predictors of test flakiness. As developers need to trust the predictions of ML classifiers, they wish to know which types of input data or test smells cause more false negatives and false positives. We concluded that RF performed better when it comes to precision (> 90%) but provided very low recall (< 10%) as compared to NB (i.e., precision < 70% and recall >30%) and SVM (i.e., precision < 70% and recall >60%).

Keywords

Improve Software Quality, Flaky Test Detection, Machine Learning Classifiers, Experimentation, Test Smells

1. Introduction

Developers need to ensure that their changes to the code base do not break existing functionality. If test cases fail, developers expect test failures to be connected to the changes. Unfortunately, some test failures have nothing to do with the code changes. Developers spend time analyzing changes trying to identify the source of the test failure, only to find out that the cause of the failure is test flakiness (TF). Many studies [1, 2, 3, 4] have been conducted to determine the root causes of test flakiness. These studies concluded that the main root cause of TF is the test smells. Test smells are poorly written test cases and their presence negatively affect the test suites and production code or even the software functionality [5]. Another definition is *"poor design or implementation choices applied by programmers or testers during the development of test cases"* [2] . Asynchronous wait, input/output calls, and test order dependency are some of the test smells that have been found to be the most common causes of TF [1]. The results presented by Luo et al. [1] were partially replicated by Palomba and Zaidman [2], leading to the conclusion that the most prominent causes of TF are test smells such as asynchronous wait, concurrency, and input output issues. There is strong evidence that the main reasons for test flakiness are specific test smells. Luo et al. suggested that *"developers should avoid specific test smells that lead to test flakiness"*. Authors in [2] investigated the question: *"To what extent can flaky tests be explained by the presence of test smells?"* They concluded that the *"cause of 54% of the flaky tests can be attributed to the characteristics of the co-occurring test smell"*.

Mapping test smells to flaky test resemble the problem of mapping words to spam/ham email. Certain words (i.e., sale, discount etc.) are more frequent in spam emails. Many studies [6, 7, 8, 9, 10, 11, 12, 13, 14, 15] have been conducted to predict email class (i.e., spam or ham) based on email contents. We adopted a similar approach in this study to determine the flakiness of test cases based on the test case code. Machine Learning approaches have been widely studied and there are lots of algorithms that can be used in e-mail classification including Naive Bayes [16][17], Support Vector Machines [18][19][15, 14], Neural Networks [20][21], K-nearest neighbor [22].

Recently, Pinto et al. evaluated five machine learning classifiers (Random Forest, Decision Tree, Naive Bayes, Support Vector Machine, and Nearest Neighbour) to generate flaky test vocabulary [23]. They concluded that Random Forest and Support Vector Machine provided best prediction of flaky tests. The investigated test cases were written in `Java` and the authors concluded that: *"future work will have to investigate to what extent their findings generalize to software written in other programming languages* [23].

In this study, we implemented supervised ML classifiers to detect if the test case is flaky or not based on the

8th International Workshop on Quantitative Approaches to Software Quality in conjunction with the 27th Asia-Pacific Software Engineering Conference (APSEC 2020) Singapore, 1st December 2020

azeem.ahmad@liu.se (A. Ahmad); ola.leifler@liu.se (O. Leifler); kristian.sandahl@liu.se (K. Sandahl)

0000-0003-3049-1261 (A. Ahmad)

contents of a test cases written in `Python`. We compared our findings with what was presented by Pinto et al. [23]. We looked for evidence if machine learning classifiers are applicable in predicting flaky tests and the results can be generalized to test cases written in other languages. In addition to this, our unique contribution is to investigate if test smells are good predictors of test flakiness. Through manual investigation of false positives and false negatives, we concluded a list of test smells that are strong and weak predictors of test flakiness. We investigated the following research questions in this study.

RQ1: What are the predictive accuracy of Naive Bayes, Support Vector Machine and Random Forest concerning flaky test detection and prediction?

RQ2: To what extent the predicting power of machine learning classifiers vary when applied on software written in other programming language?

RQ3: What can we learn about the predictive power of test smells using machine learning classifiers mentioned in RQ1?

2. Data Set Description and Prepossessing

We wrote a script to extract the contents of all test cases from open-source projects, mentioned in Table 1. After the test case content's extraction, we checked which of the test cases, in our database, has been mentioned in [24] as flaky. After this mapping, we finalized a database with the project name, test case name, test case content and a label. There are many keywords in the test case code that are irrelevant for the identification of test flakiness. We performed extensive data cleaning such as removing punctuation marks, digits and specific keywords (i.e., int, string, array, assert*) as well as converting text to lower case.

2.1. Classifiers:

An NBC, first proposed in 1998, is a probabilistic model which can determine the outcome (i.e., flaky or not flaky) of an instance (i.e., test case) based on the contents of its features (i.e., test case code). In our case, the outcome of NBC is binary. NBC is widely applied in classification and known to obtain excellent results. [25].

The attractive feature of SVM is that it eliminates the need for feature selections, which makes spam classification easy and faster [14]. SVM deals with the dual categories of classification and can find the best hyperplane to partition a sample space [15].

