Documentation
CS 251 Course Project
Red Plag - A Source Code Plagiarism Checker
Team CryptoFox

Introduction

Plagiarism is a demonstration of deceptive conduct, which includes the reuse of another person's work without expressly recognizing the first creator. A genuine illustration of plagiarism is source code plagiarism in programming courses when a student submits a course assignment whose part was duplicated from another student's course assignment. In such a case, the plagiarized source code has been derived from another piece of source code, with a small number of routine transformations. Plagiarism in college course tasks is a progressively normal issue. Source code plagiarism is a simple undertaking to do, yet it isn't so easy to recognize. Here in this project we have attempted to build a robust copy checker which would be immune to common strategies used for making changes in code to make it *look different.*
The most common forms of changes done in plagiarized source code which we have handled in our program are -

A. addition of extra spaces, tabs and newlines
B. reformatting code
C. addition of extra comments
D. changing names of variables and functions
E. modifying constant values
F. changing the order of variables in statements
G. changing the order of statements within code blocks
H. reordering of code blocks
I. addition of redundant functions
J. temporary variables and subexpressions
K. modification of scope

Language Support:

Currently, our Plagiarism Detector supports Plagiarism checking for:

- C++ (Compiler based Tokenizer, better results, but result generation is time consuming)
- Java (Tokenizer based on Syntax)

- Python (Tokenizer Based on Syntax)
- General Languages (Basic Tokenizer Based on Syntax)

Checker Core Logic

We have divided the whole task into multiple sub-parts:
1. Preprocessing
2. Tokenization
3. Exclusion of stub code and base file
4. Similarity Measure and Inferential Analysis of measurements
5. Delivering final insights of evaluated results to user

Preprocessing:
Most of the plagiarism checkers, especially for source code plagiarism that we have come across, are based on textual similarity rather than semantics of the file. For such an approach to work correctly, it's important that differences arising due to whitespaces, writing of a part of code as function, writing extra comments, and minor changes such as restructuring of code should not affect the result. For this we have first scanned the file once and created an auxiliary file with no extra whitespaces (newlines/tabs/spaces). This will be the file that we work with subsequently.

Tokenization:
One of the common ways of hiding a copied piece of code is by changing the variable names, function names, data types, initialized values, and loops. To deal with this, tokenization is one of the methods. Tokenization is the process of conversion of source code into tokens. The tokens are chosen in such a way that they characterize the essence of a program, which is difficult to change by a plagiarist. A simple tokenization algorithm can substitute all identifiers and values with tokens <IDENTIFIER> and <VALUE>, respectively. By further modifications, we can classify identifiers further into types for e.g. string, numeric etc. This eliminates false positives, as well as it takes care of any changes in the variable names. The tokens are chosen in such a way that they characterize the essence of a program, which is difficult to change by a plagiarist.

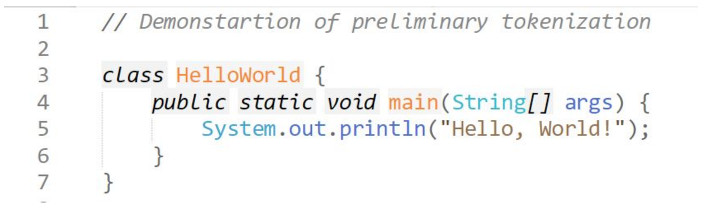
This key concept behind this module is converting source code into lexicons with steps executing lexicon recognition, lexicon categorization, transition-based tokenization, and standard text pre-processing (lowercasing, stopping, and stemming)

Steps carried out during tokenization in general:

- The file is scanned for detecting functions and a dictionary is created with code corresponding to each function mapped to its name and removed from the rest of the code
- Classes are also detected and their names are stored in a list/dictionary with information about their member variables and methods
- Tokens corresponding to each of the functions are generated and their list is mapped to the function name in another dictionary
- Subsequently the remaining file with functions excluded is tokenized and list of tokens stored
- When a token is encountered which corresponds to a function call, tokens corresponding to the function are obtained from the dictionary and appended
- Instances of classes/objects will be replaced by 'class' keyword token to make it independent of the class name
- Variable declarations of any name are replaced by ‘v’ or ‘var’ keyword as the token to make it independent to variable name changes
- Numbers maybe replaced by the ‘no’ or ‘num’ token
- Access modifiers are replaced by single fixed token for all types depending upon programming language
- Keywords, operators, identifiers, builtin methods, attributes and decorators are added as it is in string form as tokens
- Comments, punctuation (*.,(){}[];:*), literals and unnecessary text are ignored
- Information about the user defined classes is also tokenized similarly with fixed keywords for constructors, destructors and operators(overloaded ones mostly) and appended at the end

To execute the above we have used programming language specific lexers and parsers. For this we used the **Pygments** library available for Python. It is a generic syntax highlighter suitable for use in code hosting, forums, wikis or other applications that need to prettify source code. A lexer splits the source into tokens, fragments of the source that have a token type that determines what the text represents semantically (e.g., keyword, string, or comment). There is a lexer for every language or markup format that Pygments supports and that can be used to generate preliminary tokens.

