Decision Tree for Smell Code Detection in Python: A Practical Implementation

Fajar Ratnawati[1], Jaroji[2]

{fajar@polbeng.ac.id[1], jaroji@polbeng.ac.id[2]}

State Polytechnic of Bengkalis, Jl. Bathin Alam, Bengkalis, Riau, Indonesia[1,2]

Abstract. This article discusses the approach to detect code that falls under the category of code smells in the Python programming language. The methodology involves reading each line of code and subsequently analyzing whether the program contains code smells. The code analysis is performed using a model trained with decision tree and J48 algorithms. The aim of this research, apart from developing previous research, is also to improve code quality and reduce security risks. Several aspects considered in this research include the number of parameters within a class, unused variables, duplicated print statements, and the number of classes within a code program. The constructed model utilizing the decision tree algorithm is then implemented into a web-based system, where input code for smell code identification is in the form of files with the extension .py.

Keywords*: smell code, decision tree*, j48, pyhton, *class*, *duplicated print*, *unused variable*, parameter, *machine learning*.

1 Introduction

In the continuously evolving digital era, the Python programming language has become one of the most popular and widely used programming languages. Its high popularity has led to the creation of complex and large Python codes by developers. However, the increasing complexity of these codes also brings the potential for problems in their management. One of the common issues in software development is the emergence of "smell code" or "code smells." Smell code is a term used to describe poorly organized and difficult-to-understand code that tends to reduce the overall software quality. Smell code can lead to various problems, such as difficulties in maintenance, security vulnerabilities, and poor application performance [1].

In software development, smell code has become a significant concern as it can affect the quality and maintainability of the code. Smell code refers to poor programming practices that result in code that is difficult to understand, hard to maintain, and prone to errors. Smell code can hinder developer productivity, increase the likelihood of errors, and ultimately reduce project sustainability. Therefore, the effective identification and handling of smell code have become crucial in software development practices [1].

ABEC 2023, September 21, Bengkalis, Indonesia
Copyright © 2024 EAI
DOI 10.4108/eai.21-9-2023.2342883

Previously, the process of detecting smell code was done manually or by examining each line of code in a program, which was time-consuming and prone to human errors. Therefore, there is a need for a system that can efficiently detect smell code in the codebase, making it easier to identify and address the necessary improvements for better maintainability and readability of the program. To address the smell code issue, static analysis techniques can be used to identify and analyze the characteristics of smell code in a software project. One of the most popular and effective static analysis techniques is the use of Decision Trees [2].

This research aims to implement Decision Tree in a smell code application for the Python programming language. By using Decision Tree, we can perform static analysis of Python code and identify patterns that indicate the presence of smell code. This will assist developers in identifying potential issues in their code, understanding areas that need improvement, and ultimately enhancing the quality of the software produced [3].

Decision trees are used because they are easy to apply and easy to understand in making decisions in this model. Apart from that, decision trees have good generalization capabilities, meaning that if you want to apply them to other projects, you don't need to make too many changes. The performance of this algorithm is also good, after the data is drilled the decision making process will be relatively faster.

This research has limitations in the parameters used to detect smell code in the Python programming language, which include the number of parameters within a class, unused variables, duplicated print statements, and the number of classes in the code program. However, the results of this study are expected to be further developed and contribute to future research with improved systems. Through further research, it is possible that other features or methods may be discovered to enhance the accuracy and effectiveness of smell code detection.

In this study, we will explain the method used in implementing Decision Tree in the Python smell code application. Additionally, we will outline the steps taken in the analysis and testing process of the application. The results and findings of this research are expected to provide valuable guidance for Python developers in improving the quality of their code, thus creating more reliable, efficient, and maintainable software. We believe that the utilization of Decision Tree in the smell code application for the Python programming language will bring significant benefits to developers and the software industry as a whole. Therefore, this report is expected to make a positive contribution to the development and management of Python software projects in the future..