RF is an ensemble classification method (a technique that combines several base models to produce an optimal predictive model) suitable for handling problems that involve grouping data into different classes. RF predicts by using decision trees. Trees are constructed during training which can later be used for class prediction. There is a vote associated with each tree and once the class vote has been produced for all individual trees, the class with the highest vote is considered to be the output.

2.2. Performance Metrics and Parameters Tuning

To evaluate the predictive accuracy of classifiers, accuracy as the only performance indices is not sufficient [16]. We must consider precision, recall, F1-score, ROC curve, false positives and false negatives [16]. There is always some cost associated with false positives and false negatives. When a non flaky test wrongly classified as flaky, it gives rise to a some what insignificant problem, because an experienced user can bypass the warning by looking at test case code. In contrast, when a flaky test is wrongly classified as non flaky test, this is obnoxious, because it indicates the test suite still have test cases whose outcome cannot be trusted.

The experiment started with the implementation of simple NB without Laplace smoothing. The results did not provide good accuracy or precision, because without Laplace smoothing, the probability of appearing a rare test smell (i.e., test smell that was not in the training set) in the test set is set to 0, given the formula

$$\theta_j = n_{jc}/n_c$$

where the θ is the probability that an individual test smell is present in a flaky test, n_{jc} represents the number of times that particular test smell appeared in a test case and n_c represents the number of times that test smell appeared in any test case. Laplace smoothing refers to the modification in the equation:

$$\theta_j = (n_{jc} + \alpha)/(n_c + \alpha)$$

where we set the $\alpha = 1$ so that classifier adds 1 to the probability of rare test smells that were not present in the training set. Another step is to identify the threshold (i.e., 0.0 - 1.0) which will increase the predictive accuracy of the outcome. As far as SVM was concerned, although the feature data set space was linear, we decided to use both kernels (i.e., linear and poly) for the sake of experiment. For random forest, we used `ntree`

Table 1
Open-source project names provided by [24] with number of total test cases and flaky tests

Project Name	Total Number of TCs	Flaky Tests
apache-qpid-0.18	2357	284
hibernate 4	3231	273
apache-wicket-1.4.20	1250	216
apache-karaf-2.3	163	102
apache-struts 2.5	2346	60
apache-derby-10.9	3832	40
apache-lucene-solr-3.6	764	7
apache-cassandra-1.1	523	4
apache-nutch-1.4	7	4
apache-hbase-0.94	29	2
apache-hive-10.9	23	2
jfreechart-1.0.18	2292	0

between 300 - 700 as well as restricting number of variables available for splitting at each tree node known as `mtry` between 25 and 100.

3. Results

This section discusses the performance of NBL, SVM and RF with different parameters. We compared our results with the findings of Pinto et al. to discuss how results vary between Java and Python projects. We also discussed why some classifiers do not perform as expected and what can we learn about the predictive power of test smells for test flakiness detection and prediction.

3.1. RQ1: Performance of Naive Bayes Classifier, Support Vector Machine and Random Forest

Table 2 shows the 20 features with the highest information gain together with their frequency with respect to flaky and non-flaky tests. We assigned the features to the categories presented by Luo et al. in [1]. We manually traversed the code of flaky and non-flaky tests to understand the context and how features were used in the tests to assign categories. The top feature "conn" appeared in 1361 flaky tests and only 15 non-flaky tests. This feature is associated with external connection to input/output devices and lies under the category of "IO", presented by Luo et. al in [1]. The second top feature is "double" which appeared in 1190 flaky tests and 12 non-flaky tests assigned to the category of "IO" followed by "floating points operations". The top 3rd feature "tabl" was related to table creation during runtime for databases queries and appeared 1150 times in flaky tests and 52 times in non-flaky tests.

Figure 1 (A) represents the ROC curve [26] concerning NBC with Laplace smoothing denoted as NBL with different threshold (i.e., from 0.0 to 1.0). We conducted different experiments with different training and test data sets such as 50/50, 60/40, 70/30, 80/20 and 90/10. We found similar values for *k-fold* cross validation. ROC curve provides a comparison between sensitivity and specificity helping in organizing classifiers and visualizing their performance [26]. Sensitivity also known as the true positive rate represents a benefit of predicting flaky tests correctly and specificity also known as false positive rate represents the cost of predicting non flaky tests as flaky tests. In the case of false positive, developers need to spend effort and time, just to find out that this is a classifier mistake and the test case is not flaky. The optimal target, in the ROC curve, is to rise vertically from origin to the top left corner (higher true positive rate) as soon as possible because then the classifier can achieve all true positives with the cost of committing a few false positive. The diagonal line, in Figure 1 (A), represents the strategy of randomly guessing the outcome. Any classifier that appears in the lower right triangle performs worse than a random guessing and we can see that NBL lies in the upper left triangle. Looking at 1 (A), NBL with 70/30 data partition is suitable to proceed further with 0.4 probability score. NBL, as shown in 1 (A), has stopped issuing positive classification (i.e., flaky test prediction) around 0.76 - 0.87 threshold. After 0.87, it commits more false positive rate.

