

Classificação de código Fonte Utilizando Características de Textura de Code Minimap

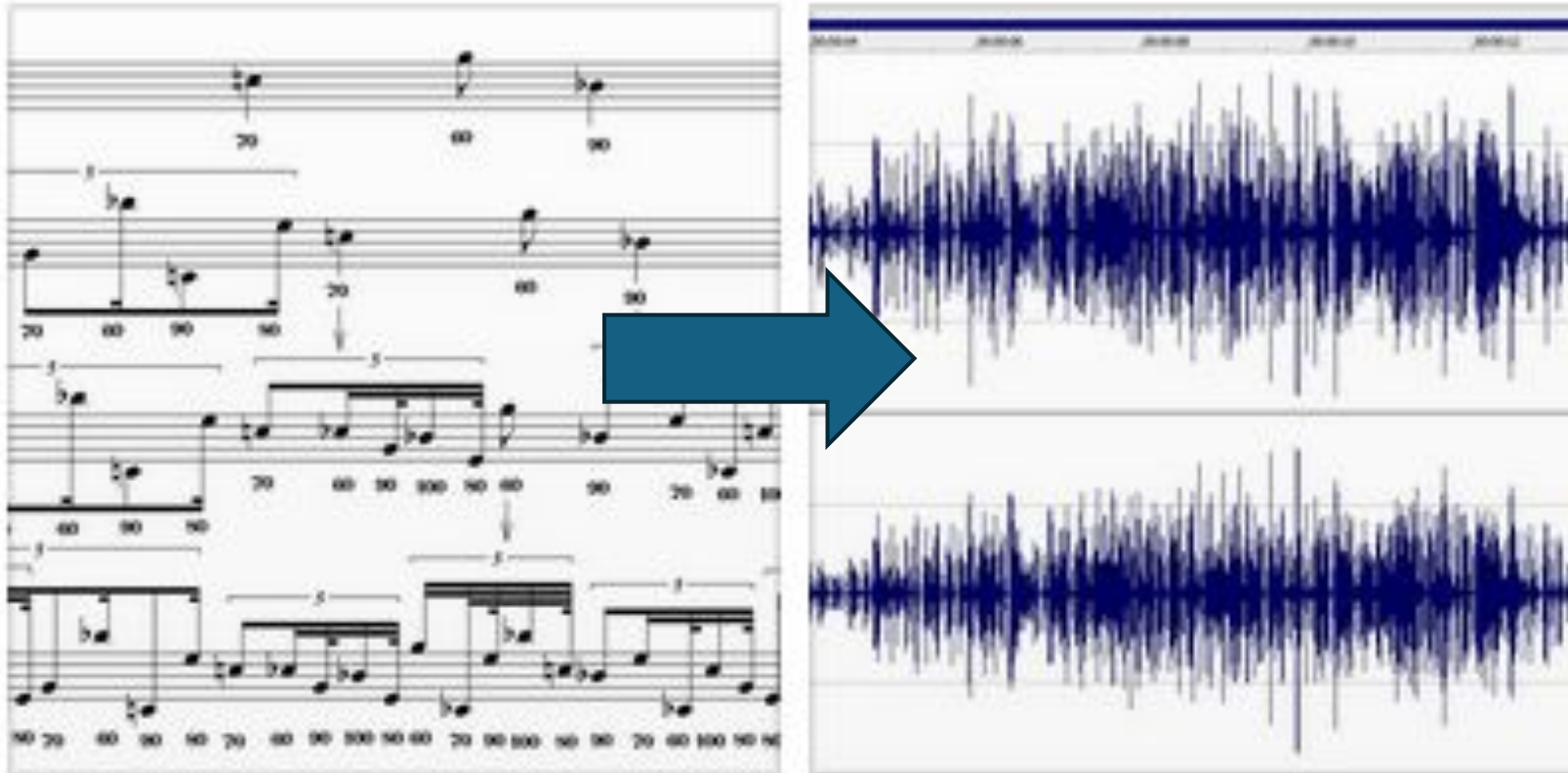
Munif Gebara Junior, (UEM)

Dr. Yandre Maldonado e Gomes da Costa (UEM)