2 Research Methods

In detecting smell code in the Python programming language, there are several stages that need to be carried out. The stages are as follows:

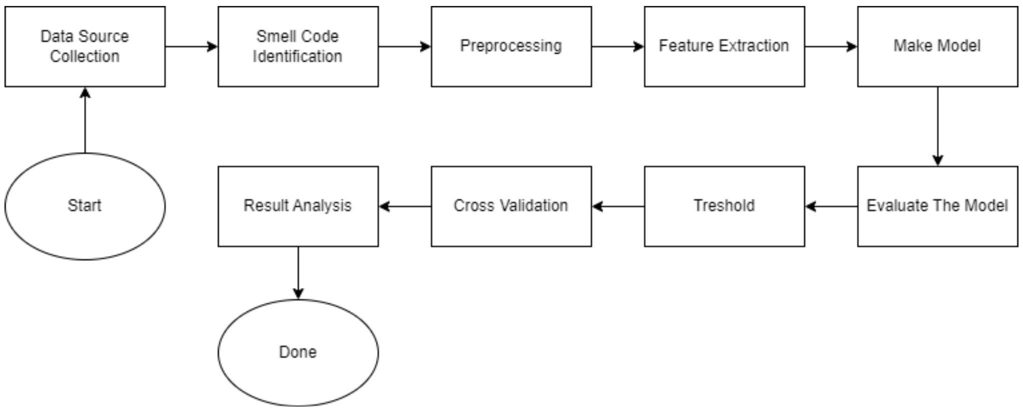


Fig. 1. Stage of smell code detection

a. Data Source Code Collection:
   - Identifying relevant data sources of Python source code for the research.
   - Gathering various software projects that use the Python programming language, encompassing different levels of complexity and code sizes.

b. Smell code indetification
   - Conducting literature reviews and previous research to identify various common types of smell code found in the Python programming language.
   - Creating a list of smell codes that are the focus of the research.
   - Establishing rules or criteria to determine whether a piece of code falls under the category of smell code or not.

c. Data Preprocessing
   - Cleaning the source code data by removing comments, spaces, and other irrelevant characters.
   - Applying other preprocessing techniques such as duplicate code removal and normalization.
   - Using preprocessing techniques like data cleaning, data integration, and data transformation to ensure the dataset used is clean and suitable for smell code classification.

d. Feature Extraction
   - Identifying relevant features to describe the characteristics of the source code related to the smell code being targeted.
   - Implementing the feature extraction process to convert the source code into a numerical representation that can be used by the Decision Tree.

e. Decision Tree Model Creation
   - Selecting the appropriate Decision Tree algorithm for smell code analysis in the Python programming language. In this research, the C4.5 algorithm is used.
   - Training the preprocessed and feature-extracted data to generate the Decision Tree model.
   - Splitting the data into training and test datasets for model evaluation to determine its accuracy.

f. Model Evaluation
   - Testing the Decision Tree model using the test data that has not been seen by the model before.

- Analyzing the model's performance with metrics such as accuracy, precision, recall, and F1-score.

g. Threshold Determination
   - Setting the threshold in the model analysis results to decide whether a code is classified as smell code or not.
   - Optimizing the threshold to achieve a balance between effectively detecting smell code and reducing identification errors.

h. Cross-validation
   - Performing cross-validation to ensure that the built Decision Tree model can generalize well on unseen data

i. Result analysis
   - Identifying frequently occurring types of smell code and their characteristic patterns in Python source code.
   - Providing interpretation of the analysis results and their implications for source code improvement.

2.1. Decision tree

A tree is a data structure consisting of nodes and edges. There are three types of nodes in a tree, namely root/node, branch/internal node, and leaf node[3]. A decision tree is a simple representation of a classification technique for several different classes, where internal nodes and root nodes have attribute names, the edges have possible attribute value labels, and leaf nodes are marked with different classes[3].