We tuned different parameters in NBL, SVM and RF before conducting further experiments. We do not intend to provide the results of all experiments because those experiments were only conducted to find the optimal parameters. The rest (i.e., simple NB, SVM with radial and sigmiod kernels) were not included in further experiments and discarded. Figure 1 (A-E) provides comparisons of NBL, SVM-Linear and SVM-Poly (i.e., different kernels) for accuracy, precision, recall and F1-score. All classifiers have achieved good accuracies ranging from 93% - 96%. NBL outperformed SVM although the difference between them is not dramatic. Looking only at the accuracy results of classifiers can be deceiving. The important factor for classifier selection is to ask the right question and motivate the choice of using specific classifier such as **are we interested in detecting flaky tests correctly (i.e., precision) or marking a non flaky test as flaky is not cost effective (i.e, recall)**. It is important to look at precision, recall and accuracy all together for classifier selection. We can assume that practitioners

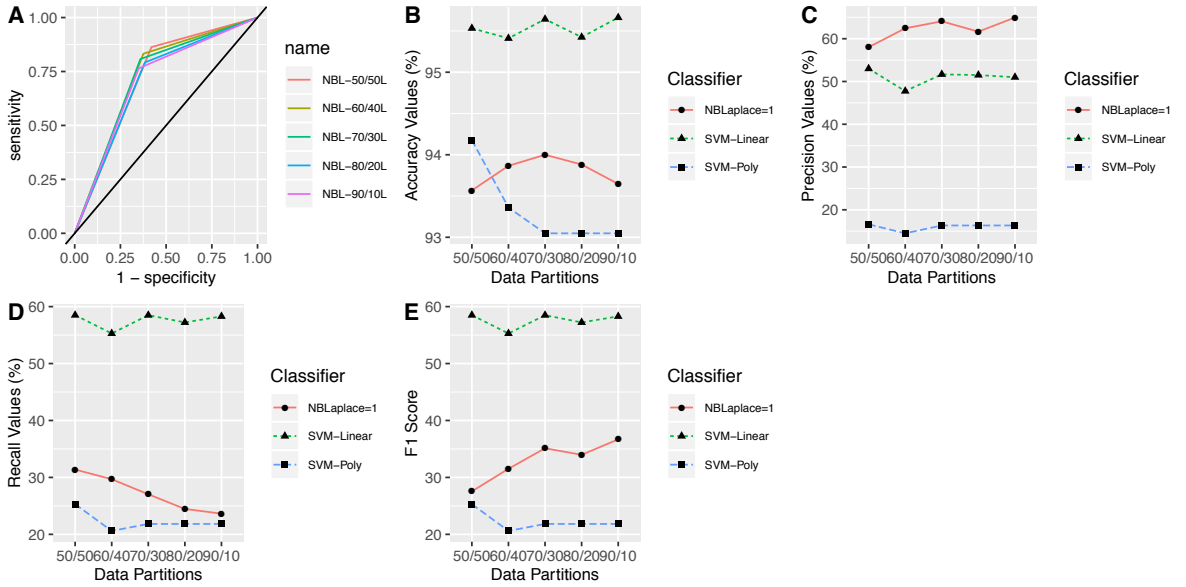

Figure 1: Performance comparison among classifiers. (A) represents the ROC curve of NBL classifier with different data partition and probability score. (B-E) represents the accuracy, precision, recall and F1-score of different classifiers with a different data partition, respectively.

are more interested in precision than recall because the test suite size, in many organizations, is very large and they cannot inspect all test cases. In this particular case, any classifier that correctly flag flaky tests will be encouraged. Precision can answer the question; ***"If the filter says this test case is flaky, what's the probability that it's flaky?"***. Figure 1 (C,D) provides precision and recall values for NBL and SVM. It can be noticed that NBL precision is increasing (in C) with the gradual decrease in recall (in D). NBL precision of 65% dictates that 35% of what was marked as flaky was not flaky. Recall is also lower in NBL as compared to SVM-Linear. SVM-Poly performs worst in terms of precision and recall as expected due the fact that the input data set is not polynomial and is well suited for image processing whereas linear kernel performs better for text classification.

F1-score, as presented in Figure 1 (E), is the harmonic mean of precision and recall. F1-score is useful and informative because of prevalent phenomenon of class imbalance in text classification [27]. NBL is a suitable candidate although it has a lower F1-score as compared to SVM-Linear because NBL performs better with short documents as in our case, the training test case consists of 6-15 lines of code [28]. NBL provides higher precision and lower recall as compared to SVM-linear. Another disadvantage of SVM is that it requires high computation and are very sensitive to noisy data [29].

