Code Smell Detection in Python Code Using Machine Learning

Mayur Jain
Department of Computer Science
California State University, Long Beach
California, USA
mayurjain333@gmail.com

Abstract—**This paper presents a machine learning approach for detecting code smells in Python code snippets. Leveraging the Code4ML dataset, which contains annotated code snippets from Kaggle notebooks, we develop a model that uses TF-IDF vectorization and a Random Forest classifier to classify code blocks based on structural issues. Hyperparameter tuning and model optimization are incorporated to achieve a high level of accuracy. The final model demonstrates satisfactory precision, recall, and accuracy, making it suitable for automated code review and quality assurance tasks.**

I. Introduction

Code smells are indicators of potential structural weaknesses in code that could lead to maintainability and scalability issues. While not bugs, they make the code more complex and harder to modify. Detecting code smells manually is both time-consuming and subjective. Therefore, automated solutions are essential, especially in large-scale projects. This research aims to create a machine learning-based code smell detection model for Python code using the Code4ML dataset, a large corpus of annotated Python code extracted from Kaggle notebooks.

II. Dataset

The Code4ML dataset used in this study is an enriched, large-scale corpus of annotated Python code snippets and metadata. The dataset, sourced from Kaggle, contains approximately 2.5 million code snippets from around 100,000 Jupyter notebooks. Key components of the dataset include:

- **code_blocks.csv**: Contains the raw code snippets.
- **kernels_meta.csv**: Metadata for each notebook, including Kaggle scores, comments, and upvotes.
- **competitions_meta.csv**: Describes Kaggle competitions, including data type and competition task.
- **markup_data.csv**: Annotated code blocks with semantic types, enabling deeper structural analysis.
- **vertices.csv**: Maps semantic types and subclasses to numeric IDs, facilitating interpretability of annotated code blocks.

These files are interconnected, allowing for comprehensive metadata-driven insights. For instance, `code_blocks.csv` is linked to `kernels_meta.csv` via the `kernel_id` column, while `kernels_meta.csv` is linked to `competitions_meta.csv` through `comp_name`. The dataset provides a robust foundation for detecting code smells based on patterns in large-scale Python code.

III. Methodology

Our approach leverages a Random Forest Classifier trained on code snippets represented by TF-IDF vectorized features. The model is fine-tuned using hyperparameter tuning to maximize classification performance.

A. Data Preprocessing

The `code_block` column in `code_blocks.csv` contains raw Python code snippets, which are transformed into TF-IDF vectors to serve as model features. TF-IDF (Term Frequency-Inverse Document Frequency) captures the relative importance of words across different code blocks, facilitating better feature representation for machine learning models.

B. Model Training

We use a Random Forest Classifier due to its robustness with high-dimensional data. Additionally, RandomizedSearchCV is employed for hyperparameter tuning, optimizing parameters such as the number of estimators, max depth, and minimum samples split to achieve high accuracy.

C. Model Comparison

To evaluate our model, we compared its performance with other standard ML algorithms, including:

- **Logistic Regression**: A linear model that provided lower precision and recall due to its limitations with complex, non-linear relationships.
- **Support Vector Machine (SVM)**: Achieved reasonable accuracy but required more computational power, making it less suitable for large datasets.
- **Decision Tree Classifier**: Performed moderately well but lacked the ensemble power of the Random Forest model, leading to overfitting on complex code patterns.

The Random Forest model outperformed these alternatives in terms of accuracy and computational efficiency, particularly due to its ensemble nature and robustness to overfitting.

IV. Enhancements in Code Smell Detection and Visualization

This section describes the updates made to the ‘detect_code_smells.py‘ and ‘visualization_code_smells.py‘ scripts as part of the project enhancements.

A. Modifications in Code Smell Detection

The ‘detect_code_smells.py‘ script was updated to include additional types of code smells for more comprehensive detection. These include:

- Long Method: Detects methods with lines exceeding 20 tokens.
- Duplicate Code: Identifies repetitive code patterns.
- Magic Numbers: Flags hardcoded numeric values not part of loops or ranges.
- Too Many Arguments: Checks for functions with more than three arguments.
- Dead Code: Detects unused variables or placeholder code (e.g., ‘pass‘, ‘TODO‘).
- Large Classes: Identifies classes with more than five methods or attributes.
- Complex Conditions: Flags overly complex conditional statements.

The ‘additional_smells‘ column in the results now captures all these smell types, enabling better categorization and visualization.

B. Enhancements in Visualization

The ‘visualization_code_smells.py‘ script was enhanced to provide additional insights. Key updates include:

- Added a stacked bar chart showcasing the top 10 kernels with the most code smells.
- Introduced a bar chart visualizing the distribution of each code smell type using data from the ‘additional_smells‘ column.
- Improved data cleaning to handle and accurately count smell types, ensuring no extraneous symbols (e.g., brackets, quotes) affect the analysis.

