Summer Fellowship Report

On

Plagiarism Detection Algorithm

Submitted by

Aryan Gupta

Under the guidance of

Prof. Prabhu Ramachandran
Department of Aerospace Engineering
IIT Bombay

September 18, 2021

Acknowledgment

The internship opportunity I had with FOSSEE IIT Bombay was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me through this internship period.

I am highly indebted to my project mentor Mr. Ankit R. Javalkar for his continuous support, supervision, motivation and guidance throughout the tenure of my project in spite of his hectic schedule who truly remained the driving spirit in my project and his experience gave me the light in handling this project and helped me in clarifying the abstract concepts, requiring knowledge and perception, handling critical situations and in understanding the objective of my work.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives.

Sincerely,
Aryan Gupta

Contents

Chapter 1

Introduction

Yaksh is an Online Test Interface for Conducting online programming quizzes. It supports various programming languages like :- C, C++, Python and simple Bash.

Users can solve any questions by using these languages. Yaksh uses test cases to test the implementations of the students. It also supports simple multiple choice questions and file uploads so that users can easily submit their code.

Not only can you practice the questions, you can also conduct a programming quiz that supports various languages. There is a separate moderator section for this where a moderator can create questions,quizzes and courses. Yaksh provides various programming courses created by moderators to be enrolled by the students.

Chapter 2

Plagiarism Detection Algorithm

2.1 Objective

To develop an algorithm which can be used in Yaksh for plagiarism detection .

2.2 Technologies Used

2.2.1 Python

Python is an interpreted high-level programming language for general-purpose programming. Python is also Open Source and used in almost all fields of Computer Science ranging from Scientific Computing, Data Science, Machine Learning to Desktop GUI Application and Operating Systems.
Since Django and Yaksh are written in Python, Python 3.6 is used for all of the back-end coding.

2.3 Proposed Algorithms

2.3.1 Simple Winnowing

We started with an existing simple winnowing algorithm . In that algorithm the following steps are used :

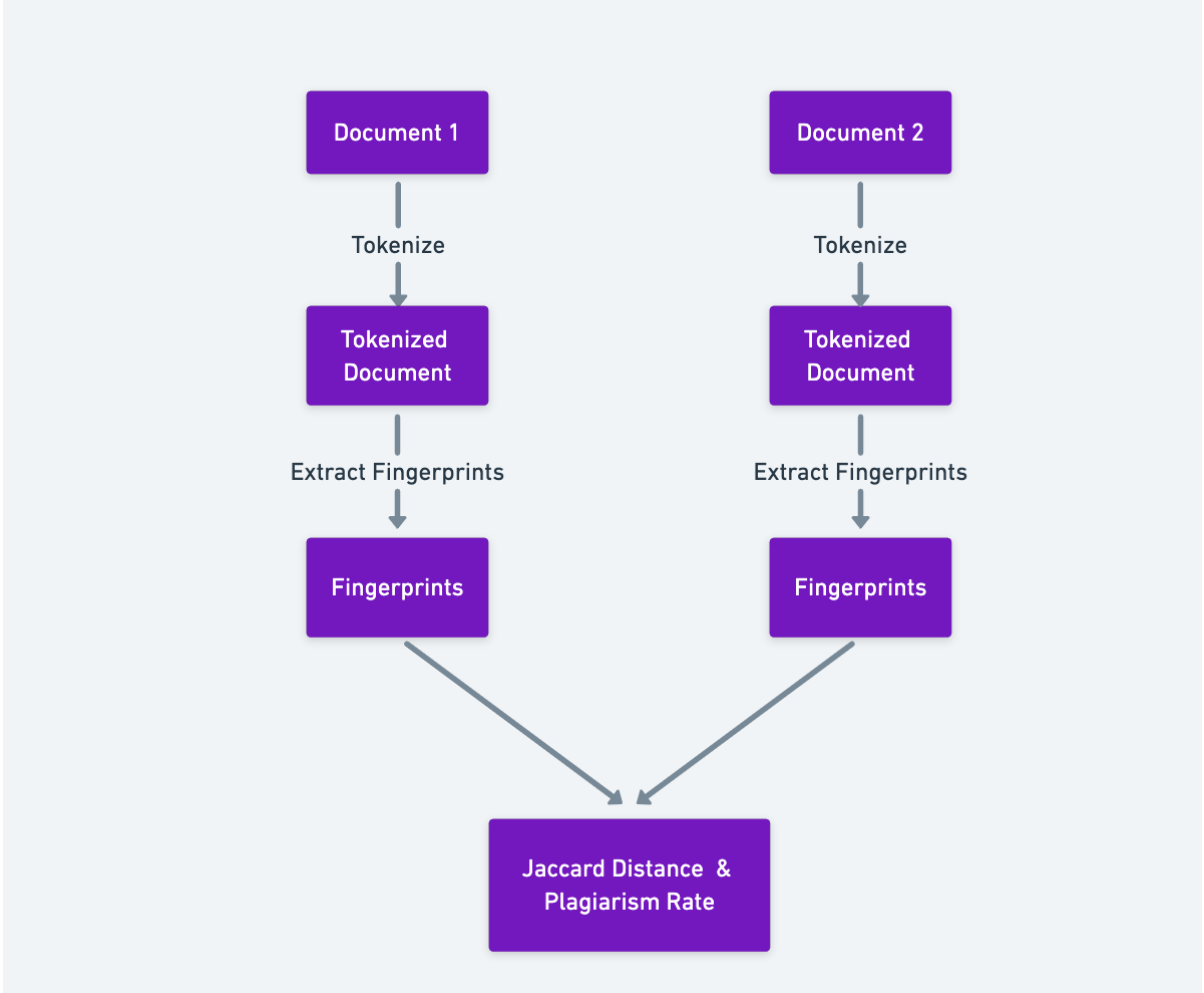
1. Tokenize their code according to the language.
2. Extract fingerprints from tokenized code.
3. Calculate the Plagiarized content from fingerprints.

