Warsaw University of Technology

Intelligent Information Systems

A.Y. 2020/2021

Plagiarism detector in programming code

Document Report

Authors:
Andrea Murino

Contents

Chapter 1

What is the programming plagiarism?

It is, simply code, using another person's source code and claiming it as your own.

Behind each source code there is lot of work and creative energy, so it's not surprising that they are protected by copyright. As other intellectual properties, is not allowed to spread or plagiarise a third person's code without his authorisation. This apply especially for the so called closed source softwares.

The situation change if we deal with open source program: indeed, there is a scenario in which software development gives definitely more "freedom".
The sense of this opened culture among developers is that they can work to a project together and so develop and better maintain the code in the long term. On GitHub, SourceForge and similar platforms, developer make available the source code to other programmers. Nevertheless usually open source program developers have to conform to specific conditions, such as mention the author and license if they intend to use the code.

Chapter 2

About the code and methodology used

2.1 Intro

With the developed software, we're going to make a **Plagiarism Detector** in Python using **machine learning techniques** such as *word2vec* and *cosine similarity*.

2.2 How to analyze code?

We all know that **computers** can only understand *0s* and *1s*, and for us to perform some **computation** on textual data we need a way to **convert** the *text* into *numbers*.

2.2.1 Word embedding

The process of converting the *textual data* into an array opf numbers is generally known as **word embedding**.

The vectorisation of textual data to vectors is not a **random** process instead it follows certain **algorithms** resulting in words being represented as **position** in space. We're going to use **scikit-learn** built-in features to do this.

2.3 How to detect similarity in source codes?

In order to achieve our objective, we are going to use the basic concept of *vector*, **dot product** to determine how closely two source codes (texts) are similar by **computing** the value of *cosine similarity* between vectors representations of software developer's code.

In order to simplify the project, we will compare source code files that are already at our disposal instead of looking for ones on the internet.

Chapter 3

Going through the code

Let's first import all the necessary modules

Listing 3.1: Importing modules

```
# Importing all necessary modules
import os
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

We are going to use *OS Module* in loading paths of textfiles and then **TfidfVectorizer** to perform *word embedding* on our textual data and **cosine similarity** to compute the **plagiarism**.

We will read all the source code files using **List Comprehension**, this will let us load all the path textfiles on our project directory as shown below.

Listing 3.2: Loading path textfiles

```
# loading all the path textfiles on our project directory
source_files = [doc for doc in os.listdir() if doc.endswith('.c')]
```

Then, we need to create **two lambda functions**, one to convert the *text to arrays* of numbers and the other one to *compute the similarity* between them.
The other two lines will let us vectorise the loaded source code files.

Listing 3.3: Lambda functions and vectorization

```
# lambda function vectorize will convert the text to arrays of numbers
vectorize = lambda Text: TfidfVectorizer().fit_transform(Text).toarray()
# lambda function similarity will compute the similarity between them
similarity = lambda doc1, doc2: cosine_similarity([doc1, doc2])

# let's vectorize the loaded source files
vectors = vectorize(source_streams)
s_vectors = list(zip(source_files, vectors))
```

Finally, there is the main function of our script responsible for managing the whole process of **computing** the similarity among the source codes.

Listing 3.4: The function computing the similarity

```
# The set that will contain the results of the similarity among the source files
score_results = set()
plagiarism_results = set()
threshold = 0.8 # Above this threshold we'll consider the plagiarism

# The actual function that will compute the similarity
def check_plagiarism():
    global s_vectors
    for code_a, text_vector_a in s_vectors:
        new_vectors = s_vectors.copy()
        current_index = new_vectors.index((code_a, text_vector_a))
        del new_vectors[current_index]
        for code_b , text_vector_b in new_vectors:
            sim_score = similarity(text_vector_a, text_vector_b)[0][1]
            code_pair = sorted((code_a, code_b))
            score = (code_pair[0], code_pair[1],sim_score)
            if sim_score > threshold:
                plagiarism_results.add(score)
            else:
                score_results.add(score)

    results = {"score": score_results, "plagiarism": plagiarism_results}
    return results
```

Chapter 4

Results

Running the code, the outcome will look as shown below

Listing 4.1: Results visualization