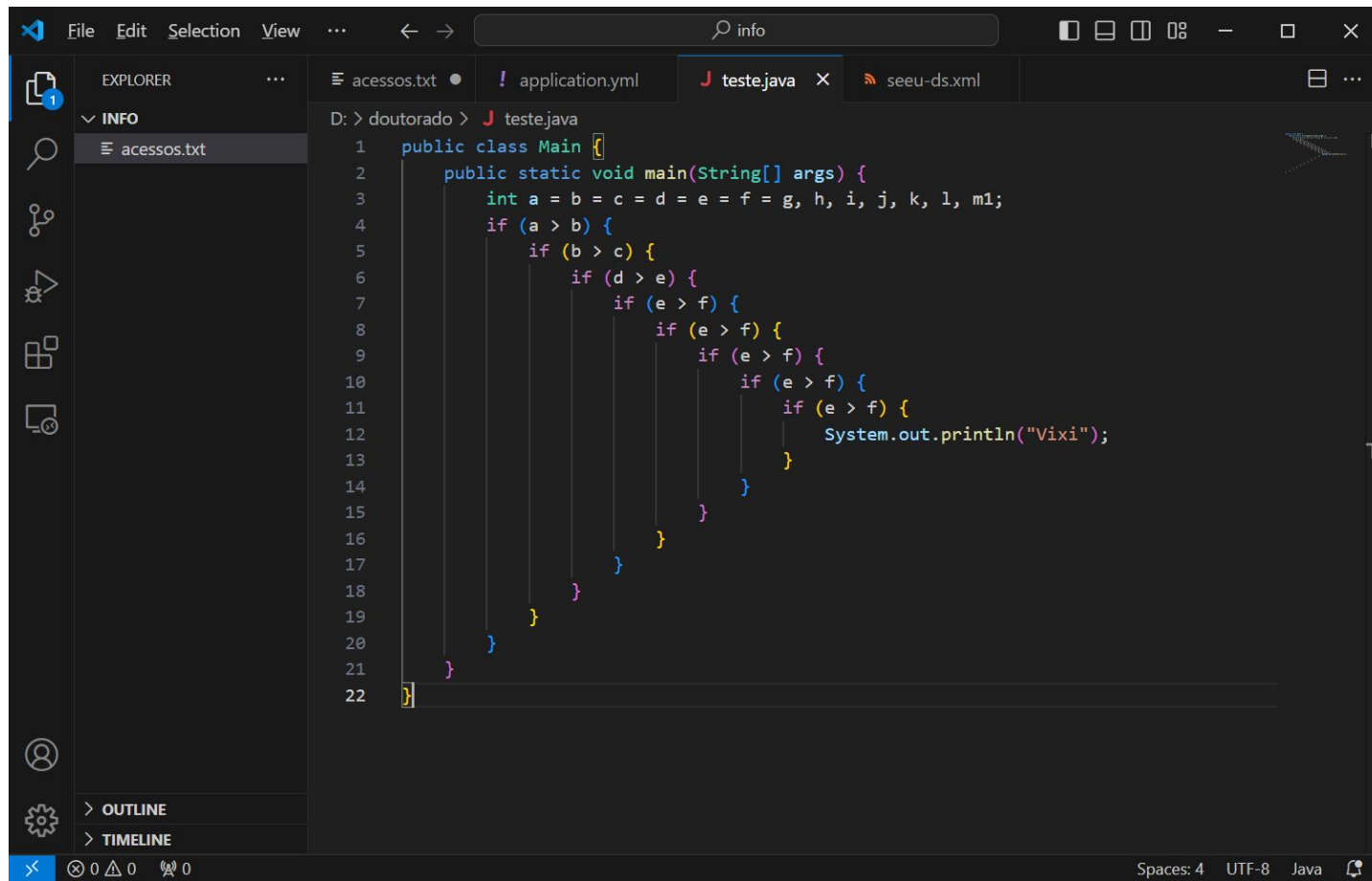
Dr. Igor Wiese, (UTFPR)

Referencial teórico

- COSTA, Y. M. G.; Oliveira, L.S.; Koerich, A.L.; GOUYON, F.; Martins, J.G. Music
- Genre Classification Using LBP Textural Features



MiniMap



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left shows a file named `acessos.txt`. The main editor window displays a Java file named `teste.java` with the following code:

```
D: > doutorado > J teste.java
1 public class Main {
2     public static void main(String[] args) {
3         int a = b = c = d = e = f = g, h, i, j, k, l, m1;
4         if (a > b) {
5             if (b > c) {
6                 if (d > e) {
7                     if (e > f) {
8                         if (e > f) {
9                             if (e > f) {
10                                if (e > f) {
11                                    if (e > f) {
12                                        System.out.println("Vixi");
13                                    }
14                                }
15                            }
16                        }
17                    }
18                }
19            }
20        }
21    }
22 }
```

The MiniMap view is visible on the right side of the editor window, showing a small, dark, and blurry representation of the code structure. A blue arrow points from the MiniMap view to the right, indicating its function as a navigation tool.