Table 2
Top 20 frequented features and assigned category

Features	Frequency	Assigned category from Luo et. al [1]
new	28083	IO
assertequ	7967	-
null	4721	-
from	4719	-
string	4126	-
sclose	3315	IO
true	3154	-
select	2842	-
for	2809	Unordered Collection
fals	2604	-
not	2294	-
int	2111	-
asserttru	1910	-
tabl*	1677	IO
should	1596	-
doubl	1588	Floating point operations
valu	1429	-
expr	1322	-
tcommit	1313	IO
expcolnam	1211	IO

RF provides lesser classification error and better F1-scores as compared to decision trees, NBL and SVM.

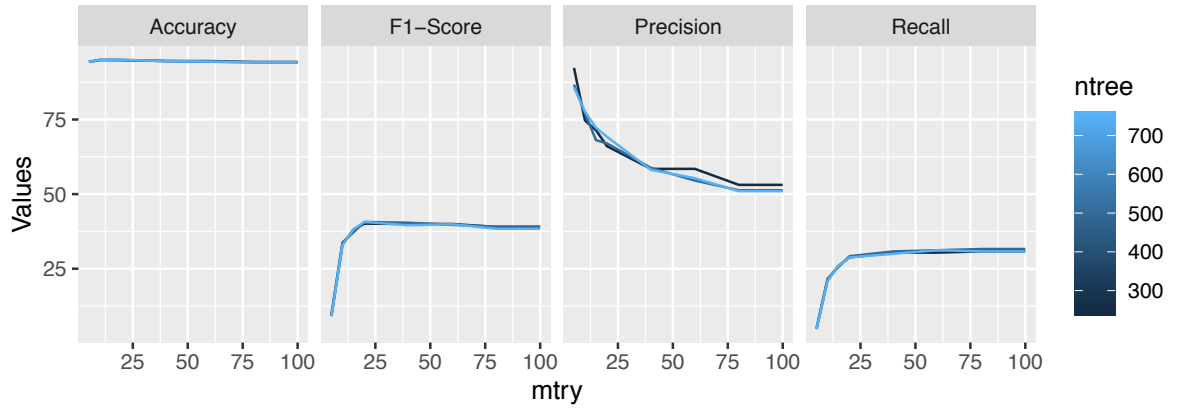

Figure 2: Performance of RF with different parameters (i.e., number of trees and mtry).

The precision, in which we are most interested, is usually better than that of SVM and NBL. Authors in [16] also concluded that RF performs better than NBL and SVM. The class outcomes are based on *"votes"* which are calculated by each tree in the forest. The outcome (i.e., flaky or not flaky) is selected based on the higher votes. Figure 2 presents the performance of RF with respect to selected metrics. *mtry* represents the number of variables randomly sampled as candidates at each split while *ntree* is the number of trees to grow. There is no way to find an optimal *mtry* and *ntree*, so we experimented with different settings, as shown in Figure 2. The *mtry* has a direct effect on precision and recall as shown in Figure 2. With an increase in *mtry*, the precision is decreasing and recall in increasing; an unwanted situation. The optimal value of *mtry* is 5 where precision is higher and recall is lower regardless of the number of trees. The change in *mtry* did not affect the accuracy but as we discussed earlier, we are not only interested in accuracy but precision too.

We performed several experiments to find optimal parameters within a classifier before comparing it to other classifiers. After these experiments, we identified three unique classifiers with unique and optimal parameters. Since, we are most interested in higher precision, we can see that RF with *mtry = 5* and *ntree=250* outperforms all other classifiers only for precision. RF has achieved more than 90% precision with less than 10% recall. We did not achieve high precision (i.e., >90%) in all classifiers. NBL provides unexpected results although it holds a good reputation in terms of detecting spam emails [29]. As compared to NBL and SVM, RF have distinct qualities such as 1) it can work with thousands of different input features without any feature deletion 2) it calculates approximation of important features for classification and 3) it is very robust to noise and outliers [30]. Caruana in [17] compared 10 different ML classifiers and concluded that decision trees and random forest outperform all other classifiers for spam classification.

3.2. RQ2: Predicting Power of ML Classifiers with Respect to Other Languages

In comparison of our findings with what was presented by Pinto et al. [23], we observed two differences. First, the top 20 frequented features are very different in both studies. Only one feature such as "tabl" marked as star (*) in Table 2 were similar in both the findings. However, we observed more features were related to "IO" output category, as presented in Table 2, which complemented the findings of Pinto et al. stating "that all projects manifesting flakiness are IO-intensive" [23]. Second, we have a very lower precision, recall and f1-score as compared to Pinto et al. except at a instance where random forest provided 0.92 precision. Table 3 provides detail statistics of precision, recall, and f1-score of three algorithms for comparison. The algorithms on `Python` language continuously performed worst contrary to what pinto et al. claimed: *"Although the studied projects are mostly written in Java, we do not expect major differences in the results if another object-oriented programming language is used instead, since some keywords maybe shared among them" [23].*

We speculate that there could be several reasons associated with these performance reduction such as (1)

Table 3
Comparison of Precision, Recall and F1-Score between our findings (A) and Pinto et al. (B)

	Precision		Recall		F1-Score		Diff
Algo.	A	B	A	B	A	B	
Random Forest	0.92	0.99	0.4	0.91	0.09	0.95	↘
Naive Bayes	0.62	0.93	0.15	0.8	0.24	0.86	↘
Support Vector	0.51	0.93	0.61	0.92	0.57	0.93	↘

We implemented the code ourselves using R libraries for aforementioned classifiers whereas pinto et al. used Weka [31] which is an open source machine learning software that can be accessed through a graphical user interface, standard terminal applications [32], (2) Number of features were very high in the training samples and in these cases other models should be considered (i.e., regularized linear regression) that might performed better, (3) the versatility offered by parameter tunning can become problematic and require special considerations that can impact the classifiers, etc.