2.2. Decision Tree C4.5 (J48)

Algorithm C4.5 is the algorithm used to build a decision tree. Decision trees are a powerful and well-known method for classification and prediction. The C4.5 algorithm uses training data which consists of cases or records in the database. Each case has an attribute value for a class, and attributes can contain discrete or continuous data[3]. The C4.5 algorithm can also handle cases where there are no values for some attributes, but class attributes must be discrete and cannot be empty. In general, the C4.5 algorithm for building decision trees involves the following steps:

1. Selecting attributes as the root of the tree.
2. Creating branches for each attribute value.
3. Dividing cases into the appropriate branches.
4. Repeating the process for each branch until all cases in the branch have the same class.

2.3. Clasification

Classification is the process of finding a model that can distinguish existing data classes or concepts. The goal is to use the model to make predictions about the class of objects whose class is unknown. In the context of this study, classification is used to identify smell code in Python program code[4]. By using the decision tree algorithm as a classification method, this study succeeded in developing an effective model for differentiating codes with a smell code and without a smell code. This model can be used to predict the class of unknown program code, whether it is included in the smell code or not. This allows software developers to take necessary actions in the repair and maintenance of program code. In this study, classification becomes an important part of the data preprocessing stage. By classifying code based on the presence of smell code, developers can have a clearer insight into the quality of the program code they are working on. In the long term, this can improve the quality of the software produced and facilitate the maintenance process in the future. Thus, the use of classification in the context of this study has significant benefits in detecting and overcoming smell codes in software development.

2.4. System Architecture

The following is the architecture of the python programming language smell code system with a website-based decision tree (J48) algorithm:

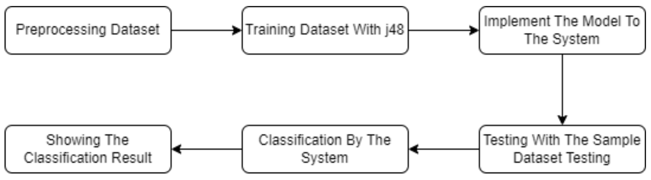
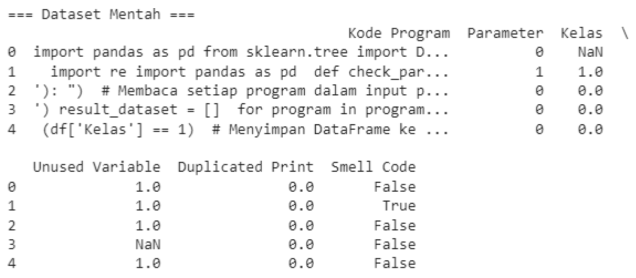

Preprocessing Dataset
Training Dataset With j48
Implement The Model To The System
Testing With The Sample Dataset Testing
Classification By The System
Showing The Classification Result

Fig. 2. System architecture smell code python programming language

2.4.1. Data Preprocessing

The first stage in the system that will be built later is preprocessing, in this stage the dataset that has been collected is then cleaned so that the data that will become the training data becomes cleaner so that it will increase the accuracy value of the algorithm used, namely the decision tree (J48)[5] . The following is an image of raw data that has not been preprocessed:



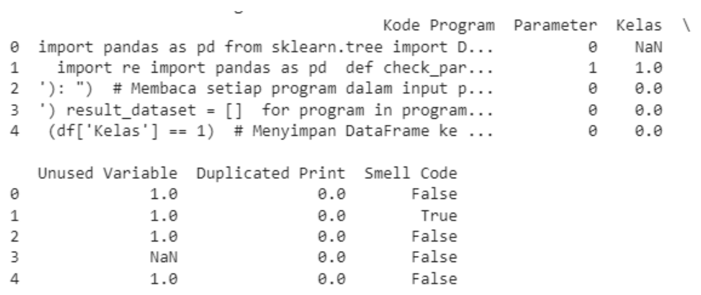
```
=== Dataset Mentah ===
                                        Kode Program  Parameter  Kelas  \
0  import pandas as pd from sklearn.tree import D...          0    NaN
1    import re import pandas as pd  def check_par...          1    1.0
2  '): ")  # Membaca setiap program dalam input p...          0    0.0
3  ') result_dataset = []  for program in program...          0    0.0
4   (df['Kelas'] == 1)  # Menyimpan DataFrame ke ...          0    0.0

   Unused Variable  Duplicated Print  Smell Code
0              1.0               0.0       False
1              1.0               0.0        True
2              1.0               0.0       False
3              NaN               0.0       False
4              1.0               0.0       False
```

