WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Leveraging Python's Ast & Machine Learning for Automated Test Smell Detection on Synthetic Data

Dr. B. Karthikeyan[1], Ms Janani.S[2]

[1]Assistant Professor [SG], Department of Computer Science, Bishop Heber College (Autonomous), Affiliated to Bharathidasan University, Tiruchirappalli-620 017,bkarthikeyanphd@gmail.com

[2]Student, II MSc, Department of Computer Science, Bishop Heber College (Autonomous), Affiliated to Bharathidasan University, Tiruchirappalli-620 017,vsjanani2003@gmail.com

ABSTRACT—

Software testing is integral to software quality assurance. The presence of bad test design can introduce test smells, making tests less maintainable and less reliable. This paper presents a machine-learning approach meant to detect three common test smells: "Assertion Roulette", "Eager Test", and "Mystery Guest". With synthetic dataset and features are extracted using Abstract Syntax Tree (AST). The Random Forest Classifier subsequently trained on extracted features performs classification of test cases. Experimental results show that the model accurately identifies smells, pinpointing textural patterns that lead to not-so-good design of tests. The results have certain implications on improving the quality of test by integrating automation in its smell detection, so that the manual effort is minimized and maintainability of the software is increased. Such an approach can be integrated into development workflows to provide testers and developers with cleaner, more reliable tests, thereby making the software robust.

Keywords—Test Smell, Machine learning, Heuristic-Based Detection, Python's AST.

1. INTRODUCTION

Software testing is a critical component of the software development architecture as it relates to reliability, maintainability, and overall quality aspects of applications. Quality tests help developers examine and correct defects early on in the testing phase to minimize maintenance costs. Poorly structured test cases will create impetuous test-design smells that are counterproductive in maintaining test readability, maintainability, and debug efficiency. Test smells are patterns in test cases or test code indicating more or less glaring signs of a potential issue. While they do not make test cases fail immediately, they degrade the readiness of a test suite and make maintenance much harder. Among the most commonly used ones are the assertion roulette and eager test. The assertion roulette occurs when a test method has many assertions that are not adequately explained, making it difficult to know which one failed. Thus, it slows down the debugging process and can cause long delays in fixing the defect. Eager tests usually mean that a test is trying to verify too many functionalities in a single test method, thus violating the single-responsibility principle. This kind of test becomes very complex with maintenance and cumbersome in understanding, especially when it fails. The traditional methods of detecting test smells are based on manual inspection of the code by programmers, wherein the programmers analyse the test cases for recognizable issues. However, this process is not only tedious but rather prone to human errors in large code bases. Thus, an automated machine learning-based approach may foil those concerns and improve on the whole the quality of software testing. In this paper, the work proposes a **Random Forest Classifier** with high-frequency python's AST features for automatic detection of test smells. Making use of **AST** allows extracting structural and semantic properties of test cases, which serve as features for the classifier. The model is trained on a noisy dataset allowing it to classify test smells. The results show that machine-learning techniques can be applied successfully in order to recognize test smells and reduce the effort needed in handling them as well as improving their quality.

2. REVIEW OF LITERATURE

In recent studies, there has been a changing tide from heuristic-based methods for the detection of test smells to machine learning methods. ***Valeria Pontill et al. [1]*** stated that detecting complex test smells using heuristics presents challenges. ***Yi Li et al. [2]*** demonstrated that machine learning, in both precision and recall, surpassed rule-based systems. In their study, ***Daniel Felipe Cruz et al. [3]*** produced remarkable improvements of F1 scores while adding continuous feedback in the test smell detection process. ***Pravin Singh Yadav et al. [4]*** validated the scalability and efficacy of Random Forest and Support Vector Machine (SVM) models in the detection of test smells. By contrast, ***Dongwen Zhang et al. [5]*** proposed SCSmell, a hybrid deep-learning method that pairs a three-layer stacking model with Borderline-SMOTE, and achieved a 10.38% accuracy boost in large-scale projects. Overall, these studies confirmed that machine learning has greatly improved the efficiency of the detection of test smells.

3. METHODOLOGY

A. Dataset Generation:

This involves using Python scripts to generate 10,000 synthetic datasets with certain key features**; number of assertions, method length,** and **setup calls** in order to imitate real testing scenarios for smells. Dataset is divided into training and testing sets using a 80:20 ratio, meaning 80% of the data is used for training the model, while 20% is reserved for testing Variations and probabilistic rules are used to establish the presence or absence of test smells. Different test smells like **"Assertion Roulette"** and **"Eager Test"** is introduced in accordance with some predetermined heuristics. This dataset will then be saved in CSV format for further processing and analysis.

