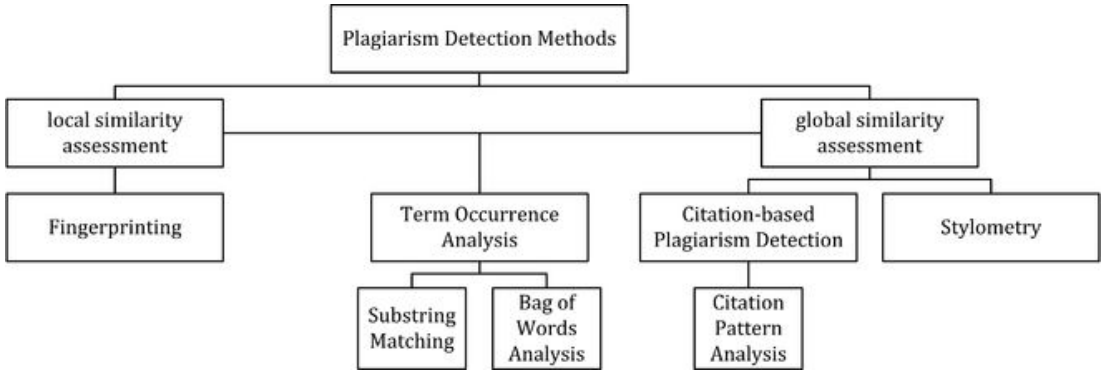
Plagiarism Detector

Wikipedia Says:

Plagiarism detection is the process of locating instances of plagiarism within a work or document. The widespread use of computers and the advent of the Internet has made it is easier to plagiarize the work of others. Most cases of plagiarism are found in academia, where documents are typically essays or reports. However, plagiarism can be found in virtually any field, including novels, scientific papers, art designs, and source code.Detection of plagiarism can be either manual or software-assisted. Manual detection requires substantial effort and excellent memory, and is impractical in cases where too many documents must be compared, or original documents are not available for comparison. Software-assisted detection allows vast collections of documents to be compared to each other, making successful detection much more likely.

1.**Plagiarism**: According to Wikipedia, plagiarism is the "wrongful appropriation" and "stealing and publication" of another author's "language, thoughts, ideas, or expressions" and the representation of them as one's own original work.

2.The approaches are characterized by the type of similarity assessment they undertake: global or local. Global similarity assessment approaches use the characteristics taken from larger parts of the text or the document as a whole to compute similarity, while local methods only examine pre-selected text segments as input.



Implementation

In this project input is a list (or collection) of text documents that are stored in a directory and implemented in following aspects:

1. Given a document, finding out if there are any similarities with other documents in the directory.

2. List out all the documents with similarities more than *x* percentage. Here *x* is the threshold decided by the user representing % of match among available text documents. This project is subdivided into following methodologies:

This project is subdivided into following methodologies:

Bag of Words:

In this method you have to compute a distance (an angle) between two given documents or between two strings using the cosine similarity metric. You start with reading in the text and counting frequency of each word. The word frequency distribution for the text D is Java's Map from words to their frequency counts, which we'll denote as freq(D) . We view freq(D) as a vector of nonnegative integers in N-dimensional space.

For example, reading the string "To be or not to be" results in the following map {be=2, not=1, or=1, to=2}.These 4 distinct words make a document representation as a 4-dimensional vector {2, 1, 1, 2} in term space.A word is a sequence of letters [a..zA..Z] that might include digits [0..9] and the underscore character. All delimiters are thrown away and not kept as part of the word. Here are examples of words:

abcd

abcd12

abc_

a12cd

15111

We'll treat all upper-case letters as if they are lower-case, so that "CMU" and "cmu" are the same word. The Euclidean norm of the frequency vector is defined by

||x|| = sqrt(x1^2+x2^2+....+xn^2)

where xk denotes frequencies for each word in the text. For the above example, the norm is

||x|| = sqrt(2^2+1^2+1^2+2^2) = 3.16228

The Dot product (or the inner product) of two frequency vectors X and Y is defined by

X = {x1,x2,x3,...,xn}; Y = {y1,y2,y3,...,yn}

X . Y = x1.y1+x2.y2+...+xn.yn

Here we multiply frequencies xk and yk of the same word in both text documents.

Finally, we define a distance between two documents D1 and D2 by using cosine similarity

measurement:

Observe, the distance between two identical documents is 0, and the distance is Pi/2 = 1.57... if two documents have no common words.

dist(D1,D2) = arccos(freq(D1) . freq(D2) / ||freq(D1)|| * ||freq(D2)||)

Expected output: If there is plagiarism, most words that are used in text will match. There will be significant similarities with frequencies as well.

Example:

File - 1 Content: To be or not to be

File - 2 Content: Doubt truth to be a liar

The words from both the files are as follows.

Note: Freq(X) is : Frequency of words from file x

Words Freq(file 1) Freq(file 2) Dot product of (Freq(File1) and Freq(File 2))

The Euclidean norm of the frequency vector is as follows:

|| File 1 || is: sqrt(10)

|| File 2 || is:sqrt(6)

So, the similarity between the documents are

Similarity Function = cos θ = Dot Product(Frequency of File 1, Frequency of File 2) / ||File 1|| *

|| File 2||

Obtained output:

```
Test1
             File1.txt       File2.txt       File3.txt       File4.txt       File5.txt
File1.txt    100             68              69              72              73

File2.txt    68              100             76              73              76

File3.txt    69              76              100             73              79

File4.txt    72              73              73              100             73

File5.txt    73              76              79              73              100
```

String matching:

In this method every text document is checked for common substrings with the verifying document. Although this method is computationally expensive finding **longest common substring** will tell us what % or how many sentences are copied from the verifying document.

This method relies on patterns, and text overlaps.

Example:

File - 1 Content: what is your name

File - 2 Content: my name is xyz

In Computer Science, brute-force search or exhaustive search, also known as generate and test, is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

Step 1: **w**hat is your name **m**y name is xyzw and m does not match so we shift the string to the next position(word), until we find a match.

Step2: what **is** your name

my name **is** xyz

Now, we can see it and it matches. So, LCS = 2 and we continue the iteration.

StepN: what is your **name**

my **name** is xyz

Now, LCS changes to 4

Length of LCS is 4

Total length of file1 and file2 is 31

%match = ((4 * 2) / 31) * 100 = 25%

Expected output: If there is plagiarism, longest common substring will have large portions of verifying documents. It is possible that we may find other sentences as well that were copied.

Obtained output:

```
H:\Projects\Plagarisam Dector\Stringmatching\Stringmatching>java Solution
Test1
               file1.txt       file2.txt       file3.txt       file4.txt       file5.txt
file1.txt       100.0 %         34.1 %          38.2 %          40.8 %          0.0 %

file2.txt       34.1 %          100.0 %         10.1 %          66.5 %          0.0 %

file3.txt       38.2 %          10.1 %          100.0 %         15.4 %          0.0 %

file4.txt       40.8 %          66.5 %          15.4 %          100.0 %         0.0 %

file5.txt       0.0 %           0.0 %           0.0 %           0.0 %           100.0 %

Maximum similarity is between file2.txt and file4.txt
```

Future Work:

- It can be used to check a couple of documents at a time.To improve the amount of input we can use various data structures and algorithms to improve its performance.
- We can also make use of an end to end application of plagiarism detector by establishing database connection and store it's results.

References:

- https://en.wikipedia.org/wiki/Content_similarity_detection
- http://en.writecheck.com/ways-to-avoid-plagiarism