Fig. 3. Initial dataset

From the raw data shown above, it is necessary to carry out the preprocessing stages which can be explained as follows:

1. Data Cleaning. In this step, the raw data will be cleaned through several processes such as filling in missing values, smoothing noisy data, and resolving inconsistencies found in the data [6]. In the raw dataset used in this study, there are several features that have an empty value, so that data cleaning can be carried out, the following are the results of this stage:

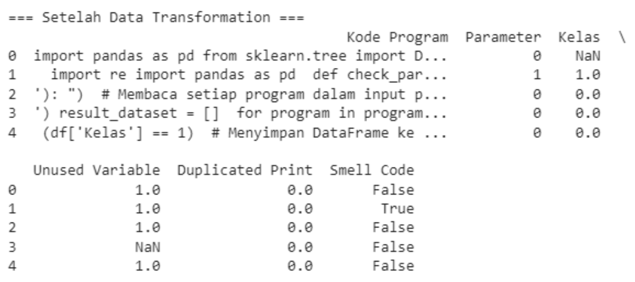


```
                                        Kode Program  Parameter  Kelas  \
0  import pandas as pd from sklearn.tree import D...          0    NaN
1    import re import pandas as pd  def check_par...          1    1.0
2  '): ")  # Membaca setiap program dalam input p...          0    0.0
3  ') result_dataset = []  for program in program...          0    0.0
4   (df['Kelas'] == 1)  # Menyimpan DataFrame ke ...          0    0.0

   Unused Variable  Duplicated Print  Smell Code
0              1.0               0.0       False
1              1.0               0.0        True
2              1.0               0.0       False
3              NaN               0.0       False
4              1.0               0.0       False
```

Fig. 4. Result of *data cleaning process*

2. Data Integration. The next step in the dataset preprocessing stage is data integration, where this step will combine data from various sources into a single data unit (dataset). This step will also ensure data has the same format and attributes / features, remove features that are not needed, and detect values that have conflicts [7]. In the dataset used there are no features or attributes that can cause conflict.
3. Data Transformation. The next step is data transformation, this step is carried out so that the data is not excessive which aims to homogenize the data in the dataset [8]. The following is the result of this stage:

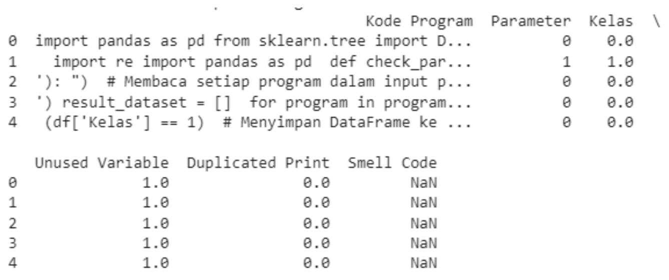


```
=== Setelah Data Transformation ===
                                        Kode Program  Parameter  Kelas  \
0  import pandas as pd from sklearn.tree import D...          0    NaN
1    import re import pandas as pd  def check_par...          1    1.0
2  '): ")  # Membaca setiap program dalam input p...          0    0.0
3  ') result_dataset = []  for program in program...          0    0.0
4   (df['Kelas'] == 1)  # Menyimpan DataFrame ke ...          0    0.0

   Unused Variable  Duplicated Print  Smell Code
0              1.0               0.0       False
1              1.0               0.0        True
2              1.0               0.0       False
3              NaN               0.0       False
4              1.0               0.0       False
```