3.3. RQ3: Test Smells Analysis and their Predictive Power for Test Flakiness Detection and Prediction

We investigated manually different cases of true positives (i.e., correct flaky test prediction), false positive (i.e., flaky test cases marked as non flaky) and false negative (i.e., non flaky test cases marked as flaky) and true negatives (correct non flaky test prediction) to answer RQ3. We observed that it is not only the frequency of test smell that makes a test case flaky but its co-existence with the class code or external factors such as operating systems or specific product. For example, The test smell *'Conditional Test Logic'* as mentioned in [3] refers to nested and complex *'if-else'* structure in the test case. Depending on which branch of *'if-else'* is executed, the system under test may require specific environment settings. Failing to set the environment, during different executions, will flip the test case outcome, thus making it flaky.

After manual investigation of all true/false positives and true/false negatives, we come up with a list of test smells that are strong or weak predictors of test flakiness, as shown in Table 4. Strong predictors refer to those test smells that existed in true positives and true negatives cases whereas weak predictors only existed in false negatives and false positives. Test smells that are classified as weak predictors in this study are still useful and can help in identification of test flakiness, but they are not useful with machine learning classifiers because they require additional information such as what operating system they are running on and whether or not specific configurations should be deployed. Test smells that are classified as strong predictors are very useful with machine learning classifiers because they only exist in test case function as one unit and do not require additional information.

4. Lesson Learned

ML and AI algorithms in recent years have established a good reputation for predicting diseases based on symptoms, spam emails based on email contents and many more. We believe that given a proper input data set which clearly distinguishes between flaky and non flaky tests, ML and AI can provide high prediction capabilities saving effort, time and resources. We strongly believe that practitioners, during training of data set, should not consider complete test cases as an input but only the test codes (i.e., only few lines) that reveal test flakiness.

It is inconclusive that predicting power of machine learning vary with respect to software written in another languages. Investigation on `Java` test cases [23] revealed good results while findings for `Python` test cases performed unexpected, thus requiring more investigations whether lexical information can be traced to flakiness.

Async wait, precision, randomness and IO test smells are string predictors can be predicted by machine learning classifiers with 100% precision because they only exist in test case code and do not require additional information from test class or operating system. Whereas all other test smells mentioned in Table 4 are weak predictors of test flakiness and require additional sources of information. We are only aware of test smells that are investigated in open-source repositories and literature on test smells in closed-source software is scarce.

5. Discussion and Implication

Valuable Indicators for Testers These classifiers can increase the awareness about flaky test vocabulary among testers. When a new test is added to a test suite, it will be easy to identify whether this test case contains specific test smells that were known to increase test flakiness during previous executions. Testers can take advantage of these types of information to reduce test flakiness. Testers can easily identify test smells that are independent of their environment with the help of Table 4.

Precision Depends on Data Set: In the literature

Table 4
Test Smells as Strong and Weak Predictors Together with Source of their Existence

Test Smell Category	Prediction Category	Test Case	Test Class	Operating System	External Libraries	Hardware/Product
Async wait	Strong	[✓]	-	-	-	-
Precision (float operations)	Strong	[✓]	-	-	-	-
Randomness	Strong	[✓]	-	-	-	-
IO	Strong	[✓]	-	-	-	-
Unordered Collection	Weak	[✓]	[✓]	[✓]	-	-
Time	Weak	[✓]	[✓]	[✓]	-	[✓]
Platform	Weak	[✓]	[✓]	[✓]	[✓]	-
Concurrency	Weak	[✓]	[✓]	[✓]	-	-
Test order dependency	Weak	[✓]	[✓]	[✓]	[✓]	[✓]
Resource Leak	Weak	[✓]	[✓]	[✓]	-	-

of ML, particularly with spam detection, it is acknowledged that precision is a function of the combination of the classifier and the data set under investigation. Classifier's precision, in isolation of data set, does not make sense. The right question is ***"how precise a classifier is for a given data set"***. Unfortunately, there is no data available that provides test case contents and an associated label thus, limiting the use of advanced ML and AI algorithms. In addition to lack of flaky test data, all research has been conducted with open-source software and we know a little about what test smells are present in closed-source software. Ahmad et. al. concluded that there are specific test smells that are associated with the nature of the product [33] known as *'company-specific'* test smells. The classifier which are trained on a specific data set or a domain cannot be generalized to be used with another data set or domain. There is a long road ahead to explore the best classifier given different data sets.