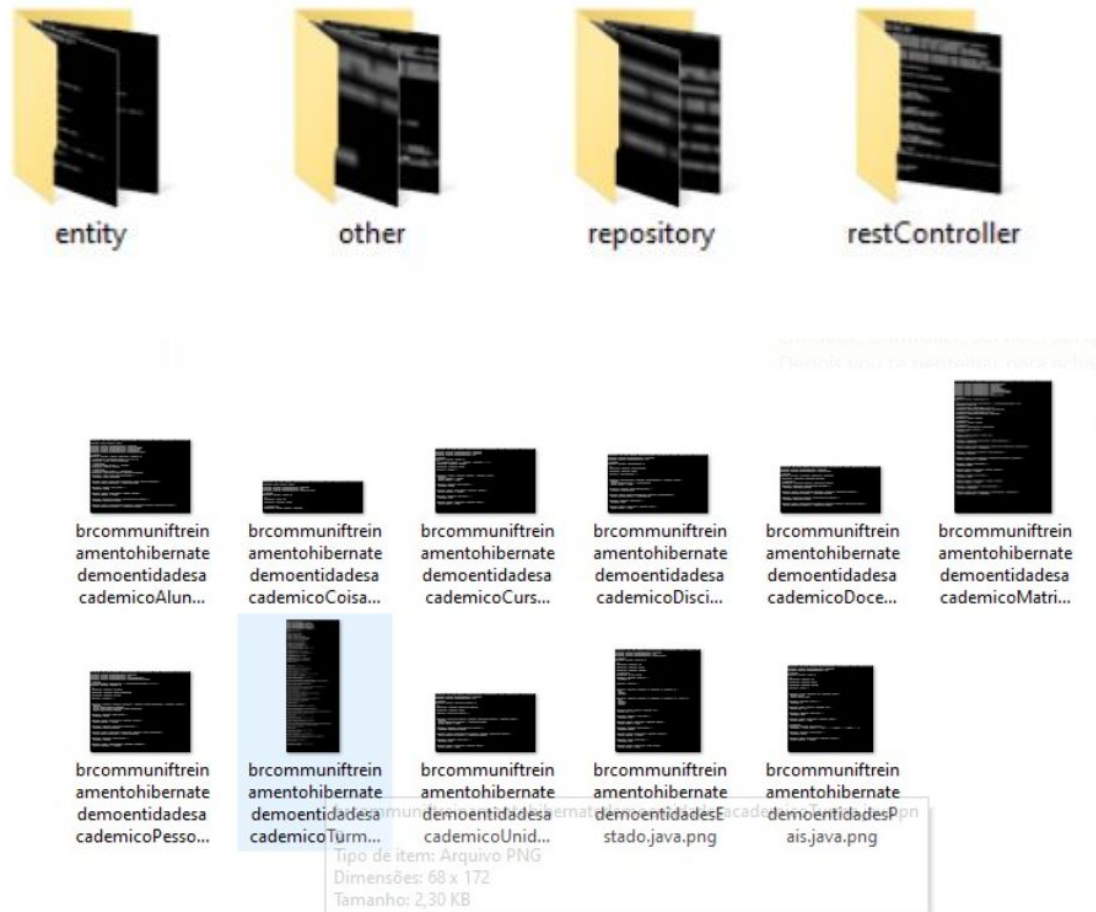
Criação do Dataset automatizada

```
with open(caminho_arquivo, 'r', encoding=encoding) as arquivo:
    for linha in arquivo:
        for x in range(len(linha.strip())):
            v = ord(linha[x])
            v = encrypt_chars(v)
            if v > 255:
                v = 255
            try:
                imagem.putpixel(xy: (x, y), value: (v, v, v))
            except IndexError:
                print(largura, altura, x, y, v)
        y += 1

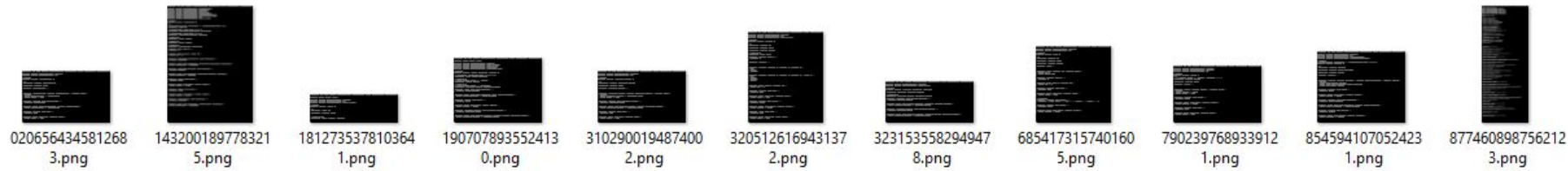
caminho_saida_com_tipo = caminho_saida + '/' + tipo + '/'
os.makedirs(caminho_saida_com_tipo, exist_ok=True)
arquivoImagem = create_16_digit_hash(caminho_arquivo) + '.png'
imagem.save(caminho_saida_com_tipo + arquivoImagem)
```