Fig. 5. Result of data transformation process

From the preprocessing stages that have been carried out above for the dataset used, the following is the result after the preprocessing of the dataset has been carried out:



```
                                        Kode Program  Parameter  Kelas  \
0  import pandas as pd from sklearn.tree import D...          0    0.0
1    import re import pandas as pd  def check_par...          1    1.0
2  '): ")  # Membaca setiap program dalam input p...          0    0.0
3  ') result_dataset = []  for program in program...          0    0.0
4   (df['Kelas'] == 1)  # Menyimpan DataFrame ke ...          0    0.0

   Unused Variable  Duplicated Print  Smell Code
0              1.0               0.0         NaN
1              1.0               0.0         NaN
2              1.0               0.0         NaN
3              1.0               0.0         NaN
4              1.0               0.0         NaN
```

Fig. 6. The final result of preprocessing

2.4.2. Training Dataset With J48

The next stage is to conduct training using the decision tree algorithm (J48). The dataset used in this research totals 147 data, where the data used for training and testing is divided by a ratio of 70:30 for data that has been preprocessed in the previous stages.

2.4.3. Implementation of Model

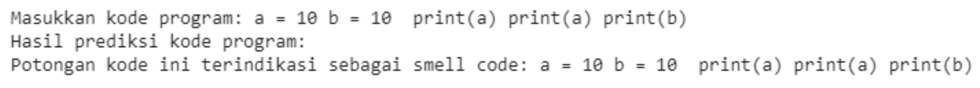
After carrying out the training stages, then the decision tree algorithm (J48) model will be implemented into the system which will be used to detect smell code in the Python programming language. In the implementation process, the system to be built will use the flask framework or a framework specifically for the Python programming language. To implement the model into the system, first of all the model that has been worked on is exported in a special file form which can later be used in the flask framework. Then the appearance of the website-based system is created with the help of the flask framework so that the website can carry out the classification process of the exported models. So finally after the appearance and business logic of the system are well made, it can be continued with website hosting that is made so that the python programming language smell code classification website can be used.

2.4.4. Model Testing

The next stage is testing the model that has been implemented in the system using previously shared data as a training dataset. In this test, the model will be tested using data that has never been seen before, which is known as the testing dataset. This aims to measure the performance and accuracy of the model in making class predictions on new data. Model testing that is done properly will provide confidence that the model can work effectively and reliably in classifying unknown data.

2.4.5. Clafication Result

The final stage is that the system will carry out a classification process which then results from the process will be displayed on the screen whether the given code belongs to the smell code type or category or not. To be able to classify smell code in program code, the system will ask for input in the form of program code in the Python programming language. Then the system will check the program code for the number of parameters in a class, unused variables, duplicated prints and the number of classes. After checking, the system will then carry out a classification process based on the dataset that has been preprocessed. Then in the end the system will display the results of the classification process that has been carried out by printing the smell code results if the program code that is input contains a smell code and vice versa if the code does not contain a smell code. The following is the result of the smell code classification process carried out by the system later.



```
Masukkan kode program: a = 10 b = 10  print(a) print(a) print(b)
Hasil prediksi kode program:
Potongan kode ini terindikasi sebagai smell code: a = 10 b = 10  print(a) print(a) print(b)
```

Fig. 6. The final result of smell code clasification

3 Result and Discussion

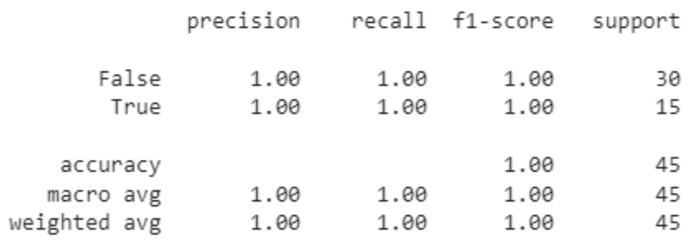
In this study, we conducted an experiment to evaluate the performance of the decision tree algorithm in identifying code smells in Python code. We analyzed several important features, such as function length, code duplication, conditional complexity, and others, and found that these features make a significant contribution to smell code recognition.

