Plagiarism Detection Using Levenshtein Distance With Dynamic Programming

Gardahadi- 13517144
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Jl. Ganesha 10 Bandung 40132, Indonesia
13517144@std.stei.itb.ac.id

***Abstract*—Information Technology is advancing at a very rapid pace. It has come to a point where we are now able to access data and share it with people all over the world through networks such as the world wide web. That includes documents, personal files, artwork and many other forms of intellectual property. With that said, the widespread use of information retrieval technologies such as search engines has raised several problems regarding the security of these intellectual property. It is very easy for people to plagiarize published work on the internet without repercussions. In this paper one method of detecting plagiarism will be discussed which is by using the concept of the levenshtein distance implemented using dynamic programming.**

Keywords—levenshtein distance; plagiarism; dynamic programming;

I. Introduction

In this day and age, information is an abundant resource. It is widely accessible in the palm of our hands through our interconnected devices. The most commonly used media for information exchange would be the internet. Everyday, millions of users upload documents and work online both for public and private uses. The rise of search engine such as Google and Bing have made it very easy for us to search and extract these information all within a very short amount of time so long as they are not protected by strict security protocols.

This ease of access towards other people's intellectual property has led to frequent misuse in the form of plagiarism. Plagiarism is the act of copying someone else's work or idea without the consent or under a situation where it is not justified to do so. This act can be done for any form of data online but is mostly done with text based information such as documents, essays, and papers. It is commonly seen in the field of academia, this is probably due to the similarities of the work students are assigned and the work already published online

With that said, there is an urgency for an effective method to detect similarities between two documents. In this paper, we will be providing one such method by calculating Levenshtein distances or minimum edit distance between two documents. We will be implementing it using the concept of dynamic programming

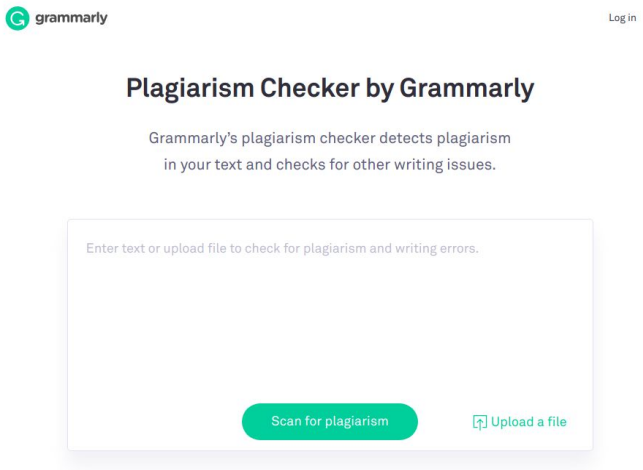

Image 1 Example of plagiarism checker (Source : grammarly.com)

II. Theoretical Framework

A. Levenshtein Distances

According to reference [2], in mathematical terms, the Levenshtein distance between two strings a and b (of length |a| and |b|) is given by the following formula

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Image 2 Levenshtein Distance Function (Source : https://en.wikipedia.org/wiki/Levenshtein_distance)

where 1(aj =/= bj) is the indicator function equal to 0 when aj = bj. The levenshtein distance is also known as the minimum edit distance. The word edit refers to a deletion, insertion or substitution of a character within that particular. In short, it is a number that represents the most

minimum amount of times an edit needs to be made to a certain string a in order to make it the same as b. Below is an example of a levenshtein distance between “lemons” and “melons”

1. lemons → memons (substitution of "l" for "m")
2. memons → melons (substitution of "m" for "l")

Because it takes at the very least 2 edits to make both strings equal, we say that lemons and melons have a levenshtein distance of 2

Another example is the Levenshtein distance between "kitten" and "sitting" which is illustrated below

1. kitten → sitten (substitution of "s" for "k")
2. sitten → sittin (substitution of "i" for "e")
3. sittin → sitting (insertion of "g" at the end).

B. *Dynamic Programming*

Dynamic programming is an algorithmic concept founded by mathematician Richard Bellman in 1950. The basic principle of dynamic programming is to decompose a problem into stages and find optimal solutions for each stage and save each solution. At a glance the definition is similar to that of the greedy method but the key difference between dynamic programming and greedy is that in dynamic programming, more than one sequence of decisions is saved and taken into consideration whereas the greedy based approach will always only generate one sequence of decisions. The principle of Dynamic programming is as follows

> If a solution is total optimum, than each k-th step towrds that solution is also optimum

The characteristics of Dynamic Programming is as follows

1. A problem can de defined as a set of stages. Where in every stage a decision must be made
2. Each stage consist of a set of states that is connected. A state is a possible decision that can be made during a certain step and can be modelled as a graph
3. Result of the decision taken within each step is used to transform the state to be used for the next stage
4. The cost in each step increases steadily and depends on the cost of previous steps

C. *Solving Levenshtein Distances using Dynamic Programming*

According to [3]. The main idea of solving Levenshtein distances using dynamic programming is to process all characters one by one starting from either from left or right sides of both strings.