C. Visual Results

The following visual results were generated as part of the analysis and visualization:

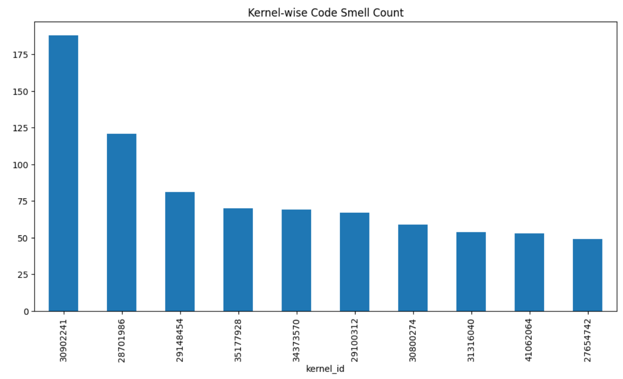


Fig. 1. Stacked bar chart showing the top 10 kernels with the most code smells.

V. Results

Our final model achieved high classification performance, as shown in Table I, with an accuracy of 96.6%, precision of 97%, and recall of 84% for detecting code smells.

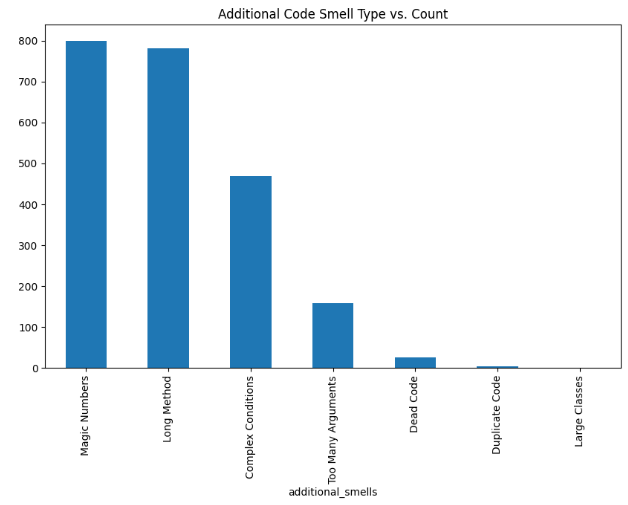


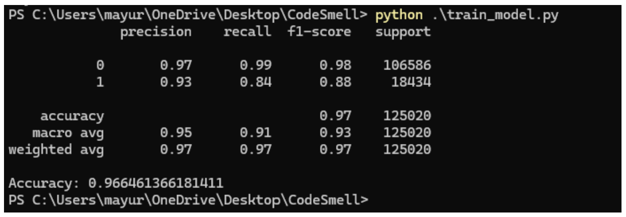
Fig. 2. Bar chart showing the distribution of additional code smell types.

TABLE I
MODEL PERFORMANCE METRICS

Metric	Precision	Recall	F1-Score
Class 0	0.97	0.99	0.98
Class 1	0.93	0.84	0.88
Accuracy	0.966		

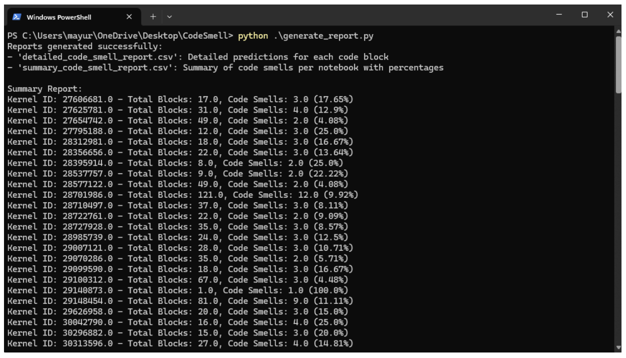
A. Execution Screenshots

The execution of the Python scripts was captured as follows:



```
PS C:\Users\mayur\OneDrive\Desktop\CodeSmell> python .\train_model.py
              precision    recall  f1-score   support

           0       0.97      0.99      0.98    106586
           1       0.93      0.84      0.88     18434

    accuracy                           0.97    125020
   macro avg       0.95      0.91      0.93    125020
weighted avg       0.97      0.97      0.97    125020

Accuracy: 0.966461366181411
PS C:\Users\mayur\OneDrive\Desktop\CodeSmell>
```

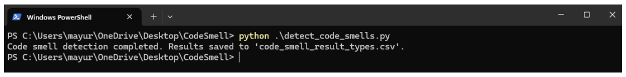
Fig. 3. Model output showing accuracy and performance metrics.