In using a trained decision tree algorithm, we can classify new code and determine whether it has a smell code or not. We use standard evaluation metrics such as accuracy, precision, recall, and F1-score to analyze algorithm performance. The experimental results show that the decision tree algorithm achieves satisfactory accuracy in classifying Python code with smell code. The values for accuracy, precision, recall, and F1-score are 100%, 100%, 100%, and 100%, respectively.

The results of this study have important implications for software developers. Developers can use the decision tree algorithm as a tool to detect and analyze smell code in the Python code they develop. By knowing the smell code contained in their code, developers can take necessary corrective actions to improve code quality and maintainability. This will help reduce complexity and make code easier to maintain, allowing developers to produce software that is cleaner, more efficient, and easier to implement.

Through the use of Google Colab, we conduct data training and testing using prepared datasets. Using Google Colab as a powerful computing platform allows us to efficiently conduct training and testing processes. We can easily import datasets, apply decision tree algorithms, and analyze the results. The results of our training and testing show the good ability of the decision tree algorithm in classifying Python code with smell code.

It is hoped that this research can make a useful contribution to the Python software developer community. We also encourage developers to explore using other machine learning algorithms and developing more complex models for smell code analysis in other programming languages. Thus, software development can be more efficient and of high quality, meeting the needs and expectations of users. The following are the results of the data carried out by training and testing using Google Colab



```
              precision    recall  f1-score   support

       False       1.00      1.00      1.00        30
        True       1.00      1.00      1.00        15

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Fig. 8. Result of matrix evaluation

4 Conclusion

This study shows that the use of a decision tree algorithm in identifying smell code in the Python programming language is an effective method. By using this algorithm, research has succeeded in recognizing smell code in Python code with an adequate level of accuracy. These findings make an important contribution in increasing understanding and awareness of smell code in Python software development.

The results of this study can provide practical guidance and recommendations for Python software developers to identify and overcome code smells that may occur in their code. By using the decision tree algorithm, developers can quickly and accurately recognize smell code and take appropriate corrective steps.

In addition, this research also provides impetus for the use of other machine learning algorithms in the analysis of smell code in other programming languages. By looking at the success of the decision tree algorithm, this research encourages the exploration of using more complex models and more sophisticated machine learning methods for smell code analysis in other programming languages.

With this research, it is hoped that software developers can be more sensitive to smell codes and implement best practices in developing clean and efficient software. In addition, this research can also be a basis for further research in this field, such as developing more sophisticated models or researching smell code in the context of different programming languages.

5 References

[1] Kevin Azwega, Adam Hendra Brata, Eriq Muhammad Adams Jonemaro: Pengembangan Sistem Deteksi *God Class* dan *Brain ClassCode Smell*. pp. 3972-3977 (2020)

[2] Hanson Prpihantoro Putro, Inggriani Liem: Deteksi *Code Smell* Pada Kode Program Dalam Representasi AST Dengan Pendekatan *By Rules*. pp. 23-28 (2010)

[3] Asmaul Husnah Nasrullah: Implementasi Algoritma *Decision Tree* Untuk Klasifikasi Produk Laris. pp. 45-51 (2021)

[4] Rintho Rante Rerung: Penerapan *Data Mining* dengan Memanfaatkan Metode *Association Rule* untuk Promosi Produk. pp. 89-98 (2018)

[5] Jajang Jaya Purnama, Sri Rahayu: Klasifikasi Konsumsi Energi Industri Baja Menggunakan Teknik Data Mining. pp. 395-407 (2022)

[6] Jason Brownlee: Data Preparation For Machine Learning. pp. 16-24 (2020)

[7] Anne Richelle, Chintan Joshi, Nathan E. Lewis: Assessing key decisions for transcriptomic data integration in biochemical networks. pp. 1-18 (2019)

[8] Nengah Widya Utami, A.A. Istri Ita Paramitha: Penerapan Data Mining Untuk Mengetahui Pola Pemilihan Program Studi di STMIK Primakara Menggunakan Algoritma K-Means Clustering. pp. 456-563 (2021)