TEAM 101- Plagiarism Detector

- Monica Malkani
- Prachi Ved
- Ayush Shukla
- Preyank Jain

Problem Statement:

- Plagiarism is a major concern at many Educational Institutions. It is often seen that the students share their work amongst themselves and produce different versions of the same work. As per the requirements our system needs to work for two kinds of situations:
  1. One version derived from another using copy-then-modify, or
  2. Developed collaboratively without permission.

- The system should not be accessible to everyone, only the intended users like Professors and Teaching Assistants should have access to the application. In order to facilitate this, only the administrator should be allowed to add or remove users.

- The application should detect following types of similarities for Python programming language:
  1. renaming of variables
  2. code modularization (one function broken into multiple functions)
  3. code move over (sequence of function/statements altered)
  4. Identical comments
  5. an overall plagiarism determinant from above mentioned strategies. (Weighted average)

- The comparisons might be performed across various student submission folders having multiple files with different directory structures.

- The determination of the overall result should be comparable with standard plagiarism detectors like Moss.

- The results must display the files in order of severity of plagiarism.

- The user should be provided with the option of downloading the report associated with the two plagiarized files.

- The downloaded report must highlight the similar content in the two plagiarized files, or give a detailed explanation for the same.

- The system should be implemented as an interactive web application in order to facilitate remote and easy access of the application.

- The application should be easy to use for any user.

Overview of Results:

The system is an interactive web application and facilitates remote and easy access. The application is easy to use and is accessible only to those users that have been registered by the administrator.

The basic functionalities achieved is that application can find plagiarism between the student's python assignments based on Variable Change, Code Modularization, Code Move Over, and Identical Comments. It also returns a concise result that combines all the strategies into one called Weighted Average.

Two sophisticated comparison strategies were used – Levenshtein distance and Cosine Similarity. As the structure of python code was very important to determine the different types of plagiarism, we used ANTLR to generate Parse tree and Abstract syntax tree for the python code.

It was observed that Levenshtein distance gave good results in determining the plagiarism of the form – variable change, code move over and identical comments, whereas cosine similarity was useful in detecting modularization-based plagiarism. We observed that in some cases results given by cosine similarity for plagiarism techniques other than Code modularity had deviations.

Apart from the basic functionality our application provides additional functionalities(stretches) as mentioned below:

- Multiple student submissions as well as two particular submissions can be compared.
- More than one comparison strategy can be used on demand.
- Weighted Average results are comparable with MOSS.
- Results can be sorted into Plagiarized and Not Plagiarized.
- Results can be downloaded for any comparison made.
- Downloaded Report includes highlighted line similarity / detailed analysis of plagiarism case.

8 student submissions with src and test folders were created in order to validate the correct functioning of comparison strategies (All these submissions can be found in the project folder by the name of Sample_Submissions) -

- Code modularization was present in sorting submissions of Student 3 and Student 4.
- Code move over was present in searching submissions of Student 1 and Student 3.
- Variable change was present in searching submissions of student 5 and student 7.
- Plagiarism present in split files were present in sorting submissions of student 6 and student 7.

We had 210 issues created overall and all of them have been completed i.e. none in Backlog. Also attached is the stats of when the issues were resolved.

The quality of the code is upto the mark. Sonarqube is used to support these claims.

- The code coverage achieved by unit tests is 90.6% there were 42 unit tests written for the same.
- There are few bug smells.
- Maintainability and Security is graded A.

The team also felt that unit testing could have been done more exhaustively directly on classes having comparison for checking even more subtle cases of plagiarism.

Development Process:

We followed the Agile software development life cycle. The development phase was divided into three phases-

- During Phase A the emphasis was on coming up with a strategy on how the team would work, and how the iterations would be delivered for each sprint. The team came up with mock-ups for the user interface and also the use cases were drafted.

- During Phase B the emphasis was on the design. The classes, Interfaces, and design patterns that would be used were identified. Also, class diagrams and sequence diagrams were made.

- During Phase C implementation was started and development of the application was divided into 3 sprints –

    1. In sprint 1, the emphasis was on setting up the environment that would allow unhindered development to all the team members. And getting the basic application up and running with just a simple login page.
    2. In Sprint 2, our focus was on to complete all the UI and get the algorithm for various comparison strategies working on demand.
    3. In Sprint 3, we wrapped up the project by adding the functionality to download the report and highlighting the difference between the files, along with verifying our results to that of MOSS.

Since we could not meet in person all the time, the team held daily scrum meetings on slack/whatsapp to discuss the progress made thus far. We focused on three questions:

1. What we did the previous day?
2. What we planned to do the next day?
3. Any challenges the team members were facing?

We met the TA's during sprints to obtain insights about some of the requirements that were ambiguous. The team had a very insightful session with the professor during the planning phase, where some of the teams asked certain questions to professor (virtual client) which helped to obtain an understanding of how requirement gathering from client is done.

We had biweekly sprint reviews in which we got feedback from the instructors about the progress we had made during that sprint. We discussed any challenges that we faced during that sprint and we also discussed our plan for the next sprint, in short.

All the code that any team member wrote was peer-reviewed by other team members as well which not only helped improve the quality of the code but also made sure that all the team members understood all parts of the application and could give any inputs regarding the same.

We made use of Git approval process for merging the pull requests where every team member received email notifications once a request was made. This ensured that the code was reviewed by a team member before it was merged. Also, we used Jenkins and Sonarqube to ensure the code quality and test coverage.

To keep a track of the progress of development team made use of JIRA and smart commits.

Some challenges that the team faced, were:

- Everyone was new to coding in spring-boot, so it created a bit of a challenge.
- None of the team members had any experience using continuous integration thus setting up the environment posed some challenges.
- We also made use of AWS S3 buckets to store user uploaded files, and coding that entire module in Java was a tedious task.
- We wanted a grammar where we could get all the function definitions of a file. This needed some tweak to the grammar generated by ANTLR. Understanding this grammar was also a major roadblock that we faced.

<u>Retrospective of the Project</u>:

What did we learn?

1. We got a good practice with Java.
2. Gained insight into the field of dev-ops.
3. Managing software from start to end was really challenging, the structure of the course work made it easy.
4. We learnt a lot about AWS S3 in the bargain.
5. Always trying to achieve more than what is expected is a nice attitude to have, which this course made sure we learnt.
6. We learnt a lot from our peers, and their experiences.

What did the team like best?

1. Lots of learning.
2. It was good to have a hands-on experience with the latest industry standard tools for maintaining code quality and continuous integration.
3. We really liked that the application scope that spanned across Java development, database integration to cloud deployment.
4. Working in a team with people having different skill sets was helpful. We learned a lot through the sprints, overcoming each other’s shortcomings.
5. The bug hunt process really helped identify the bugs that the team missed out.
6. We really liked that the project’s focus was on the backend, code structuring and not mainly on the UI.
7. Overall it was an enjoyable experience where we got to manage our entire project from planning to deployment.

What did the team like worst?

1. A lot of time was spent resolving Jenkins build issues. Jenkins sometimes behaved abruptly and clearing workspace always was an overhead.
2. It would be good if TAs were approachable for minor DevOps related issues.

What needs to change in the course to support a great experience?

The course structure is great. The learning curve is good. Learnt a lot. Nothing that we would want to change.