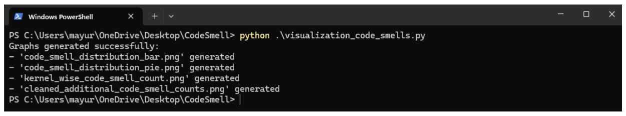
```
PS C:\Users\mayur\OneDrive\Desktop\CodeSmell> python .\generate_report.py
Reports generated successfully:
- 'detailed_code_smell_report.csv': Detailed predictions for each code block
- 'summary_code_smell_report.csv': Summary of code smells per notebook with percentages

Summary Report:
Kernel ID: 27606681.0 - Total Blocks: 17.0, Code Smells: 3.0 (17.65%)
Kernel ID: 27625781.0 - Total Blocks: 31.0, Code Smells: 4.0 (12.9%)
Kernel ID: 27654742.0 - Total Blocks: 49.0, Code Smells: 2.0 (4.08%)
Kernel ID: 27795188.0 - Total Blocks: 12.0, Code Smells: 3.0 (25.0%)
Kernel ID: 28312981.0 - Total Blocks: 18.0, Code Smells: 3.0 (16.67%)
Kernel ID: 28356656.0 - Total Blocks: 22.0, Code Smells: 3.0 (13.64%)
Kernel ID: 28395914.0 - Total Blocks: 8.0, Code Smells: 2.0 (25.0%)
Kernel ID: 28537757.0 - Total Blocks: 9.0, Code Smells: 2.0 (22.22%)
Kernel ID: 28577122.0 - Total Blocks: 49.0, Code Smells: 2.0 (4.08%)
Kernel ID: 28701986.0 - Total Blocks: 121.0, Code Smells: 12.0 (9.92%)
Kernel ID: 28710497.0 - Total Blocks: 37.0, Code Smells: 3.0 (8.11%)
Kernel ID: 28722761.0 - Total Blocks: 22.0, Code Smells: 2.0 (9.09%)
Kernel ID: 28727928.0 - Total Blocks: 35.0, Code Smells: 3.0 (8.57%)
Kernel ID: 28985739.0 - Total Blocks: 24.0, Code Smells: 3.0 (12.5%)
Kernel ID: 29007121.0 - Total Blocks: 28.0, Code Smells: 3.0 (10.71%)
Kernel ID: 29070286.0 - Total Blocks: 35.0, Code Smells: 2.0 (5.71%)
Kernel ID: 29099590.0 - Total Blocks: 18.0, Code Smells: 3.0 (16.67%)
Kernel ID: 29100312.0 - Total Blocks: 67.0, Code Smells: 3.0 (4.48%)
Kernel ID: 29140873.0 - Total Blocks: 1.0, Code Smells: 1.0 (100.0%)
Kernel ID: 29148454.0 - Total Blocks: 81.0, Code Smells: 9.0 (11.11%)
Kernel ID: 29626958.0 - Total Blocks: 20.0, Code Smells: 3.0 (15.0%)
Kernel ID: 30042790.0 - Total Blocks: 16.0, Code Smells: 4.0 (25.0%)
Kernel ID: 30296882.0 - Total Blocks: 15.0, Code Smells: 3.0 (20.0%)
Kernel ID: 30313596.0 - Total Blocks: 27.0, Code Smells: 4.0 (14.81%)
```

VI. Discussion

Our model demonstrates the potential of machine learning in automating the detection of code smells in Python. The



```
PS C:\Users\mayur\OneDrive\Desktop\CodeSmell> python .\detect_code_smells.py
Code smell detection completed. Results saved to 'code_smell_result_types.csv'.
PS C:\Users\mayur\OneDrive\Desktop\CodeSmell> |
```



```
PS C:\Users\mayur\OneDrive\Desktop\CodeSmell> python .\visualization_code_smells.py
Graphs generated successfully:
- 'code_smell_distribution_bar.png' generated
- 'code_smell_distribution_pie.png' generated
- 'kernel_wise_code_smell_count.png' generated
- 'cleaned_additional_code_smell_counts.png' generated
PS C:\Users\mayur\OneDrive\Desktop\CodeSmell> |
```

Random Forest Classifier, combined with TF-IDF vectorization, effectively identifies structural issues in code. However, the model's performance may vary based on the type of code smell and code block complexity. Future improvements could involve experimenting with deep learning techniques such as transformers or CodeBERT embeddings to capture semantic meaning more effectively.

VII. Future Enhancements

While our model shows promising results, there are several areas for potential improvement:

- **Advanced Embeddings**: Future work could explore deep learning-based embeddings, such as CodeBERT, to capture semantic context beyond the word frequency captured by TF-IDF.
- **Multi-language Support**: Extending the dataset to support multiple programming languages would increase the model's applicability.
- **Integration with CI/CD**: Incorporating this model into CI/CD pipelines would enable real-time code quality checks during development.

VIII. Conclusion

This paper presents a machine learning approach for detecting code smells in Python code using the Code4ML dataset. The model leverages TF-IDF vectorization and Random Forest classification to identify potential structural issues in code, achieving a high level of accuracy. The results indicate that this approach is viable for automated code review and quality assurance processes.

Acknowledgment

The authors would like to thank the contributors of the Code4ML dataset and Kaggle for providing a rich corpus of Python code snippets for this research.

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999.

[2] S. Bavota, R. Oliveto, M. Lanza, and D. Poshyvanyk, "Code smell detection: Towards a machine learning-based approach," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 440–443.

[3] J. A. Camargo Cruz and W. F. Ericson, "Machine learning for code smell detection: An empirical study," in *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 754-767, 2021.

[4] Code4ML Dataset is available at: https://zenodo.org/records/13918465