Beyond Static Analysis of Test Smells and their Frequency: ML is capable of incorporating different sources of information to increase predictive accuracy as compared to the limited experiment in this study where we only utilized the frequency of test smells in the test case. During the investigation of the cases of *'false negative'* and *'false positive'*, it has been observed that the frequency of test smells in the test case will not be sufficient for prediction. Some test case code (i.e., seeds()) will cancel the effect of test smell (i.e., random()), no matter how frequent the random() function appears in the test case. Some test smell, even with single appearance, will weight more than a test smell for higher frequency.

Precision Vs Recall: When a test suite grows in size, developers would like any indications of tests that are more likely to be flaky rather than adopting an approach of re-run which of-course is not cost effective in terms of time and resources. Developers like to increase precision at the expense of recall. When encountering *'false negative'*, an experienced developer, having sufficient knowledge of the test smells, will bypass the outcome, however, with *'false positive'*, developers are unaware of the fact that test suite still contains flaky tests. The motivation of employing ML classifiers (i.e., higher precision - low recall vs balances precision and recall) should be made clear before proceeding with implementation.

Multi-Factor Input Criteria for Flaky Test Detection: We observed that the ML algorithm should include different sources of information to increase predictive accuracy. These sources may include 1) assigning specific weight (i.e., in numbers) to specific test smells or test code, 2) developer's experience (i.e., new developer, unaware of the test design guidelines are more likely to write flaky tests), 3) company-specific test smells.

6. Related Work

Luo et al., in [1], investigated 52 open-source projects and 201 commits and categorized the causes of test case. Asynchronous wait (45%), concurrency (20%), and test order dependency (12%) were found to be the most common causes of TF. Palomba and Zaidman in [2] partially replicated the results presented by Luo et al. concluding that the most prominent causes of TF are asynchronous wait, concurrency, and input output and network issues. Authors investigated, in [3], the relationship between smells and TF. Another empirical study of the root causes of TF in Android Apps was conducted by Thorve et al. [4] by analyzing the commits of 51 Apache open-source projects. Thorve et al. [4] complement the results of Luo et al. and Palomba and Zaidman, but they also report two additional test smells (user interface and program logic) that are re-

lated to TF in Android Apps. Bell et al. in [34] and proposed a new technique called DeFlaker, which monitors the latest code coverage and marks the test case as flaky if the test case does not execute any of the changes. Another technique called PRADET [35] does not detect flaky tests directly, rather it uses a systematic process to detect problematic test order dependencies. These test order dependencies can lead to flakiness. King et al. in [36] present an approach that leverages Bayesian networks for flaky test classification and prediction. This approach considers flakiness as a decease mitigated by analyzing the symptoms and possible causes. Teams using this technique improved CI pipeline stability by as much as 60%. To best of our knowledge, no study has been conducted to evaluate the predictive accuracy of machine learning classifiers that can help developers in flaky test case prediction and detection.

Dutta et al. [37] and Sjobom [38] investigated projects written in `Python` language to classify test smells that increase test flakiness. Their study is limited to list the test smells and their effect on test flakiness. Our study worked with the test smells identified in [38]. Pinto et al. evaluated five machine learning classifiers (Random Forest, Decision Tree, Naive Bayes, Support Vector Machine, and Nearest Neighbour) to generate flaky test vocabulary written in `Java` [23]. The concluded that Random Forest and SVM performed very well with high precision and recall. They concluded that features such as "job", "action", and "services" were commonly associated with flaky tests. We replicated the similar experiment with different programming language and extended the current knowledge by answering RQ2 and RQ3.

7. Validity Threats

The authors in this study selected only those ML classifiers which have established a good reputation of high accuracy in spam detection thus reducing the selection bias.

The authors in this study reduced the experimenter bias by performing several experiments with different thresholds (i.e., probability scores, kernels, number of trees, etc.) before selecting a champion.

External validity refers to the possibility of generalizing the findings, as well as the extent to which the findings are of interest to other researchers and practitioners beyond those associated with the specific case being investigated. Since the precision strongly depends on the data set under investigation, we have an external validity threat. We cannot generalize the findings of this study for other data set.

8. Conclusion

At the moment of writing this paper, literature is scarce on test flakiness (i.e., root causes, challenges, mitigation strategies, etc.) which requires significant attention from researchers and practitioners. We extracted flaky and non flaky test case contents from open source repositories. We implemented three ML classifiers such as Naive Bayes, Support Vector Machine and Random Forest to see if the predictive accuracy can be increased. The authors concluded that only RF performs better when it comes to precision (i.e., > 90%) but the recall is very low (< 10%) as compared to NBL (i.e., precision < 70% and recall >30%) and SVM (i.e., precision < 70% and recall >60%). The authors concluded that predicting accuracy of ML classifiers are strongly associated with the lexical information of test cases (i.e., test cases written in `Java` or `Python`). The authors investigated why other classifiers failed to produce expected results and concluded that; 1) it is a combination of the test smell and an external environment that makes a test case flaky, and in this study, the external environment was not taken into consideration, 2) ML classifiers should not only consider the frequency of test smells in the test case but other important test codes that have an ability to cancel the effect of test smells.