```
The score of all the comparisons:
('storing_with_structures.c', 'writing_to_file.c', 0.06026568558259773)
('matrix_transpose.c', 'sorting_elements.c', 0.13306636300946817)
('inch_feet_system.c', 'string_length.c', 0.04300389194878646)
('inch_feet_system.c', 'storing_with_structures.c', 0.07959286399554981)
('display_code.c', 'matrx_addition.c', 0.04237189131576213)
('inch_feet_system.c', 'matr_trans.c', 0.09378316181677258)
('matrix_transpose.c', 'string_length.c', 0.24292804184803693)
('inch_feet_system.c', 'writing_to_file.c', 0.022894402149311488)
('freq_of_char_in_string.c', 'string_length.c', 0.15762277002131314)
('freq_of_char_in_string.c', 'inch_feet_system.c', 0.049516908202381464)
('matrix_transpose.c', 'storing_with_structures.c', 0.1531043337821651)
('matrx_addition.c', 'sorting_elements.c', 0.11646460494144935)
('add_two_matrices.c', 'inch_feet_system.c', 0.13314796116462815)
('add_two_matrices.c', 'string_length.c', 0.22928619984297396)
('storing_with_structures.c', 'string_length.c', 0.10087034423029752)
('freq_of_char_in_string.c', 'writing_to_file.c', 0.0760444798661701)
('freq_of_char_in_string.c', 'storing_with_structures.c', 0.10787144095014761)
('matr_trans.c', 'string_length.c', 0.2510520529945792)
('storing_with_structures.c', 'string_concatenation.c', 0.01919499742415373)
('add_two_matrices.c', 'sorting_elements.c', 0.11745748937736197)
('matr_trans.c', 'writing_to_file.c', 0.06181861819256015)
('matr_trans.c', 'sorting_elements.c', 0.14768611540293994)
('display_code.c', 'matr_trans.c', 0.032365747238306354)
('matrx_addition.c', 'string_length.c', 0.22862101531217155)
('add_two_matrices.c', 'matr_trans.c', 0.5038416995629318)
('add_two_matrices.c', 'string_concatenation.c', 0.08405504300881542)
('freq_of_char_in_string.c', 'matrx_addition.c', 0.1430763305623668)
('freq_of_char_in_string.c', 'matr_trans.c', 0.1531928751435721)
('display_code.c', 'writing_to_file.c', 0.14900803046706573)
('display_code.c', 'freq_of_char_in_string.c', 0.057115707405490206)
('display_code.c', 'matrix_transpose.c', 0.05737401911570949)
('display_code.c', 'storing_with_structures.c', 0.013641227873448597)
('inch_feet_system.c', 'sorting_elements.c', 0.024642977341684802)
('sorting_elements.c', 'writing_to_file.c', 0.04986862283603167)
('matrix_transpose.c', 'writing_to_file.c', 0.05058701824676228)
('add_two_matrices.c', 'display_code.c', 0.04731554517439309)
('inch_feet_system.c', 'matrx_addition.c', 0.10388867408760219)
('sorting_elements.c', 'string_length.c', 0.1549246822592314)
('sorting_elements.c', 'string_concatenation.c', 0.04155822754006029)
('string_concatenation.c', 'writing_to_file.c', 0.021324969101189084)
```

```
('matrix_transpose.c', 'string_concatenation.c', 0.0583654938762819)
('freq_of_char_in_string.c', 'sorting_elements.c', 0.47410357388808827)
('matrx_addition.c', 'string_concatenation.c', 0.08367871047437224)
('matrx_addition.c', 'writing_to_file.c', 0.0498547498909309)
('string_concatenation.c', 'string_length.c', 0.36482058023929664)
('matr_trans.c', 'storing_with_structures.c', 0.18572828996017504)
('inch_feet_system.c', 'matrix_transpose.c', 0.07401339130299278)
('add_two_matrices.c', 'writing_to_file.c', 0.048011582643184544)
('add_two_matrices.c', 'freq_of_char_in_string.c', 0.1377866920224639)
('inch_feet_system.c', 'string_concatenation.c', 0.01181943473516005)
('matr_trans.c', 'matrix_transpose.c', 0.9057121360752486)
('freq_of_char_in_string.c', 'matrix_transpose.c', 0.12503803786378717)
('add_two_matrices.c', 'storing_with_structures.c', 0.18352704786510118)
('display_code.c', 'string_concatenation.c', 0.03915024062191028)
('matrix_transpose.c', 'matrx_addition.c', 0.4728838074657197)
('matrx_addition.c', 'storing_with_structures.c', 0.19057266113335183)
('add_two_matrices.c', 'matrx_addition.c', 0.9811201944588239)
('freq_of_char_in_string.c', 'string_concatenation.c', 0.034023383556188155)
('display_code.c', 'string_length.c', 0.14001110554256818)
('display_code.c', 'sorting_elements.c', 0.046456812411309494)
('matr_trans.c', 'matrx_addition.c', 0.5202166370619383)
('add_two_matrices.c', 'matrix_transpose.c', 0.46612831894572937)
('string_length.c', 'writing_to_file.c', 0.06750495155766065)
('display_code.c', 'inch_feet_system.c', 0.02060435608175412)
('matr_trans.c', 'string_concatenation.c', 0.050193719207040306)
('sorting_elements.c', 'storing_with_structures.c', 0.06349814084300548)

Detected plagiarism:
('matr_trans.c', 'matrix_transpose.c', 0.9057121360752486)
('add_two_matrices.c', 'matrx_addition.c', 0.9811201944588239)
MacBook-Pro-di-Andrea:einis andreamurino$
```

From the results, we can read the coefficient of similarity between every couple of source code files, then if this coefficient is above a fixed threshold (in this case was set to 0.8) then we will consider the case of plagiarism.