Automated Plagiarism Detection System

Priyansh Gupta
IIIT-HYDERABAD
priyansh.gupta@students.iiit.ac.in

Abstract

Plagiarism has been one of the major issue in colleges all over the world. Because of large number of students, the manual evaluation and identification of plagiarism can be a very difficult and time consuming task for the course instructors. Hence it will be very useful for honest evaluation of students if we have a software system which can automatically detect and identify plagiarism. It would save a lot of time of course instructors and teaching assistants and would also bring some honesty inside students towards their courses.

1. Introduction

1.1. Problem

Plagiarism has been a long standing issue in the academics and it is found statistically that in most of the institutes around 50 to 80 percent [6] undergraduates plagiarise at least once during their academic period of undergraduate degree. Because of this only, academic instructors suspects their students and need to verify whether they have done their assignments on their own or have plagiarised to complete them. Now we know that it is not possible to manually check all the submissions as there are large number of students enrolled for a particular course and also there are multiple assignments with a single course. So if you do this manually then it will be very time consuming. Hence in this paper, the goal is to come up with a software model which can actually assist instructors in checking plagiarism. Also in addition, we would like our software to assist students also to check whether their submission format matches with the required one or not.

1.2. Challenges in designing software

First of all we have to design a software which can detect plagiarism in 3 kinds of submissions: handwritten , programming assignments(codes) and graphical submissions (like diagrams , flowcharts, etc.). Next challenge is to decide that how should we give similarity score between two submissions because we both under-scoring and over-scoring is unwanted as in case of under-scoring we won't be able to detect plagiarism in most of the cases and in case of over-scoring, software may detect plagiarism in unrelated submissions also. Next challenge that we face is that many students try to bring some transformations in copied assignments to make their submission look a bit different. This makes it hard to check for plagiarism. The other challenge is that in handwritten assignments, there is lot of variation in hand writing of students and again activities like bringing transformations in copied assignment also exist here.

1.3. How will this software assist

The software will use some techniques of computer vision and artificial intelligence to detect plagiarism automatically. Software will take input of assignment submissions and will compare them and will give similarity score between each submission as an output, based on which instructor or TA can identify and penalize copy cases. The software will also provide a functionality where the instructor can design a specific submission format and students can simply check whether their submission format is correct or not through this software. I'll name this software model as WePlag.

2. Literature Review

2.1. Handwritten Plagiarism

The biggest challenge in our software designing is building a system which can actually find the similarities between two written documents. So firstly, the system have to perform text recognition from the images, so this means that our system must have access to a dictionary(database) where all the words of different-different languages must be stored. Secondly, Praveen Krishnan and CV Jawahar proposed in one of their paper [7] that a data-set consisting of 1 million handwritten word images(case insensitive) is required for training handwritten word images. Recognition of text in this data space is possible in present using CNN [1](Convolutional Neural Network) algorithm. In the same paper [7], the document similarity is defined as symmetric

distance between the best word matches across the document as follows:-

$$S_N(D_i, D_j) = \frac{1}{|D_i| + |D_j|}\left(\sum_{w_k \varepsilon D_i}^{n} \min_{w_l \varepsilon D_j} d_{kl} + \sum_{w_l \varepsilon D_j}^{n} \min_{w_k \varepsilon D_i} d_{lk}\right) \quad (1)$$

Now to reduce the exhaustive matches, K-dimension trees can be be used. Some other concepts that are important to read about for this technology is word-spotting which is wonderfully explained in papers: [2] and [8].

2.2. Plagiarism in programming assignment

In the paper, [5], the authors designed a java based software called BPlag that models the behaviour of a program using three identified aspects: data and it's relations; the transformation of data through arithmetic operations; and interactions with the execution environment. The authors divided the BPlag into two phases. Firstly assignment submissions are analyzed to derive an approximate behavioural representation. Secondly the behavioural representation of all programs are analyzed for similarity. The figure below represents the overview of this process:-

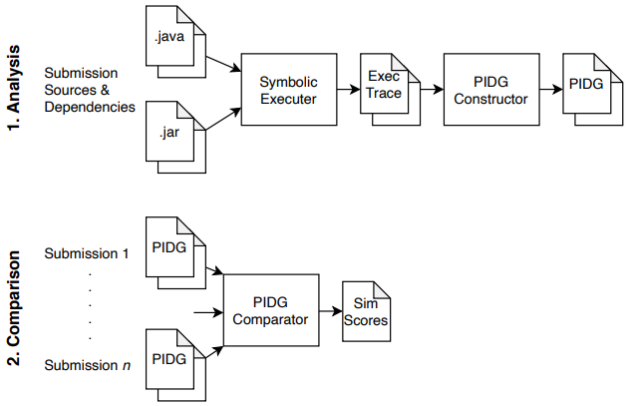

Figure 2.2 BPlag phases

This is decomposed into three main components:

- A symbolic execution tool to record and extract a program's execution behaviour.
- A component to construct the behavioural representation of a program as a set of Program Interaction Dependency Graphs[4](PIDG).
- A component to evaluate analyse the similarity of two programs by comparing PIDGs.

2.3. Submission Format Checker

This part is pretty easy. The software will simply just allow the instructor to design a submission format and this format(whole directory) will be stored in a tree datastructure. After this software can simply just compare the two trees and will give True as output if both trees are exactly same and otherwise, False.