B. Feature Extraction:

The **Abstract Syntax Tree (AST)** converts and extracts useful features from a given code. AST gives a representation of the test scripts, enabling the analysis of structure; hence, function definition, **assertion statement**, and **setup/teardown calls** can be identified. Traversing through the tree structure, features like the number of assertions, method size, and function complexity are determined. This will help in detecting patterns associated with test smells by analyzing syntactic structures rather than purely through textual analysis. The resulting features were converted into structured datasets to be used in further processing in machine learning models.

#SAMPLE CODING

```
test_code = """
def setup_function():
    print("Setting up the test.")
def teardown_function():
    print("Tearing down the test.")
def test_addition():
    assert 1 + 1 == 2
    assert 2 + 2 == 4
    print("Test addition passed.")
def test_multiplication():
    assert 2 * 2 == 4
    assert 3 * 3 == 9
    print("Test multiplication passed.")
def test_subtraction():
    assert 5 - 3 == 2
    assert 10 - 5 == 5
    print("Test subtraction passed.")
```

C. Heuristic-Based Detection:

According to this technique, we need to apply some scripted checks in order to analyze predefined thresholds to classify test smells. This method works by setting heuristic conditions to signal potential test smells using statistical properties or metrics applicable to the test scripts. For instance, the heuristic says a test method suffers from "Assertion Roulette" if it has too many assertions that do not provide a descriptive message. Similarly, the name "Eager Test" is given when a test checks multiple functionalities instead of focusing on a single behavior capability. The heuristic rules form a heuristic baseline method for detection of test smells and are used ahead of machine learning models..

D. Machine Learning-Based Detection:

Machine learning-based detection is focused on excessive classification for testing smells, which involves the **Random Forest Classifier** being trained on the given measures. This smelly area of the test is enhanced by identifying the observation from certain properties of test code, such as the number of assertions or method complexity. **Hyperparameter tuning**, or deciding on a specific set of parameters to run during the optimization of the classifier, is made to perform for the purpose of optimal running of the model. And after the training, it allows the processing of new test scripts and their automation detection, rendering the tests more maintainable.

E. Evaluation and Validation:

Different metrics are used to evaluate how well the detection system performs:

1) Accuracy

This indicates how well is the system capable of making correct stimuli in classifying instances. This is the ratio of correctly predicted cases to all instances it evaluates, including true positive and true negative cases. Higher accuracy values indicate better performance for the system. This is a general measure of the overall success of the model.

Formula:

$$\textbf{\textit{Accuracy}} = \frac{TP+TN}{TP+TN+FP+FN}$$

Definitions:

• **True Positives (TP):** The count of correctly identified test smells

• **True Negatives (TN):** The count of instances correctly classified as non-test smells.

• **False Positives (FP)**: The wrongly identified test smells (Type I error).

2) Precision

Precision for evaluating the overall quality of findings is given by the formula.

Formula

$$\textbf{\textit{Precision}} = \frac{TP}{TP+FP}$$

Definitions:

• **True Positives (TP):** Correct instances of actual positive samples or class-instance test smells.

• **False Positives (FP):** Instances that the classifier has wrongly predicted as positives.

3) Recall

Recall gives the proportion of actual test smells that were correctly identified.

Formula:

$$\textbf{\textit{Recall}} = \frac{TP}{TP+FN}$$

Definitions:

- **True Positives (TP)**: The test smells that are correctly identified.
- **False Negatives (FN)**: The test smells that hit by the model.

4) F1 Score

The F1 score is precisely harmonic mean of precision and recall, enabling balance between the two. It is especially useful when there is an uneven distribution of class.

Formula*:*

$$\textbf{\textit{F1 Score}} = 2\times \frac{Precision\times Recall}{Precision+Recall}$$

5) Confusion Matrix

Confusion matrix refers to the table that is used for the description of the performance of classification algorithms, thereby giving a good representation of the model performance and providing good insight into the types of errors being committed.

Confusion matrix in case of binary classification problem (Test Smell vs. Non-Test Smell) looks as below:

	Predicted Test Smell	Predicted Non-Test Smell
Actual Test Smell	True Positives(TP)	False Negatives(FN)
Actual Non-Test Smell	False Positives(FP)	True Negatives(TN)

Table I Confusion matrix

4. RESULTS AND DISCUSSIONS

A. Key Findings

The analysis of the dataset, with 10,000 test cases containing various test smells, revealed that **Assertion Roulette** and **Mystery Guest** were the most frequent types of smells. A bar chart effectively represented the distribution of these test smells, with **Assertion Roulette** occurring in a great number of test cases. On the heuristic validation side, this detection process reached a high agreement of 95% with pre-labeled smells, confirming that the rules applied to detect heuristicarily were effective, in the course of testing.