Let us traverse from right corner two strings named str1 and str2, there are two important variables to take into consideration :

1. m: Length of str1 (first string)
2. n: Length of str2 (second string)

If last characters of two strings are same, we will not do anything. Ignore last characters and get count for remaining strings. So we recur for lengths m-1 and n-1. Else (If last characters are not same), we consider all operations on ‘str1’, consider all three operations on last character of first string, recursively compute minimum cost for all three operations and take minimum of three values.

1. Insert: Recur for m and n-1
2. Remove: Recur for m-1 and n
3. Replace: Recur for m-1 and n-1

A simple recursive implementation of the levenshtein distance calculator using dynamic programming is seen in the following pseudo-code :

```
function LevenshteinDistance(char s[1..m], char t[1..n]):
  // for all i and j, d[i,j] will hold the Levenshtein distance
between
  // the first i characters of s and the first j characters of t
  // note that d has (m+1)*(n+1) values
  declare int d[0..m, 0..n]
  set each element in d to zero

  // source prefixes can be transformed into empty string by
  // dropping all characters
  for i from 1 to m:
     d[i, 0] := i
  // target prefixes can be reached from empty source prefix
  // by inserting every character
  for j from 1 to n:
     d[0, j] := j
  for j from 1 to n:
     for i from 1 to m:
        if s[i-1] = t[j-1]:
          substitutionCost := 0
        else:
          substitutionCost := 1
        d[i, j] := minimum(d[i-1, j] + 1,                  // deletion
                  d[i, j-1] + 1,                  // insertion
                  d[i-1, j-1] + substitutionCost)  //substitution
```

III. Solution Breakdown

A. Overview

In order to determine whether or not two documents are considered the same we shall first split our documents into sentences. Then, we shall compare both documents sentence per sentence using a modified levenshtein distance solver implemented with dynamic programming.

The result of the distance solver will be passed on to another function which will determine whether or not a potential plagiarism is detected in a certain sentence by taking into factor the distance of sentence and the number of words.

B. Implementation Plan

We shall create two main function, one function to calculate the minimum edit distance and one function to tokenize paragraphs into sentences for pre processing to be passed on into the minimum edit distance function. The implementation shall be done using the Python programming language and the NLTK library

To determine whether or not there is an indication of plagiarism we will use the following threshold for simplicity

> If the Levenshtein distance between two sentences is **Less than** the number of characters in that sentence/2 than there is an **indication of plagiarism**

Note that this threshold is customizable and still needs to be tested in further research

IV. Implementation

A. Document Pre-Processor

First, we will create a function to split paragraph into sentences using the python NLTK library function sent_tokenizer(). This is used to split a paragraph into an array of sentences which will then be passed to the distance calculation function

```
import nltk

def paragraph_tokenizer(paragraph) :
    par_data = paragraph
    nltk_tokens = nltk.sent_tokenize(par_data)
    return nltk_tokens

def sentence_tokenizer(sentence) :
    sent_data = sentence
    nltk_tokens = nltk.word_tokenize(sent_data)
    return nltk_tokens
```

Tes result using the input paragraph “The First sentence is about Python. The Second: about Django. You can learn Python,Django and Data Analysis here. " This algorithm will separate the paragraph into 3 sentences.

```
gardahadi@gardahadi-Lenovo-V310:~/Devspace/Semester_4/S
The First sentence is about Python.
The Second: about Django.
You can learn Python,Django and Data Ananlysis here.
```

Image 3 : Result of Tokenization (Source : Personal Collection)

B. Levenshtein Distance Calculator

We Shall first implement the distance calculator recursively using python

```
// Fungsi Menghitung Levenshtein Distance
Menggunakan Dynamic Programming

def edit_distance(s, t):

    //basis
    if s == "":
        return len(t)
    if t == "":
        return len(s)
    if s[-1] == t[-1]:
        cost = 0
    else:
        cost = 1

    res = min([edit_distance(s[:-1], t)+1,
               edit_distance(s, t[:-1])+1,
               edit_distance(s[:-1], t[:-1]) + cost])
    return res
```

We shall do some modification to optimize the program by eliminating the recursive functions. This is done to lessen the memory burden done by recursively creating new stack frames in each iteration. Below is the implementation using a matrix to substitute recursion

```
def edit_distance(s1, s2):
    m=len(s1)+1
    n=len(s2)+1

    tbl = {}
    for i in range(m): tbl[i,0]=i
```

```
    for j in range(n): tbl[0,j]=j
    for i in range(1, m):
        for j in range(1, n):
            cost = 0 if s1[i-1] == s2[j-1] else 1
            tbl[i,j] = min(tbl[i, j-1]+1, tbl[i-1, j]+1, tbl[i-1,
j-1]+cost)

    return tbl[i,j]
```

We shall create another program called test.py and include this edit_distance() function using the example above of kitten and sitting and see what the result is.

test.py Function :

```
from leven import edit_distance

def main() :
    X = input("String 1 :" )
    Y = input ("String 2 : ")

    print(edit_distance(X,Y))


if __name__ == "__main__":
    main()
```