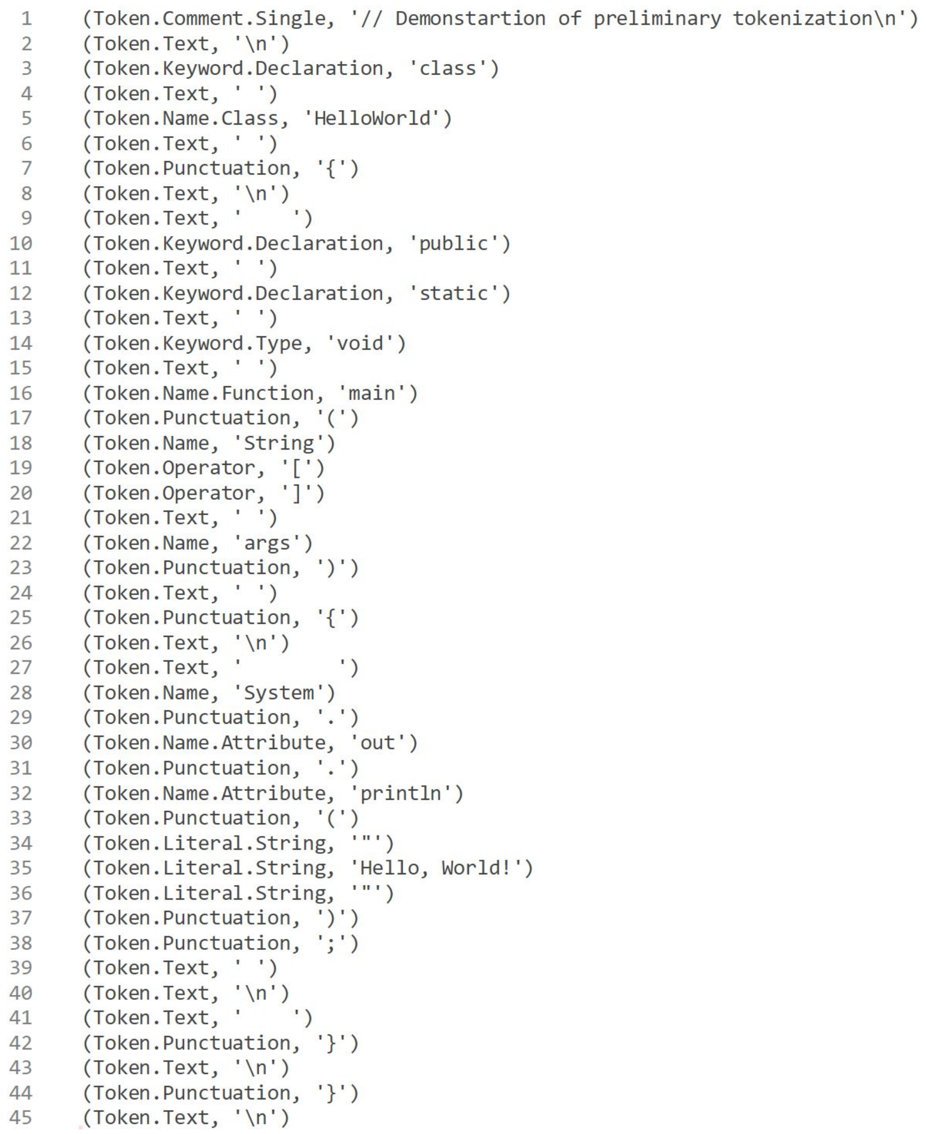
A demonstration -



```java
// Demonstartion of preliminary tokenization

class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Here is the code to be passed through Pygments



