Progress Report

Name	Claire (Yumeng) Luo	Sang Jun Chun
UNI	yl4655	sc4658
Team Name	Imposters2	

Topic: Code Similarity Detection

Changes

We tried feeding processed jar files into SAFE but received exceptions. After carefully looking into its source code and tried fixing multiple areas to allow reading for the jar file, we got stuck on the r2pipe command stage. The binary file processing for SAFE is conducted through r2pipe which outputs the following information when reading target java file:

```
{'core': {'type': '', 'file': '../../4156/assignments/4156-PublicAssignment/hw1/target/hw1-0.0.1-SNAPSHOT.jar', 'fd': 3, 'size': 22572, 'humansz': '22.0K', 'iorw': False, 'mode': 'r-x', 'block': 256, 'format': 'any'}}
```

It is not able to detect the type of “jar” files. There are not enough documentations on r2pipe and whether “jar” file is accepted and if so which command should be used. So we have decided to switch to DeepSim as our second binary similarity detection tool.

We changed the tools that we would be using. The final list (and all tested using sample code) is:

- DeepSim: binary code similarity detector
- GEMINI: binary code similarity detector
- Plaggie: source code similarity detector
- Copyleaks: (online) text difference detector

The public repository for the 4156 individual project on GitHub went private before we could download the code. We will be using Anthony Krivonos’ repository instead, which should contain enough submissions.

Research Questions

The following research questions examine three different approaches used in code similarity detection: source code similarity detection, binary code similarity detection, and text difference detection. (In using these tools, detecting a high degree of similarity, and thus a plagiarism, would yield a positive result.)

- RQ1: How effectively can code similarity tools detect different degrees of code plagiarism? (i.e., are the tools susceptible to producing false negative results?)
    - For instance, would it detect plagiarism if only the variable names/values are changed?

    - Experiment idea: Modify a code segment by changing the overall code structure and variable names/types/values, or adding/removing irrelevant lines. Time-permitting, rewrite the code while maintaining only the logic.
- RQ2: How do different types of code similarity tools react to original works addressing the same problem? (i.e., are the tools susceptible to producing false positive results?)
    - Experiment idea: Compare the group members' original submissions for the individual project for COMS4156. Also, for more data, we may consider user submissions to LeetCode problems, where different logics are used to address the same problem.
- RQ3: Which approach, among examining source code, binary code, and text difference detection, demonstrates particular effectiveness for detecting code academic plagiarism?
    - Our main research question: we will be looking at a school assignment that we are very familiar with: students' individual projects for COMS4156 at Columbia University

<u>Challenges</u>

- Can the proposed tools run smoothly on a team member's laptop (Since some tools are not maintained regularly and may require a specific version of an environment that's obsolete)?
    a. For deepsim, we were able to run the tools on processed test data but the provided "encoding.jar" file for encoding is written in JDK 9 and obsolete. To satisfy the need of our project, there are some hard coded sections (data path, output type) and some incomplete if statements that we wish to change. The source code for this jar file is given but we are unable to compile this file due to an incompatible version of the "com.ibm.wala" library. After rewriting a bunch of library calls to newer versions, we were able to compile the encoding.jar file and use it to generate encodings of test data.
    b. For Plaggie, we have done a test run on 2 files and we were able to get meaningful results
- There are two challenges associated with student submissions available on GitHub. While we were able to collect about 35-40 submissions for the 4156's first individual project assignment, we don't expect all submissions to be complete, nor contain the same version. That is, some versions may be for assignment 1 while others may be for assignment 2 or 3. Specifically, we do not want submissions for the third assignment, which implements data persistency.
    a. Unless we manually inspect and grade each submission, it would be difficult to know whether the submissions fully meet the assignment requirements. Instead of *grading* each submission, we will run them without checking and look for any outliers. Potentially, we can check with Professor Kaiser to confirm whether these outliers align with the grade they received for the project.
    b. To specifically collect code for assignment 1, we will rely on the commit messages and change history, as well as the date of submission (The due date

for the first assignment was Sept 24, 2020. Time flies!) in case of unclear commit messages.

- How would we clean data? For instance, how would we merge the entire project into 1 java file?
    a. For the binary tools, merging source code into 1 file is not necessary because the executable links all files in compilation. But the encoding generated is still in separate files. 1 encoding for each function in each class.
    b. For the text diff tool, compilation doesn't matter, we can just append them into 1 file.
    c. While we expected having to rewrite the java files into a single large class with inner classes, Plaggie, our source code tool, offers a convenient feature allowing users to compare by directories.
- What is the effect of the code skeleton? Would its existence alone mark all code pairs as similar for similarity detection tools?
    a. For text diff tools, since code is not required to compile, we can simply remove the given sections
    b. For Plaggie, there was given an option to specify the given code skeleton such that the scores won't be affected.
    c. For binary tools, still need to run similarity score between 1 member's code and skeleton alone
- How to define plagiarism, and how to *simulate* it?
    a. If the 2 code of interest can be converted into each other using only the following type of changes, then they are considered as a plagiarized pair:
        - Lexical changes of formatting, layout modifications
        - Identifier renaming
        - Structural changes (such as for to while, insert/delete of statements)
    b. Otherwise the code pair will be considered as a non-plagiarized pair.
    c. In answering RQ1, we need to compare plagiarized code, but we do not know which, if any, of the submissions would be applicable. Currently, we plan on *simulating* plagiarism by only slightly modifying existing works, but generating many modified copies this way will be a challenge given that we have limited time to complete the project.

Additionally, when setting the threshold for similarity scores in an actual experiment, it is important to consider the effect of the given code skeleton. Therefore, the threshold for non-plagiarized code will be set as the similarity score between one group member's code vs the code skeleton.

Demo

Since our project focuses on comparing existing tools, our *demo* will focus on describing the design of our experiments and the obtained results. Our demo is structured as follows:

1. Introductory elevator pitch (~1 min): We will present our topic, the comparison of various code similarity tools, the research questions posed by the project, and the motivation

behind it, which is finding the best code similarity detection approach for catching academic plagiarism. The three different approaches are: source code similarity, binary code similarity, and text difference detection. Finally, we will present the research questions posed by the project.

   a. RQ1: How effectively can code similarity tools detect different degrees of code plagiarism?
   b. RQ2: How do different types of code similarity tools react to original works addressing the same problem?
   c. RQ3: Which approach, among examining source code, binary code, and text difference detection, demonstrates particular effectiveness for detecting code academic plagiarism?

2. Experiment and data: We will cover the design of our experiments, and briefly discuss the data we used, and how we cleaned it. Also, we will discuss how we, whenever necessary, modified the tools so they could be compatible with our test data.
3. Results and analysis: We will present an illustrative slide summarizing the experiment results, and analyze them to answer the research questions posed. We will conclude the demo with the main takeaway: which of the tools we examined performed the best in detecting academic plagiarism?
   a. Possible points to cover in our discussion of the results:
      i. The percentage of submissions the tools identified as plagiarized
      ii. Precision, recall score for each tool
   b. Possible points to cover in our analysis:
      i. What features do code plagiarized pairs have in common for them to be identified by all tools?
      ii. What features of each particular tool are better at detecting than others? What type of changes are likely to be overlooked by tools?
      iii. What do false positives cases have in common? (RQ1 and RQ 2)