9. Acknowledgment

We appreciate Linköping University students to provide their expertise to collect flaky test data from online repositories.

References

[1] Q. Luo, F. Hariri, L. Eloussi, D. Marinov, An Empirical Analysis of Flaky Tests, in: Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, ACM, New York, NY, USA, 2014, pp. 643–653. URL: http://doi.acm.org/10.1145/2635868.2635920. doi:10.1145/2635868.2635920, event-place: Hong Kong, China.

[2] F. Palomba, A. Zaidman, Does Refactoring of Test Smells Induce Fixing Flaky Tests?, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 1–12. doi:10.1109/ICSME.2017.12.

[3] F. Palomba, A. Zaidman, The smell of fear: on the relation between test smells and flaky tests, Empirical Software Engineering 24 (2019) 2907–2946. URL: https://doi.org/10.1007/s10664-019-09683-z. doi:10.1007/s10664-019-09683-z.

[4] S. Thorve, C. Sreshtha, N. Meng, An Empirical Study of Flaky Tests in Android Apps, in: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018, pp. 534–538. doi:10.1109/ICSME.2018.00062.

[5] V. Garousi, B. Küçük, Smells in software test code: A survey of knowledge in industry and academia, Journal of Systems and Software 138 (2018) 52–81. URL: http://www.sciencedirect.com/science/article/pii/S0164121217303060. doi:10.1016/j.jss.2017.12.013.

[6] R. Shams, R. E. Mercer, Classifying Spam Emails Using Text and Readability Features, in: 2013 IEEE 13th International Conference on Data Mining, 2013, pp. 657–666. doi:10.1109/ICDM.2013.131, iSSN: 2374-8486.

[7] S. K. Tuteja, N. Bogiri, Email Spam filtering using BPNN classification algorithm, in: 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT), 2016, pp. 915–919. doi:10.1109/ICACDOT.2016.7877720, iSSN: null.

[8] E. Sahın, M. Aydos, F. Orhan, Spam/ham e-mail classification using machine learning methods based on bag of words technique, in: 2018 26th Signal Processing and Communications Applications Conference (SIU), 2018, pp. 1–4. doi:10.1109/SIU.2018.8404347, iSSN: null.

[9] K. Mathew, B. Issac, Intelligent spam classification for mobile text message, in: Proceedings of 2011 International Conference on Computer Science and Network Technology, volume 1, 2011, pp. 101–105. doi:10.1109/ICCSNT.2011.6181918, iSSN: null.

[10] A. B. M. S. Ali, Y. Xiang, Spam Classification Using Adaptive Boosting Algorithm, in: 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), 2007, pp. 972–976. doi:10.1109/ICIS.2007.170, iSSN: null.

[11] R. K. Yin, Case study research design and methods, 4th ed ed., Thousand Oaks, Calif Sage Publications, 2009. URL: https://trove.nla.gov.au/work/11329910.

[12] A. A. Alurkar, S. B. Ranade, S. V. Joshi, S. S. Ranade, P. A. Sonewar, P. N. Mahalle, A. V. Deshpande, A proposed data science approach for email spam classification using machine learning techniques, in: 2017 Internet of Things Business Models, Users, and Networks, 2017, pp. 1–5. doi:10.1109/CTTE.2017.8260935, iSSN: null.

[13] S. Vahora, M. Hasan, R. Lakhani, Novel approach: Naïve Bayes with Vector space model for spam classification, in: 2011 Nirma University International Conference on Engineering, 2011, pp. 1–5. doi:10.1109/NUiConE.2011.6153245, iSSN: 2375-1282.

[14] M. R. Islam, W. Zhou, M. U. Choudhury, Dynamic Feature Selection for Spam Filtering Using Support Vector Machine, in: 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), 2007, pp. 757–762. doi:10.1109/ICIS.2007.92, iSSN: null.

[15] T.-Y. Yu, W.-C. Hsu, E-mail Spam Filtering Using Support Vector Machines with Selection of Kernel Function Parameters, in: 2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC), 2009, pp. 764–767. doi:10.1109/ICICIC.2009.184, iSSN: null.

[16] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, O. E. Ajibuwa, Machine learning for email spam filtering: review, approaches and open research problems, Heliyon 5 (2019) e01802. URL: http://www.sciencedirect.com/science/article/pii/S2405844018353404. doi:10.1016/j.heliyon.2019.e01802.

[17] R. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in: Proceedings of the 23rd international conference on Machine learning, ICML '06, Association for Computing Machinery, Pittsburgh, Pennsylvania, USA, 2006, pp. 161–168. URL: https://doi.org/10.1145/1143844.1143865. doi:10.1145/1143844.1143865.

[18] C.-Y. Chiu, Y.-T. Huang, Integration of Support Vector Machine with Naïve Bayesian Classifier for Spam Classification, in: Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007), volume 1, 2007, pp. 618–622. doi:10.1109/FSKD.2007.366, iSSN: null.