```
(Token.Comment.Single, '// Demonstartion of preliminary tokenization\n')
(Token.Text, '\n')
(Token.Keyword.Declaration, 'class')
(Token.Text, ' ')
(Token.Name.Class, 'HelloWorld')
(Token.Text, ' ')
(Token.Punctuation, '{')
(Token.Text, '\n')
(Token.Text, '    ')
(Token.Keyword.Declaration, 'public')
(Token.Text, ' ')
(Token.Keyword.Declaration, 'static')
(Token.Text, ' ')
(Token.Keyword.Type, 'void')
(Token.Text, ' ')
(Token.Name.Function, 'main')
(Token.Punctuation, '(')
(Token.Name, 'String')
(Token.Operator, '[')
(Token.Operator, ']')
(Token.Text, ' ')
(Token.Name, 'args')
(Token.Punctuation, ')')
(Token.Text, ' ')
(Token.Punctuation, '{')
(Token.Text, '\n')
(Token.Text, '        ')
(Token.Name, 'System')
(Token.Punctuation, '.')
(Token.Name.Attribute, 'out')
(Token.Punctuation, '.')
(Token.Name.Attribute, 'println')
(Token.Punctuation, '(')
(Token.Literal.String, '"')
(Token.Literal.String, 'Hello, World!')
(Token.Literal.String, '"')
(Token.Punctuation, ')')
(Token.Punctuation, ';')
(Token.Text, ' ')
(Token.Text, '\n')
(Token.Text, '    ')
(Token.Punctuation, '}')
(Token.Text, '\n')
(Token.Punctuation, '}')
(Token.Text, '\n')
```

Here is the output

This will be passed through the steps shown above to produce the required set of tokens to be used for winnowing.

Although Pygments provides support for most languages, processing these tokens to tackle plagiarism effectively is a language specific process. We have implemented the above described process for JAVA and Python in detail and added basic support for most languages. For JAVA and python, specific programs have been written and it cannot be fooled by any of the above mentioned basic forms of changing plagiarized code. For other programs, it works fairly well but functions calls have not been handled so code redundancy can create a problem with accuracy of results. For C++, a python library **pygccxml** parser and xml generator is used which works on using **CastXML** compiler and adds extra robustness to the program. Steps followed are more or less the same but class identification is more effective but a bit more time taking.

Winnowing:

Once, the code files have been tokenized, we now need a way to compare two files for similarity. One of the popular ways of doing this, is by using winnowing to generate document fingerprints and comparing them.
Winnowing basically has three important steps:

1. **K-grams generation :**
    - It is the most important of the three steps. What our fingerprints represent depends on what value of 'k' we choose.
    - If **'k'** value is too large, we will end up missing a lot of information within the file after winnowing, as for a large chunk of code snippet, we would not have any hash value in the document's fingerprint
    - If 'k' value is too small, not only would it be inefficient to perform winnowing on large files, but this would also mean that we will miss on relative location based information in the document. This will generate more false positives while checking similarity.
    - Therefore, it is important to choose an optimal value of 'k'. In our implementation, based on tests we performed, we have set **k=10** for c++ and python checkers, and **k=15** for java checkers.
2. **Hashing the K-grams :**
    - We have used the 'winnowing' module in Python, which uses the 'hashlib' module to generate hash values for each kgram.
    - By default, 'winnowing' module uses **SHA-1** hashing. We have not changed it.
3. **Choosing Hash Values as fingerprints for the document:**
    - Now, we use sliding window of size 4, on the list of fingerprints, and choose the minimum hash value of the window as one of the fingerprints
    - These chosen hash values, along with the positional information make up the document fingerprint for the document(source code in our case).

We have used the ‘winnowing’ module available in python to perform winnowing and generate fingerprints for each source code file that is to be compared.

Exclusion of stub code and base file:

When students are given some coding assignment, and the problem statement is similar, there is a high possibility of base code such as some predefined class definitions or functions to be the same. In such cases, comparing the files with removing this stub code would result in false positives.
We have used winnowing for generating document fingerprints for each code file. In order to reduce the impact of the stub code, we simply remove all the fingerprints of the boilerplate code file, from the fingerprints of the code files. This ensures that no fingerprints, which are common due to the same stub code, of the two files match. Now, only those fingerprint matches are counted in results which are due to code other than the stub code.

Similarity Measures and Inferential Analysis of measurements:

Once, winnowing has been performed on the tokenized codes to generate document fingerprints, we now compare the fingerprints of the files pairwise, to find the common fingerprints of the two source codes.
In order to calculate the result in the form of percentage of similarity (percentage of plagiarised code detected), we simply take the ratio of the matched fingerprints to the minimum of the total fingerprints of the two tokenized source codes. Choosing minimum of the two, is a fair assumption, as in case if a file contains only a part of code from the other file, it is also a case of plagiarism. Choosing minimum ensures, high percentage of plagiarism is returned in such cases.

Delivering final insights of evaluated results to user:

By making pairwise comparisons, we obtain a ratio (a decimal value in 0-1) which represents the percentage similarity. We perform this evaluation on every possible pair of source code files and produce this result in the form of a similarity matrix.
In order to make it easier for the end user to understand the result, we also generate a pictorial representation of the similarity matrix, which is displayed to the user as soon as the processing is over. We have used the ‘pandas’ module to generate csv file and ‘matplotlib’ module to generate pictorial representation.