Result of function :

```
String 1 :kitten
String 2 : sitting
3
```

C. *Main Program*

Our main program will use the edit_distance() function mentioned above as well as the paragraph_tokenizer() and will use the threshold defined in the previous section.

```
from mytokenizer import paragraph_tokenizer
from mytokenizer import sentence_tokenizer
import nltk

from leven import edit_distance

def main() :

    #Hardcoded data
    Test = "The First sentence is about Python. The
Second: about Django. You can learn Python,Django
and Data Ananlysis here. "
    Source = "The First sentence is about Python. The
Second: about Django. You can learn Python,Django
and Data Ananlysis here."
    count = 1

    TestArr = []
    SourceArr = []
    for s in paragraph_tokenizer(Test) :
        TestArr.append(s)
    for s in paragraph_tokenizer(Source) :
        SourceArr.append(s)

    #Constraint : Harus sama isi element arraynya
    for s1,s2 in zip(TestArr,SourceArr) :
        if edit_distance(s1,s2) < len(s1)/2 :
            print("Terdapat indikasi plagiarisme di kalimat
%s" % (count))
        count = count+1


if __name__ == "__main__":
    main()
```

V. Testing And Analysis

A. *Testing*

Test Case 1 (Expected outcome : Detected)

Text 1 :	Text 2
"The First sentence is about Python. The Second: about Django. You can learn Python,Django and Data Ananlysis here."	"The First sentence is about Python. The Second: about Django. You can learn Python,Django and Data Ananlysis here."

Result :

```
gardahadi@gardahadi-Lenovo-V310:~/Devspace/Seme
Terdapat indikasi plagiarisme di kalimat 1
Terdapat indikasi plagiarisme di kalimat 2
Terdapat indikasi plagiarisme di kalimat 3
```

Test Case 2 (Expected Outcome : No indication in sentence number 2)

Text 1 :	Text 2 :
"The First sentence is	"The Initial sentence is

about Python. The Second: about Django. You can learn Python,Django and Data Ananlysis here."	not about Java. This second sentence is different. You can learn everything about anything here."

```
gardahadi@gardahadi-Lenovo-V310:~/Devspace/Semester
Terdapat indikasi plagiarisme di kalimat 1
Terdapat indikasi plagiarisme di kalimat 3
gardahadi@gardahadi-Lenovo-V310:~/Devspace/Semester
```

Test Case 3 (Expected Outcome : Detected)

Text 1 : "The First sentence is about Python. The Second: about Django. You can learn Python,Django and Data Ananlysis here."	Text 2 : "The Initial sentence is about Python. The Second: about Flask. You can learn Python,Django and Data manipulation here."

Result :

```
gardahadi@gardahadi-Lenovo-V310:~/Devspace/Semester_4/Stima
Terdapat indikasi plagiarisme di kalimat 1
Terdapat indikasi plagiarisme di kalimat 2
Terdapat indikasi plagiarisme di kalimat 3
gardahadi@gardahadi-Lenovo-V310:~/Devspace/Semester_4/Stima
```

Test case 4 (Expected Outcome : No Detection)

Text 1 : "The First sentence is about Python. The Second: about Django. You can learn Python,Django and Data Ananlysis here. "	Text 2 : "The first is differenta. This second sentence is different. The third sentence is also very different."

Result :

```
gardahadi@gardahadi-Lenovo-V310:~/Devspace/Seme
Tidak ada indikasi plagiarisme
gardahadi@gardahadi-Lenovo-V310:~/Devspace/Seme
```

B. *Analysis*

Based on the three test cases above, we can see that the algorithm can detect several cases of plagiarism even when the test cases contain edited sentences. However the current implementation still contains many weaknesses as follows :

1. Will not work optimally if the number of sentences is different from both cases thus we shall need to do more preprocessing to make the paragraph equal in length
2. The threshold is only based on heuristic analysis and is open to many possible optimization
3. For cases where the text is completely different then there might still be a chance of a false positive case as the algorithm might consider the amount of distance to be within the range

VI. Conclusion

The levenshtein algorithm is an effective tool to detect similarities between two documents because it is able to represent the amount of edit one sentence needs to make in order to transform into the other. It can be used as an added alternative to existing plagiarism detection programs that mainly use string pattern matching algorithms. Although with that said, there needs to be more research on the most effective threshold to be put to determine whether or not two documents are the same or not

Acknowledgment

The author of this document would like to thank Mr. Rinaldi Munir for his guidance and patience in teaching IF2211 algorithm strategies subject for class K-3 this year. It has been an honor to learn from him and gain useful insights as well as inspiration to continue studying.

References

[1] Anany Levitin, Introduction to The Design and Analysis of Algorithm, 3rd ed., London: Pearson, 2011, pp.283-297.

[2] Stanford CS124 Lecturer, “Minimum Edit Distance”, accessed from https://www.stanford.edu/ on 26 April 2019.

[3] GeeksforGeeks, “Dynamic Programming | Set 5 (Edit Distance)”, accessed from https://www.geeksforgeeks.org/ on 13 Mei 2018.

[4] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2019

Gardahadi / 13517144