[19] Z. Jia, W. Li, W. Gao, Y. Xia, Research on Web Spam Detection Based on Support Vector Machine, in: 2012 International Conference on Communication Systems and Network Technologies, 2012, pp. 517–520. doi:10.1109/CSNT.2012.117, iSSN: null.

[20] A. S. Kataev, L. Y. Emaletdinova, D. V. Kataseva, Neural Network Spam Filtering Technology, in: 2018 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 2018, pp. 1–5. doi:10.1109/ICIEAM.2018.8728862, iSSN: null.

[21] M. K., R. Kumar, Spam Mail Classification Using Combined Approach of Bayesian and Neural Network, in: 2010 International Conference on Computational Intelligence and Communication Networks, 2010, pp. 145–149. doi:10.1109/CICN.2010.39, iSSN: null.

[22] L. Firte, C. Lemnaru, R. Potolea, Spam detection filter using KNN algorithm and resampling, in: Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing, 2010, pp. 27–33. doi:10.1109/ICCP.2010.5606466, iSSN: null.

[23] G. Pinto, B. Miranda, S. Dissanayake, What is the Vocabulary of Flaky Tests? (2020) 11.

[24] W. Lam, R. Oei, A. Shi, D. Marinov, T. Xie, iDFlakies: A Framework for Detecting and Partially Classifying Flaky Tests, in: 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), 2019, pp. 312–322. doi:10.1109/ICST.2019.00038, iSSN: 2159-4848.

[25] M. Sasaki, H. Shinnou, Spam detection using text clustering, in: 2005 International Conference on Cyberworlds (CW'05), 2005, pp. 4 pp.–319. doi:10.1109/CW.2005.83, iSSN: null.

[26] T. Fawcett, An introduction to ROC analysis, Pattern Recognition Letters 27 (2006) 861–874. URL: http://www.sciencedirect.com/science/article/pii/S016786550500303X. doi:10.1016/j.patrec.2005.10.010.

[27] D. Zhang, J. Wang, X. Zhao, Estimating the Uncertainty of Average F1 Scores, in: Proceedings of the 2015 International Conference on The Theory of Information Retrieval, ICTIR '15, Association for Computing Machinery, Northampton, Massachusetts, USA, 2015, pp. 317–320. URL: https://doi.org/10.1145/2808194.2809488. doi:10.1145/2808194.2809488.

[28] Wang, Baselines and bigrams | Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2, ???? URL: https://dl-acm-org.e.bibl.liu.se/doi/10.5555/2390665.2390688.

[29] S. Abu-Nimeh, D. Nappa, X. Wang, S. Nair, A comparison of machine learning techniques for phishing detection, in: Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit on - eCrime '07, ACM Press, Pittsburgh, Pennsylvania, 2007, pp. 60–69. URL: http://portal.acm.org/citation.cfm?doid=1299015.1299021. doi:10.1145/1299015.1299021.

[30] L. Breiman, Random Forests, Machine Learning 45 (2001) 5–32. URL: https://doi.org/10.1023/A:1010933404324. doi:10.1023/A:1010933404324.

[31] I. H. Witten, E. Frank, Data mining: practical machine learning tools and techniques with Java implementations, ACM SIGMOD Record 31 (2002) 76–77. URL: https://doi.org/10.1145/507338.507355. doi:10.1145/507338.507355.

[32] Weka 3 - Data Mining with Open Source Machine Learning

Software in Java, ???? URL: https://www.cs.waikato.ac.nz/ml/weka/index.html.

[33] A. Ahmad, O. Leifler, K. Sandahl, Empirical Analysis of Factors and their Effect on Test Flakiness - Practitioners' Perceptions, arXiv:1906.00673 [cs] (2019). URL: http://arxiv.org/abs/1906.00673, arXiv: 1906.00673.

[34] J. Bell, O. Legunsen, M. Hilton, L. Eloussi, T. Yung, D. Marinov, DeFlaker: Automatically Detecting Flaky Tests, in: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), 2018, pp. 433–444. doi:10.1145/3180155.3180164.

[35] A. Gambi, J. Bell, A. Zeller, Practical Test Dependency Detection, in: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), 2018, pp. 1–11. doi:10.1109/ICST.2018.00011.

[36] T. M. King, D. Santiago, J. Phillips, P. J. Clarke, Towards a Bayesian Network Model for Predicting Flaky Automated Tests, in: 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), IEEE Comput. Soc, Lisbon, 2018, pp. 100–107. doi:10.1109/QRS-C.2018.00031.

[37] S. Dutta, A. Shi, R. Choudhary, Z. Zhang, A. Jain, S. Misailovic, Detecting flaky tests in probabilistic and machine learning applications, in: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2020, Association for Computing Machinery, New York, NY, USA, 2020, pp. 211–224. URL: https://doi.org/10.1145/3395363.3397366. doi:10.1145/3395363.3397366.

[38] A. Sjöbom, Studying Test Flakiness in Python Projects : Original Findings for Machine Learning, 2019. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-264459.