There are some problems with simple winnowing algorithms . Simple winnowing method doesn't give results for each and every kind of file .

2.3.2 Extended Winnowing

Extended Winnowing is an extended version of simple winnowing . in this algorithm these steps are used :

1. Tokenize their code according to the language.
2. Extracting fingerprints with location information form tokenized code.
3. Calculate Text Similarity with Jaccard Distance and Plagiarism Rate.



1. Jaccard distance

1. We have two sets of fingerprints X , Y.
2. Jaccard Distance for two sets X and Y is the quotient of intersection elements number and union elements number.

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$

2. Plagiarism rate

1. As each document is represented by a discrete hash sequence, formula shows the calculation method, F(A) represents all fingerprints of a document, F'(A) represents the fingerprints also appear in other documents (suspected plagiarism fingerprints):

$$P(A) = \frac{|F'(A)|}{|F(A)|}$$

2.4 Implementation

We made a class called Winnowing , in which users can pass the path of two documents or directly the content of documents .

```
class Winnowing():
    def __init__(self, file1=None, file2=None,code1=None,code2=None,lang=None):
        if file1 and file2:
            if not lang:
                raise Exception("Language required")
            self.text1 , self.text2  = self.checkPoint(file1,file2)
            lexer =pygments.lexers.get_lexer_by_name(lang)
            self.token1 = tokenize(text= self.text1,lexer=lexer)
            self.str1 = toText(self.token1)
            self.token2 = tokenize(text= self.text2,lexer=lexer)
            self.str2 = toText(self.token2)

        elif code1 and code2:
            if not lang:
                raise Exception("Language required")

            if len(code1.strip())<=0 or len(code2.strip())<=0:
                raise Exception("Code can not be empty!")

            self.text1=code1
            self.text2=code2
            lexer =pygments.lexers.get_lexer_by_name(lang)
            self.token1 = tokenize(text= self.text1,lexer=lexer)
            self.str1 = toText(self.token1)
            self.token2 = tokenize(text= self.text2,lexer=lexer)
            self.str2 = toText(self.token2)

        else:
            raise Exception("Invalid Arguments")

        self.kGrams1 = self.kgrams(self.str1)  #stores k-grams, their hash values and positions in cleaned up text
        self.kGrams2 = self.kgrams(self.str2)
        self.HL1 = self.hashList(self.kGrams1)  #hash list derived from k-grams list
        self.HL2 = self.hashList(self.kGrams2)
        self.fpList1 = self.fingerprints(self.HL1)
        self.fpList2 = self.fingerprints(self.HL2)
```