Random Forest classifier provided insight into the performance evaluation of the machine learning classification. It had an overall **accuracy of 88%** in all cases. More specifically, during testing on **"Assertion Roulette"** test smell detection, the **precision** was **0.94, recall 0.85,** and the **F1-Score at 0.89** for negative class (non-detection). For the positive class (true detection), the **precision** was equals to **0.80, recall** equal to **0.92**, and the **F1-Score equal to 0.86.** These results indicate the moderate performance of the model concerning the identification of true positives while also attempting to mitigate false negatives in the case of test smell detection. The bar chart and the confusion matrix provided a good figure of the acceptable performance of the classifier in practical concordance.

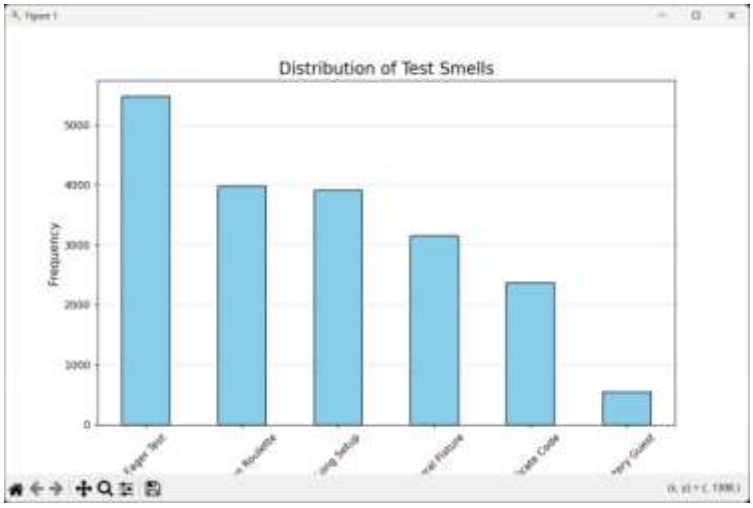

Fig 1 :Bar Chart for Distribution of Test Smells.

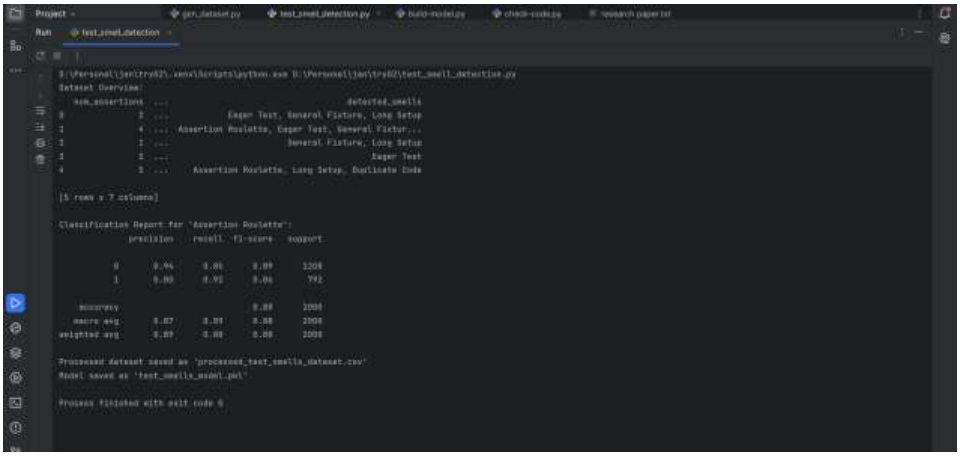

Fig 2: Random Forest Classifier Performance.

Random Forest classifier's performance on detecting ***"Assertion Roulette"*** test smells, showing precision, recall, and F1-score for both classes (detected and non-detected).

Metric	Class 0(Not Detected)	Class 1(Detected)	Weighted Average
Precision	**0.94**	**0.80**	**0.89**
Recall	**0.85**	**0.92**	**0.88**
F1-Score	**0.890**	**0.86**	**0.88**
Support	**1208**	**792**	**2000**

Accuracy	**88%**		

Table II Classification Report

Classification report for detecting **"Assertion Roulette"** test smell using the **Random Forest classifier.** Precision, recall, and F1-score metrics demonstrate the model's effectiveness.