Criar um Dataset classificando Estereótipos

Fase Atual



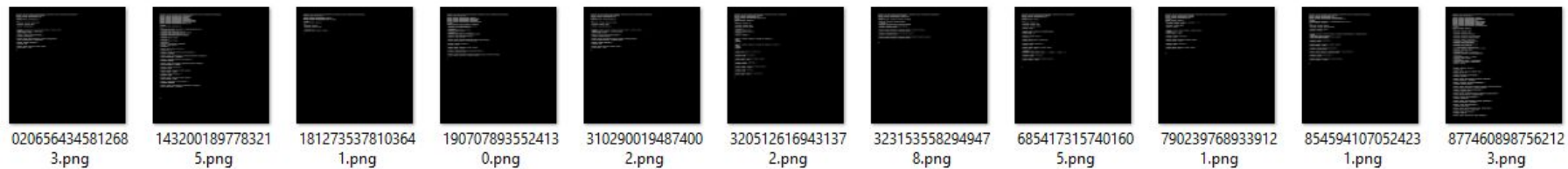
Padronizar tamanho da saída



-128px 128px

-Bordas 8px

-Área com dados 128px-8px-8px 112px 112px



Impedir engenharia reversa do código fonte

- Hash do nome dos arquivos
- Código ASCII transformado



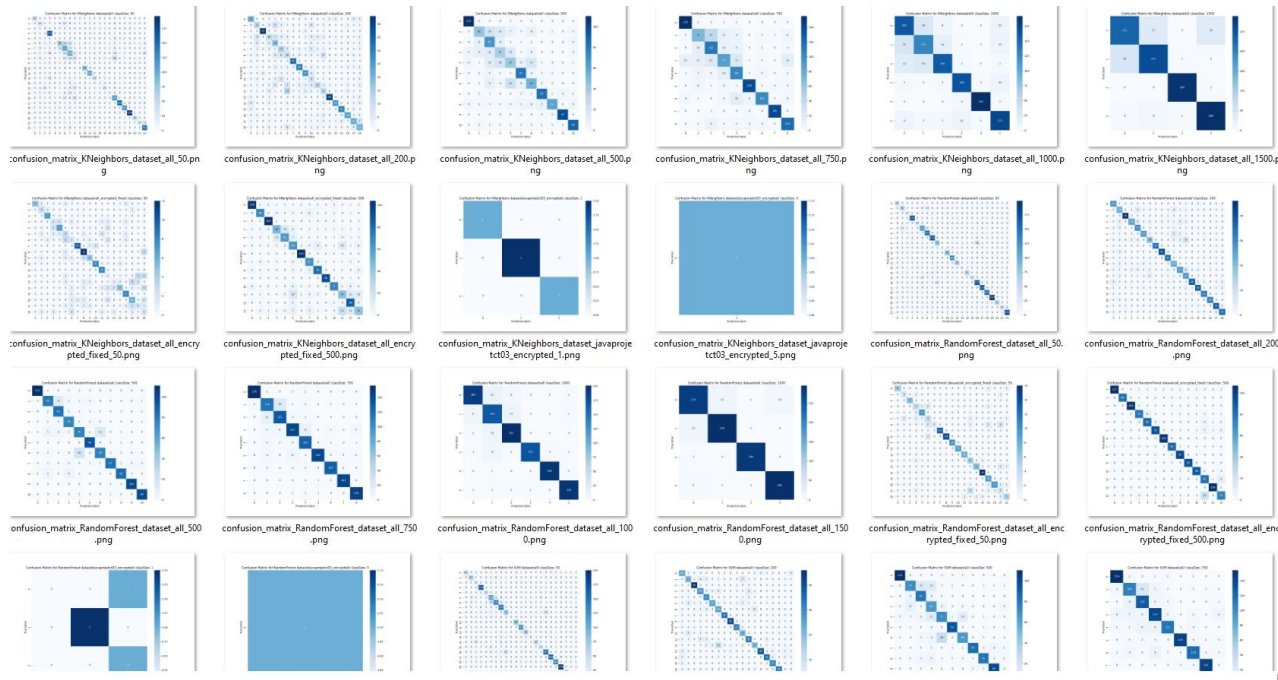
```
def encrypt_chars(asc_code): 1 usage new *  
    if asc_code < 33: # não-imprimíveis  
        return 32  
    if asc_code < 48: # símbolos  
        return asc_code  
    if asc_code < 58: # números  
        return 53  
    if asc_code < 65: # símbolos  
        return asc_code  
    if asc_code < 91: # maiúsculas  
        return 77  
    if asc_code < 97: # símbolos  
        return asc_code  
    if asc_code < 123: # minúsculas  
        return 109  
    if asc_code < 127: # símbolos  
        return asc_code  
    return 130
```