3. Software Architecture

My software design for WePlag will follow the following sequence:

- Use cases(by displaying Use case diagram)
- Defining classes required and private and public methods.
- Relationships between sub-modules or classes (including inheritance, abstraction, etc).
- How software will access database.
- Computation or Algorithm involved

3.1. Use Cases

The most basic model of software have 7-8 use cases which are shown in the diagram below:-

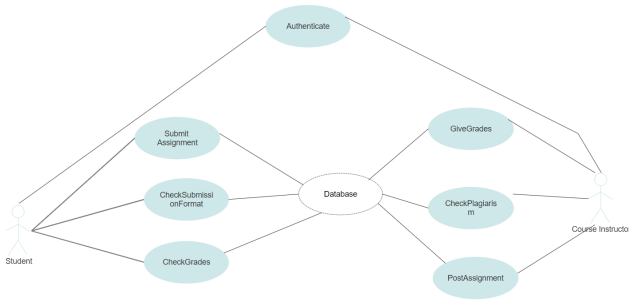

Figure 3.1 Use-Case-Diagram

3.2. Classes

List of classes defined:-

- MAIN class: Will have all the methods for use cases specified above.
- USERS : have two child classes Student and Instructor.
- COURSE: have methods and fields about a course.
- PORTAL: it is class for portal of assignments, it has methods to store submissions in database and can use plagiarism checking classes methods.
- BLACKBOX classes: classes for checking plagiarism and have methods that were discussed in literature section.
- IIIT-H DATABASE: This class have methods that deal with data sets that are used by blackbox classes for checking plagiarism.

The below UML diagram gives a complete overview about all the classes(and their methods) and their relationships:-

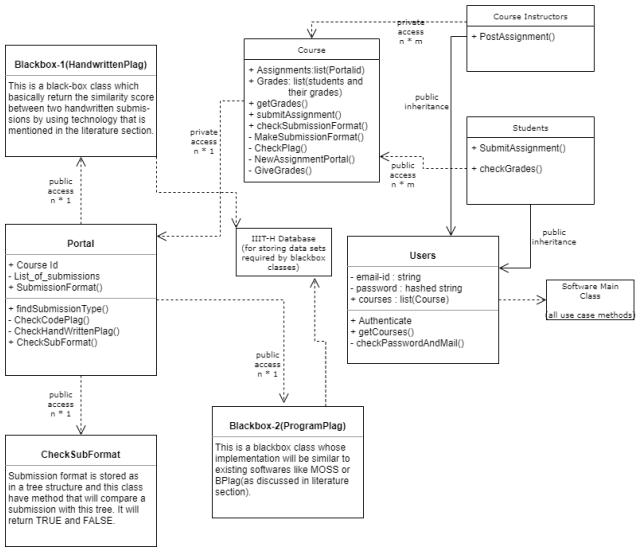

Figure 3.2 UML-Diagram

3.3. Relationship between different classes

- ”Users” have methods like authenticate which are common for both ”Students” and ”Course Instructor”, hence I made these two as child classes of ”Users”.
- ”Student” class can access all the public fields and methods of ”Course” class.
- ”Course Instructor” can access even the private information of ”Course” class and have authority to change grades, check plagiarism and post assignment.
- ”Course” can create new instance of ”Portal” class and use methods for checking plagiarism and setting submission format for the particular assignment portal.
- ”Portal” can access all the classes where most of the computation and algorithm are used on assignment submissions.

3.4. How database will be used and accessed

For WePlag model, there is need to have two separate databases. One of the them is a live database which stores the information about instances of classes like ”Students”,”Course Instructor”,”Course” and ”Assignment Portals”. This live database can be accessed by using the methods specified in the previously mentioned classes. The other database that my software needs is the one which will store data-sets which are used by computer vision and machine learning techniques that the software will be using for checking plagiarism. To access and change this database, I have defined a separate class: ”IIIT-H Database”. The blackbox classes can also access this database indirectly by accessing the methods of ”IIIT-H Database” database.

3.5. Computation and Algorithm

In the methods of my blackbox classes, I will be using techniques which I briefly described in the literature section. For example, In handwritten plagiarism checker, I am using convolutional neural network algorithms which can be optimized using machine learning algorithms like Genetic Algorithm, Particle Swarm Optimization, Simulated Annealing and Harmony Search[3]. Now also after performing word-matching I will perform some computation for evaluating similarity score. In case of programming assignment plagiarism checker, I need to perform some special computation like BPlag also does[5] to account for some transformations that are brought in the copied submission.

4. Conclusion and future work

So the software model that is designed above, have best performance of 0.8 mAP on the word-spotting on popular IAM dataset and plagiarism checker that we are using (similar to BPlag) can reasonably account for transformation in plagiarised submissions. This shows that the product built by following the design mentioned in this paper, can be fully functional in terms of it’s usability and advantages that it will provide. Several new techniques in machine learning and computer vision have opened scope for improvement in the handwritten plagiarism detector while I personally don’t think that there is any need to come up with better strategy for designing programming plagiarism checker apart from just small optimizations (the current ones are good enough).

References

[1] Convolutional neural network algorithms.

[2] G. A. F. A. V. Almaz´an, J. Word spotting and recognition with embedded attributes. *PAMI*, 2014.

[3] V. Ayumi and M. I. Fanani. Optimization of convolutional neural network using microcanonical annealing algorithm. *ICACSIS*.

[4] H. Cheers and Y. Lin. A novel graph-based program representation for java code plagiarism detection. *Proceedings of the 3rd International Conference on Software Engineering and Information Management, ser. ICSIM ’20*, pages 115–122.

[5] Y. L. Hayden Cheers and S. P. Smith. Academic source code plagiarism detection by measuring program behavioural similarity. *arXiv:2102.03995v1*, 2021.

[6] M. Joy and M.Luck. “plagiarism in programming assignments? *IEEE Transactions on Education*, 42(2):129–133, 1999.

[7] P. Krishnan and C. Jawahar. Matching handwritten document images. pages 4–5.

[8] H. C. R. Manmatha, R. Word spotting: A new approach to indexing handwriting. *CVPR*, 1996.