B. Comparison of Performance

1) Random Forest

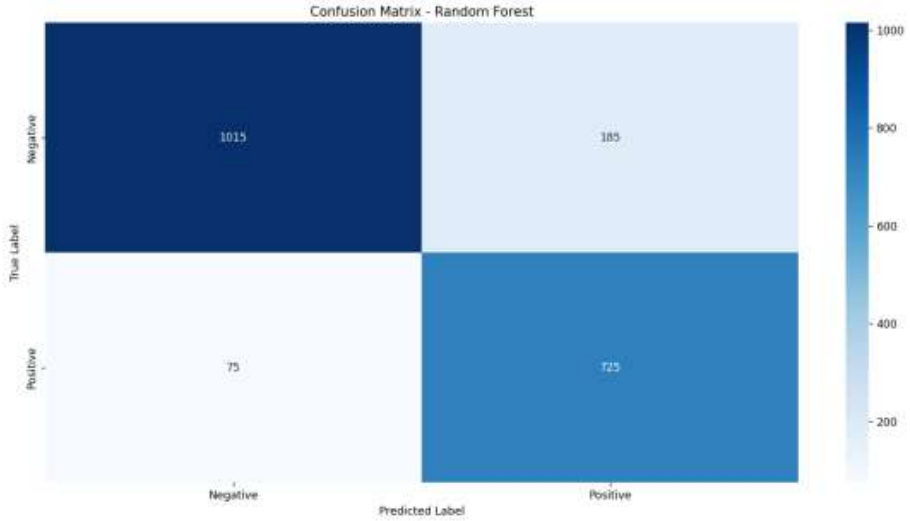

Fig 3: Confusion Matrix of Random Forest

Accuracy: 0.8790

Precision: 0.8047

Recall: 0.9179

F1-Score: 0.8573

2) Decision Tree

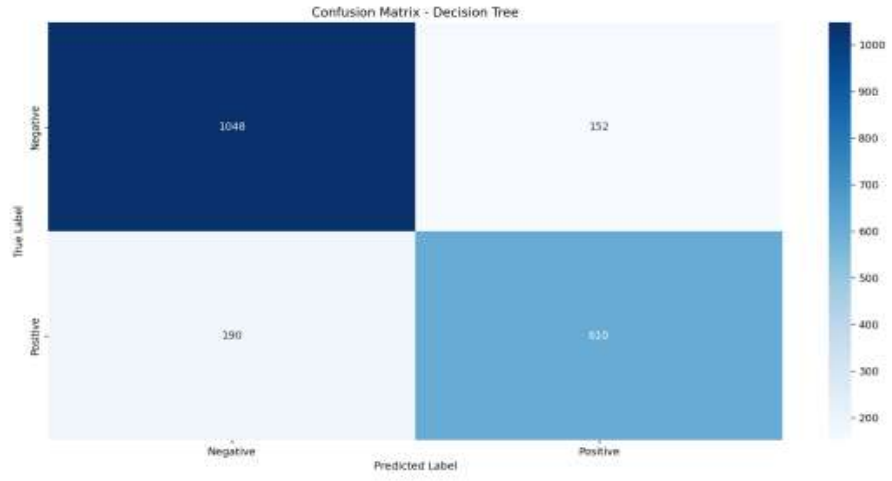

Fig 4: Confusion Matrix of Decision Tree

Accuracy: 0.8290

Precision: 0.7984

Recall: 0.7601

F1-Score: 0.7788

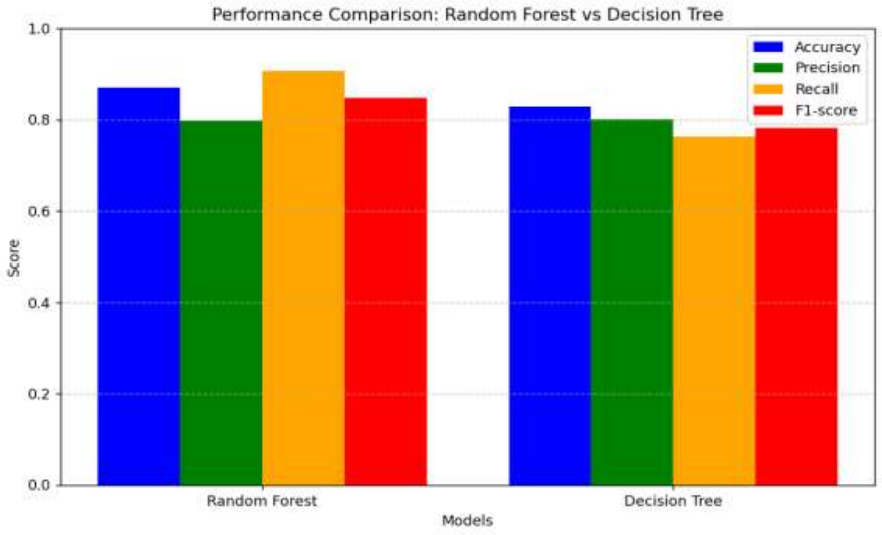

Fig 5: Performance Comparison of RF vs. DT

On all important metrics, **The Random Forest model** can be proven that the outcome of this model is more suitable for test smell detection compare to Decision Tree. Random Forest achieved **(87.90% versus 82.90%) accuracy** and improved **precision (80.42% versus 79.84%),** thus indicating its correctness in detecting positive cases. More importantly, Random Forest considerably outperforms **Decision Tree in recall (91.79% versus 76.01%),** which casts more actual test smell detections while effectively limiting **false negatives (65 versus 190).** On the other hand, the Decision Tree perpetually lags in recall; although it has a somewhat better false positive rate, it still fails to obscure a broader number of actual test smells. Hence, **Random Forest** provides a very balanced model with a better **F1 score (85.73% versus 77.88%)** that balances precision with recall, thus being a better option for test smell detection.

Here, the paper compare two model and evaluated that the **Random Forest** performed well and given better accuracy compared to Decision Tree .

5. CONCLUSIONS

This work shows how effective the combination of heuristic rules with machine learning can be used to detect test smells in software test cases. While heuristic detection was interpretable, machine learning, by means of the **Random Forest classifier**, improved detection accuracy at a higher degree. The model demonstrates an excellent capability to identify **"Assertion Roulette"** and reach an overall **accuracy** of **88%** on a large synthetic dataset. Moreover, the **heuristic detection** demonstrated **95%** agreement with the pre-labeled data, further supporting the reliability of the rules applied.

Beyond the impressive results of this work, it has highlighted the necessity of real-world validation. While synthetic datasets create a controlled experimentation space, applying this framework to our actual test suites will set its practicality. In addition, even though single test smells have been focused on here, future work should evolve into applying multi-label classification to detect multiple test smells simultaneously. Such an application of deep learning would refine the accuracy in detection by learning complex patterns that are beyond the currently defined heuristics.