Base de Dados

14 Classes

500 amostras em cada classe

<https://github.com/munifgebara/codeminimap>



0 css

1 html

2 javaconverter

3 javadto

4 javaentity

5 javaimplementation

6 javajasper

7 javajsp

8 js

9 json

10 sql

11 javaintegrationtest

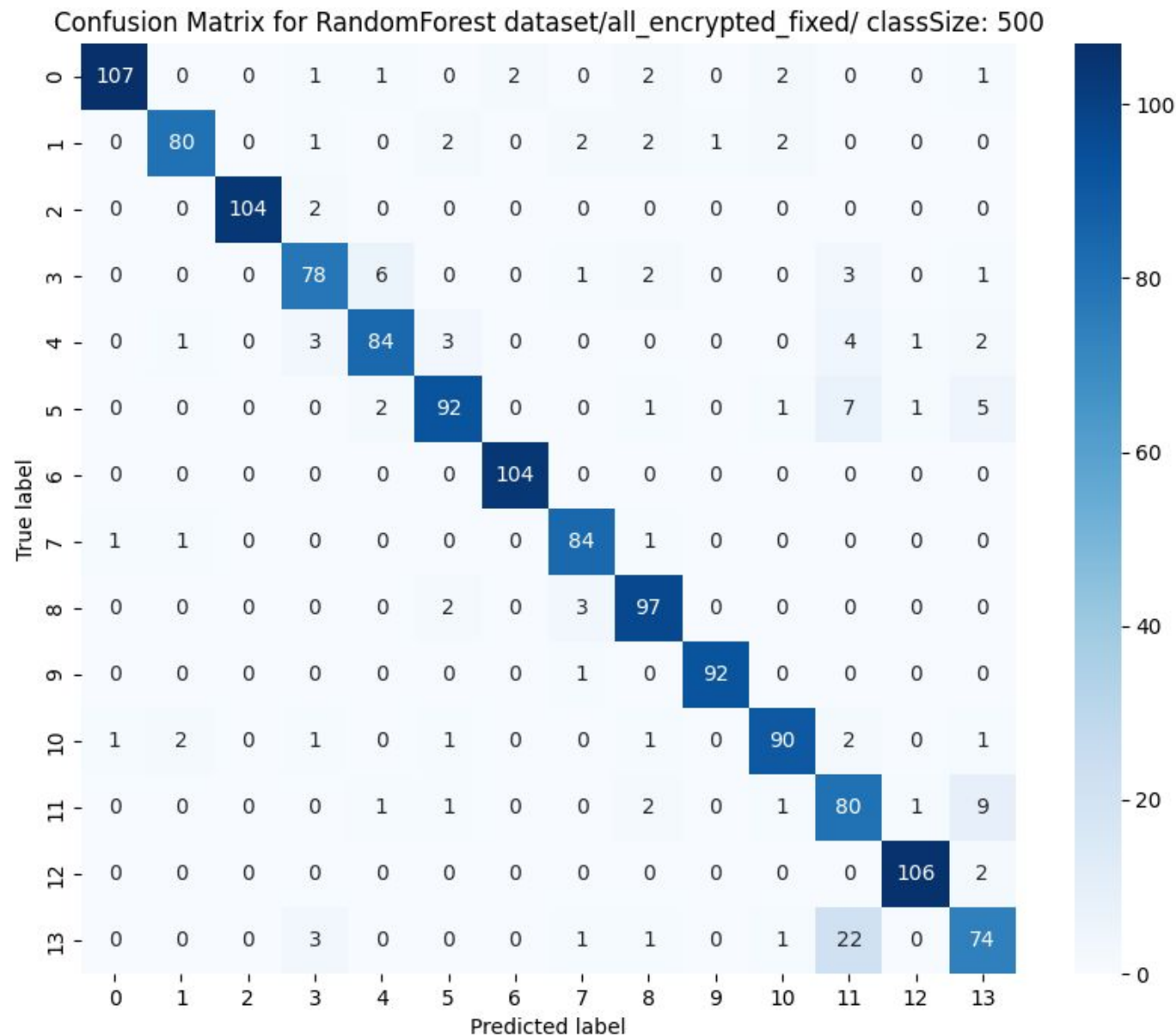
12 javajsf

13 javaunittest

Resultados

0 css
1 html
2 javaconverter
3 javadto
4 javaentity
5 javaimplementation
6 javajasper
7 javajsp
8 js
9 json
10 sql
11 javaintegrationtest
12 javajsf
13 javaunittest

accuracy 0.91
macro avg 0.91
weighted avg 0.91



Próximos passos

- Comparar com outros métodos
- Responder: “Tais métodos são promissores para análise de código fonte ?
- Melhoras deep learning , pareceu muito ajustado

Preparar dataset para publicação

Preparar artigo

Feature Detection in Software: A Source Code Mini-Map Approach

Decoding Software using Sourcecode mini-maps to identify programming Languages and stereotypes

1st Munif Gebara Junior

PCC

UEM - State University of Maringá
Maringá, Brazil
pg55752@uem.br

2nd Yandre Costa

PCC

UEM - State University of Maringá
Maringá, Brazil
ymgcosta@uem.br

3rd Igor Wiese

PCC

UTFPR - Federal University of Technology (Brazil)
Paraná, Brazil
igor.wiese@gmail.com

Abstract—In this paper, we propose a novel approach for feature detection in software systems using source code mini-maps. Source code mini-maps provide visual representations of the structural and behavioral characteristics of code, enabling identification of features such as programming languages, code patterns, and stereotypes. We employ Local Binary Patterns (LBP) to extract texture features from these mini-maps and classify them using machine learning models, including K-Nearest Neighbors (KNN), Random Forest, and Support Vector Machines (SVM). The proposed approach is evaluated on a dataset consisting of various Java-based software components, achieving an accuracy of up to 87% with Random Forest. The results demonstrate the effectiveness of source code mini-maps in feature detection and software analysis.