Frontend and Backend
Angular and Django Frameworks

Working With Frontend:

We have used the **Angular 10 framework** for preparing the frontend and user interface. It has components for Homepage, User Dashboard, Change Password Page, Login and Register Pages.
We have used **AuthGaurd** to prohibit illegitimate access to the data of the users, i.e. user component can't be accessed without logging in, and also once logged in user will be restricted to Dashboard and Change Password pages, any attempt to access any other page would result in forced log out.
Services fetch data from the backend REST API, there are different services for login, register, comparisons, downloading and deleting files. Also, we have used **Interceptor** to intercept all the HTTP requests made to the server and insert authentication token into the header sent to the backend in every request.
Styling has been done in SCSS for various components, and we have also used a bootstrap theme to improve aesthetics.

Usage:

- **Login and Register:**
  - Anyone is allowed to register with us, using an Email ID, and password.
  - If a user already exists with the same email id, it will throw a message with "error"
  - Once a user is successfully registered or an existing user successfully logs in, a JWT Token is generated at the backend which allows us to validate session and allow the user to access Authenticated services offered by the backend, and he/she will be redirected to a user specific dashboard.
- **User Dashboard:**
  - User dashboard displays all the files that the user has uploaded on the server, an option to upload new files to the backend, functionality to download or delete files which are available on the backend.
  - In order to make a comparison of source codes, a user needs to upload a compressed file of .zip/.tar/.tar.gz format to the server. The generated results will be available for download in a .zip format, which will have a graphical representation, and a csv file containing the results.
  - Results will also be displayed on the user dashboard.
  - Users also can optionally select a cpp/java/py file as boilerplate code(stub code) to improve the results.

    - We have also given an option to change the user password.

Backend

We have used **Django 3.1 REST Framework** to create backend API.
It has two apps, one manages user profiles and provides API endpoints for login, register and change password, and the other manages user files, and provides endpoints to upload, download, delete and compare files. Both have separate models and serializers. All the APIs other than for login or register have restricted access to authenticated users. This is ensured by using **JWT Authentication**, and validating it each time a request is made to the server.

API Endpoints:

- **/api/signup/:**
    - Allows to make HTTP POST Request, with credentials ‘email’ and ‘password’ in the request body in JSON Format.
    - Validates the credentials, and checks if the user already exists, in case of a new user, saves its credentials, and generates a JWT Token, which is returned to the client in the response header.
    - In case of an error, returns “Bad Request”.
- **/api/signin/ :**
    - Allows to make HTTP POST Request, with credentials ‘email’ and ‘password’ in the request body in JSON Format.
    - Validates the credentials, and generates a JWT Token, which is returned to the client in the response header.
    - In case if credentials are wrong, it raises an error.
- **/api/change/:**
    - Allows to make an HTTP PUT Request, with credentials ‘old_password’ and ‘new_password’ in the request body in JSON Format.
    - It validates the ‘old_password’ and updates the password.
    - If the ‘old_password’ doesn’t match the user’s password, it returns a ‘Bad Request’.

- **/files/upload/:**
    - Allows to make HTTP POST Request, with data having a field ‘file’, which maps to the file to be uploaded by the user.
    - Uploaded file is stored in ‘/userfiles/{userid}’ for each user. Since, each user is assigned a unique UserID (UUID Type) while registration, all files of a user are always stored in his/her folder. If a file with the same name already exists, a random string gets appended at the end of the filename.

- **/files/download/:**
    - Allows to make HTTP POST Request, with data having a field ‘filename’, which is the name of the file to be downloaded, in JSON Format.
    - Searches the given filename in the user’s folder, and if it exists, returns it in the response, ‘content-disposition’ has filename and tells the client that response has an attachment with it. ‘Content-type’ in the response tells the user about the file type.
    - In case if the requested file is not found, returns a NOT FOUND status in response.
- **/files/compare/:**
    - Allows to make HTTP POST Request, with data having a field ‘filename’ and ‘boilname’, which is the name of the compressed file having code files to be compared, and boilerplate filename(optional), in JSON Format.
    - These filenames could only be those files which the user had uploaded previously on the server.
    - It invokes the checker by passing the parameters (file path for boilerplate and code files) , and generates results.
    - Generated results are sent to the client in response, same as in the case of download API.
    - In case, if file formats are not correct it returns ‘BAD REQUEST’ as the response status.
- **/files/lists/:**
    - Allows to make an HTTP GET Request, and returns an array of strings, containing a list of all the files that were uploaded by the user on the server.
- **/files/resim/:**
    - Allows to make an HTTP GET Request, and returns the graphical representation of the generated result in the response.

Copyrights:
This Project uses Django and Angular Frameworks, and many modules like Pygments, Pandas, Pygccxml, castxml, matplotlib, winnowing etc. which are open source, and have been used for educational purposes only.