A. Future Enhancement

So, for future work, we will extend detection to other test smells such as "Redundant Setup" and "Flaky Tests." Improvements also include the incorporation of some deep learning techniques, such as recurrent or transformer-based models, for better classification. Deploying this framework to real test suites across various programming languages and testing frameworks will enhance its generalizability. Moreover, expanding the model into multiple test smells simultaneous detection is going to capture more fine-grained relationships between entities in the test code and thus makes it more effective. Furthermore, this approach can be a plugin in the IDE for real-time feedback to the developers during the testing process.

REFERENCES

[1] Cruz, D., Santana, A., & Figueiredo, E. (2024, May). An Exploratory Evaluation of Continuous Feedback to Enhance Machine Learning Code Smell Detection. In Congresso Ibero-Americano em Engenharia de Software (CIbSE) (pp. 76-90). SBC.

[2] Li, Y., & Zhang, X. (2022). Multi-Label Code Smell Detection with Hybrid Model based on Deep Learning. In SEKE (pp. 42-47).

[3] Zhang, D., Song, S., Zhang, Y., Liu, H., & Shen, G. (2023). Code Smell Detection Research Based on Pre-training and Stacking Models. IEEE Latin America Transactions, 22(1), 22-30.

[4] Khleel, N. A. A., & Nehéz, K. (2023). Detection of code smells using machine learning techniques combined with data-balancing methods. International Journal of Advances in Intelligent Informatics, 9(3), 402-417.

[5]Li, F., Zou, K., Keung, J. W., Yu, X., Feng, S., & Xiao, Y. (2023). On the relative value of imbalanced learning for code smell detection. Software: Practice and Experience, 53(10), 1902-1927.

[6]Yadav, P. S., Rao, R. S., Mishra, A., & Gupta, M. (2024). Machine learning-based methods for code smell detection: a survey. Applied Sciences, 14(14), 6149.

[7] Peruma A, Almalki K, Newman CD, M, MW, Ouni A, Palomba F (2020) Tsdetect: an open source test smells detection tool. In: ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, pp 1650–1654

[8] Pecorelli F, Di Lillo G,Palomba F, De Lucia A (2020) Vitrum: a plug-in for the visualization of test-related metrics. In: AVI 2020, pp 1-3

[9] Martins L, Costa H, Machado I (2023) On the diffusion of test smells and their relationship with test code quality of java projects. J Softw Evol Process e2532

[10] Lambiase S, Cupito A, Pecorelli F, De Lucia A, Palomba F (2020) Just-in-time test smell detection and refactoring: the darts project. In: International conference on program comprehension, pp 441–445