Then we clean the code and tokenize it for further process .

```
def tokenize(text=None,lexer=None):
    if not text:
        raise Exception("Text can not be None.")

    tokens = lexer.get_tokens(text)
    tokens = list(tokens)
    result = []
    lenT = len(tokens)
    count1 = 0    #tag to store corresponding position of each element in original code file
    count2 = 0    #tag to store position of each element in cleaned up code text
    # these tags are used to mark the plagiarized content in the original code files.
    for i in range(lenT):
        if tokens[i][0] == pygments.token.Name and not i == lenT - 1 and not tokens[i + 1][1] == '(':
            result.append(('N', count1, count2))  #all variable names as 'N'
            count2 += 1
        elif tokens[i][0] in pygments.token.Literal.String:
            result.append(('S', count1, count2))  #all strings as 'S'
            count2 += 1
        elif tokens[i][0] in pygments.token.Name.Function:
            result.append(('F', count1, count2))   #user defined function names as 'F'
            count2 += 1
        elif tokens[i][0] == pygments.token.Text or tokens[i][0] in pygments.token.Comment:
            pass   #whitespaces and comments ignored
        else:
            result.append((tokens[i][1], count1, count2))
            #tuples in result-(each element e.g 'def', its position in original code file, position in cleaned up
            code/text)
            count2 += len(tokens[i][1])
        count1 += len(tokens[i][1])

    return result

def toText(arr):
    if not (len(arr)):
        raise Exception("Code is Invalid")
    return ''.join(list(zip(*arr))[0])
```

Then we calculate KGrams and hash them to a numerical form and then we make a hash list.

```
    def hashList(self,arr):
        if not (len(arr)):
            raise Exception("Code is Invalid")
        return list(zip(*arr))[1]

    #function to form k-grams out of the cleaned up text
    def kgrams(self,text, k = 25):
        tokenList = list(text)
        n = len(tokenList)
        kgrams = []
        for i in range(n - k + 1):
            kgram = ''.join(tokenList[i : i + k])
            hv = self.hash(kgram)
            kgrams.append((kgram, hv, i, i + k))  #k-gram, its hash value, starting and ending positions are stored
            #these help in marking the plagiarized content in the original code.
        return kgrams

    #function that returns the index at which minimum value of a given list (window) is located
    def minIndex(self,arr):
        return arr.index(min(arr))

    #sha-1 encoding is used to generate hash values
    def hash(self,text):
        #this function generates hash values
        return int((hashlib.sha1(text.encode('utf-8'))).hexdigest()[-4:],16)#using last 16 bits of sha-1 digest
```

Then we calculate fingerprints from the hash list .

```
61      def fingerprints(self,arr, winSize = 4):
62          arrLen = len(arr)
63          prevMin = 0
64          currMin = 0
65          fingerprintList = []
66          for i in range(arrLen - winSize):
67              win = arr[i: i + winSize]  #forming windows
68              currMin = i + self.minIndex(win)
69              if not currMin == prevMin:  #min value of window is stored only if it is not the same as min value of
                prev window
70                  fingerprintList.append(arr[currMin])  #reduces the number of fingerprints while maintaining
                    guarantee
71                  prevMin = currMin  #refer to density of winnowing and guarantee threshold (Stanford paper)
72
73          return fingerprintList
74
```

Now we calculate Jaccard Distance And Plagiarism rate .

```
    def jaccardCheck(self):
        fp1 = set(self.fpList1)
        fp2 = set(self.fpList2)
        comman = fp1.intersection(fp2)
        total = len(fp1)+len(fp2)
        plagcount = len(comman)
        return (plagcount/(total-plagcount))


    def plagiarismRate(self):
        fp1 = set(self.fpList1)
        fp2 = set(self.fpList2)
        total = len(fp1)
        plagcount = len(fp1.intersection(fp2))
        return (plagcount/total)
```

Jaccard Distance And Plagiarism rate are numerical values which tells about the Plagiarized Content .

Resources

1. *Simple Winnowing*
2. *A Plagiarism Detection Algorithm based on Extended Winnowing*