Index Terms—Feature detection, source code mini-maps, static code analysis, software engineering, code stereotypes, programming languages.

I. INTRODUCTION

Understanding the features and functionalities of software systems is a recurrent task in software engineering, impacting maintenance, evolution, and overall system comprehension. As software systems grow in complexity, traditional feature detection methods become increasingly heavy due to their reliance on dynamic analysis or manual code inspection, which are both time-consuming and prone to errors.

This paper presents a source code mini-map approach to feature detection. Source code mini-maps are visual abstractions of the codebase that encapsulate essential static characteristics, enabling straightforward identification of features. By focusing on static code elements such as programming languages used, code stereotypes (e.g., classes, interfaces, methods), and other code metrics, the proposed approach provides a scalable and efficient solution to feature detection.

The remainder of this paper is organized as follows: Section 2 reviews related work in feature detection and static code analysis. Section 3 describes the methodology of the source code mini-map approach. Section 4 presents the experimental results and analysis. Section 5 discusses the implications and potential applications. Finally, Section 6 concludes the

paper and suggests directions for future research. (REVISAR DEPOIS DE COMPLETO)

II. RELATED WORK

Feature detection in software has been extensively studied, with various methods proposed over the years. Dynamic analysis techniques, such as execution tracing and profiling, have been commonly used but require the software to be executed, which may not always be feasible. Static analysis methods have gained attention due to their ability to analyze code without execution. Techniques involving code metrics, program slicing, and code pattern recognition have been proposed. Visualization tools that represent code structures to aid comprehension have also been developed. However, these tools often lack the integration of multiple static characteristics for feature detection.

Code stereotypes classify code based on common patterns and roles within the codebase. This classification aids in understanding the code's purpose and can be leveraged for feature detection. Our approach builds upon these foundations by integrating static code characteristics into a unified source code mini-map, enhancing the ability to detect features effectively. (CADE AS REFERENCIAS)

III. METHODOLOGY

The proposed source code mini-map approach involves the following steps:

A. Source Code Mini-Map Dataset Generation

The process of generating source code mini-maps begins with the analysis of source code files. Each file is transformed into a visual representation (mini-map), where the characters in the code are mapped to pixel intensities. This transformation retains the structural and syntactic information of the code, making it easier to extract features for classification.

The Python script used for this process categorizes files based on their type (e.g., '.java', '.html', '.js') and annotations (e.g., '@Entity', '@Repository'), before generating fixed-size images (128x128 pixels) for each file. An anonymization step

is performed to prevent reverse engineering by hashing file names and encrypting ASCII characters.

B. Feature Extraction Using LBP

To extract features from the mini-maps, we use Local Binary Patterns (LBP), a texture descriptor that has proven effective in capturing local structural patterns in images. The LBP operator creates a histogram representing the frequency of specific texture patterns in the image, which serves as the input for the machine learning classifiers.

C. Classification Models

We evaluated three machine learning models for the classification of source code mini-maps: K-Nearest Neighbors (KNN): A simple instance-based learning method where the classification is based on the majority class among the nearest neighbors; Random Forest: An ensemble learning method that constructs multiple decision trees and outputs the majority vote of the trees; Support Vector Machine (SVM): A classifier that separates the data points into classes using hyperplanes in a high-dimensional space. We optimized the SVM with a radial basis function (RBF) kernel.

IV. EXPERIMENTAL RESULTS

A. Dataset

We evaluated the proposed approach on a dataset of 500 samples per class, consisting of source code files categorized into 12 different classes, such as 'javaconverter', 'javajsf', 'javajsp', and 'sql'. The dataset was split into training and testing sets (80/20 split), and the classification models were evaluated using precision, recall, F1-score, and accuracy.

B. Results

In K-Nearest Neighbors (KNN) model achieved an overall accuracy of 68%, with a macro average F1-score of 0.67 . The performance was higher for some classes, such as 'javaconverter' (F1-score = 0.93), but lower for classes like 'javaentity' (F1-score = 0.57).

In Random Forest model outperformed KNN, achieving an accuracy of 87%. It provided consistently high precision and recall across all classes, particularly in 'javaconverter' (F1-score = 0.98) and 'javajsf' (F1-score = 0.96).

In Support Vector Machine (SVM) model performed well, with an overall accuracy of 82%. It achieved strong performance in the 'javaconverter' (F1-score = 0.94) and 'javajsf' (F1-score = 0.95) classes, but slightly lower scores for 'javaintegrationtest' (F1-score = 0.69).

V. DISCUSSION

PARECIDO DEMAIS!!! The classification results show that Random Forest was the most effective model, with an accuracy of 87%. SVM also performed well, achieving an accuracy of 82%, while KNN had a lower performance at 68%. The confusion matrices for each model showed that the errors were mostly concentrated in classes with lower representation in the dataset, such as 'javaintegrationtest' and 'javaentity'.

Overall, the use of source code mini-maps, combined with LBP for feature extraction and machine learning classifiers, proved to be a viable approach for detecting software features. The scalability and effectiveness of the Random Forest model demonstrate the potential for applying this method to larger and more complex codebases.

VI. CONCLUSION

PARECIDO DEMAIS!!! In this paper, we presented a novel approach for feature detection in software using source code mini-maps. The method was evaluated using three machine learning classifiers, with Random Forest achieving the best performance. The results demonstrate the viability of using mini-maps for software feature detection and highlight the potential for future work in applying deep learning techniques and expanding the dataset to cover more programming languages and code patterns.

TEM QUE ARRUMAR AS REFERÊNCIAS

REFERENCES

- [1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," *Phil. Trans. Roy. Soc. London*, vol. A247, pp. 529–551, April 1955.
- [2] J. Clerk Maxwell, *A Treatise on Electricity and Magnetism*, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
- [3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in *Magnetism*, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
- [4] K. Elissa, "Title of paper if known," unpublished.