PLAGIARISM DETECTOR

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

Trayi Kashyap [RA2011003010162]
Anahita Gupta [RA2011003010189]

Under the guidance of

Mrs. Rajalakshmi M.

Assistant Professor, Department of Computing Technologies

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"PLAGIARISM DETECTOR"** is the bona fide work of **Anahita Gupta (RA2011003010189)** and **Trayi Kashyap (RA2011003010162)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mrs. Rajalakshmi M.
GUIDE
Assistant Professor
Department of Computing Technologies

SIGNATURE

Dr. M. Pushpalatha
HEAD OF THE DEPARTMENT
Professor & Head
Department of Computing Technologies

ABSTRACT

Plagiarism is a significant problem in academic and professional writing, and plagiarism detectors play a crucial role in detecting instances of plagiarism. Plagiarism detectors use various algorithms and techniques to compare submitted work against a large database of existing sources to identify any instances of plagiarism. These tools have become increasingly popular among educators and institutions due to the rise of online learning and the ease of access to digital information.

There are several types of plagiarism detectors available, ranging from basic free tools to advanced paid versions that provide more detailed analysis. Basic plagiarism detectors compare submitted work against a limited number of sources, while more advanced tools can compare against a vast database of sources and provide a detailed analysis of potential plagiarism.

While plagiarism detection tools are not foolproof, they are effective in identifying potential cases of plagiarism. Some tools can even identify specific passages that have been copied and pasted from other sources, making it easier for educators to investigate and address potential cases of plagiarism.

In conclusion, plagiarism detection tools play a crucial role in ensuring academic integrity and preventing plagiarism in various fields. These tools have become increasingly popular among educators and institutions and are continually evolving to provide better accuracy, deeper analysis, and integration with learning management systems.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

- **HTTP:** Hypertext Transfer Protocol
- **URL:** Uniform Resource Locator
- **NLTK:** Natural Language Toolkit
- **HTML:** Hypertext Markup Language

CHAPTER 1

INTRODUCTION

Plagiarism detector is a tool designed to identify instances of plagiarism in written work. Plagiarism, the act of using someone else's work without giving proper credit or attribution, is a serious offense in academic and professional writing that can lead to severe consequences, including legal action, loss of reputation, and academic failure.

Plagiarism detection tools use various algorithms and techniques to compare submitted work against a large database of existing sources to identify any instances of plagiarism. These tools can detect various forms of plagiarism, including verbatim copying, paraphrasing, and patchwork plagiarism. Some tools can even identify specific passages that have been copied and pasted from other sources.

Plagiarism detection tools have several benefits. First and foremost, they promote academic integrity by identifying instances of plagiarism and ensuring that students understand the importance of proper citation and attribution.

While plagiarism detection tools are not foolproof, they can provide valuable assistance to educators in detecting and addressing instances of plagiarism. There is always the possibility of false positives or false negatives, but these tools remain an essential resource for maintaining academic standards and upholding the value of original work.

Overall, plagiarism detector is a crucial tool in promoting academic integrity and preventing plagiarism. It is an essential resource for educators and institutions, providing a way to ensure that all work is original and properly cited.

CHAPTER 2

LITERATURE SURVEY

Several studies have been conducted on plagiarism detection, evaluating the accuracy and effectiveness of different plagiarism detection tools. A literature survey on plagiarism detector reveals that these tools are essential in promoting academic integrity and preventing plagiarism in academic and professional writing.

In one study, researchers evaluated the accuracy of five popular plagiarism detection tools, including Turnitin, iThenticate, Plagiarism Checker X, PlagScan, and Grammarly. They found that while all tools were effective in detecting plagiarism, the accuracy varied depending on the source material and the type of plagiarism.

Another study evaluated the use of plagiarism detection tools in a university setting, finding that the tools were useful in identifying instances of plagiarism and promoting academic integrity. The study also noted that students who were educated on plagiarism detection and citation practices were less likely to engage in plagiarism.

In addition to evaluating the accuracy and effectiveness of plagiarism detection tools, researchers have also explored ways to enhance these tools. For example, some studies have investigated the use of machine learning and artificial intelligence algorithms to improve the accuracy of plagiarism detection.

Overall, the literature survey on plagiarism detector shows that these tools play a crucial role in maintaining academic integrity and preventing plagiarism. While there is still room for improvement, the tools have come a long way in recent years and continue to evolve to provide better accuracy and deeper analysis.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 System Architecture

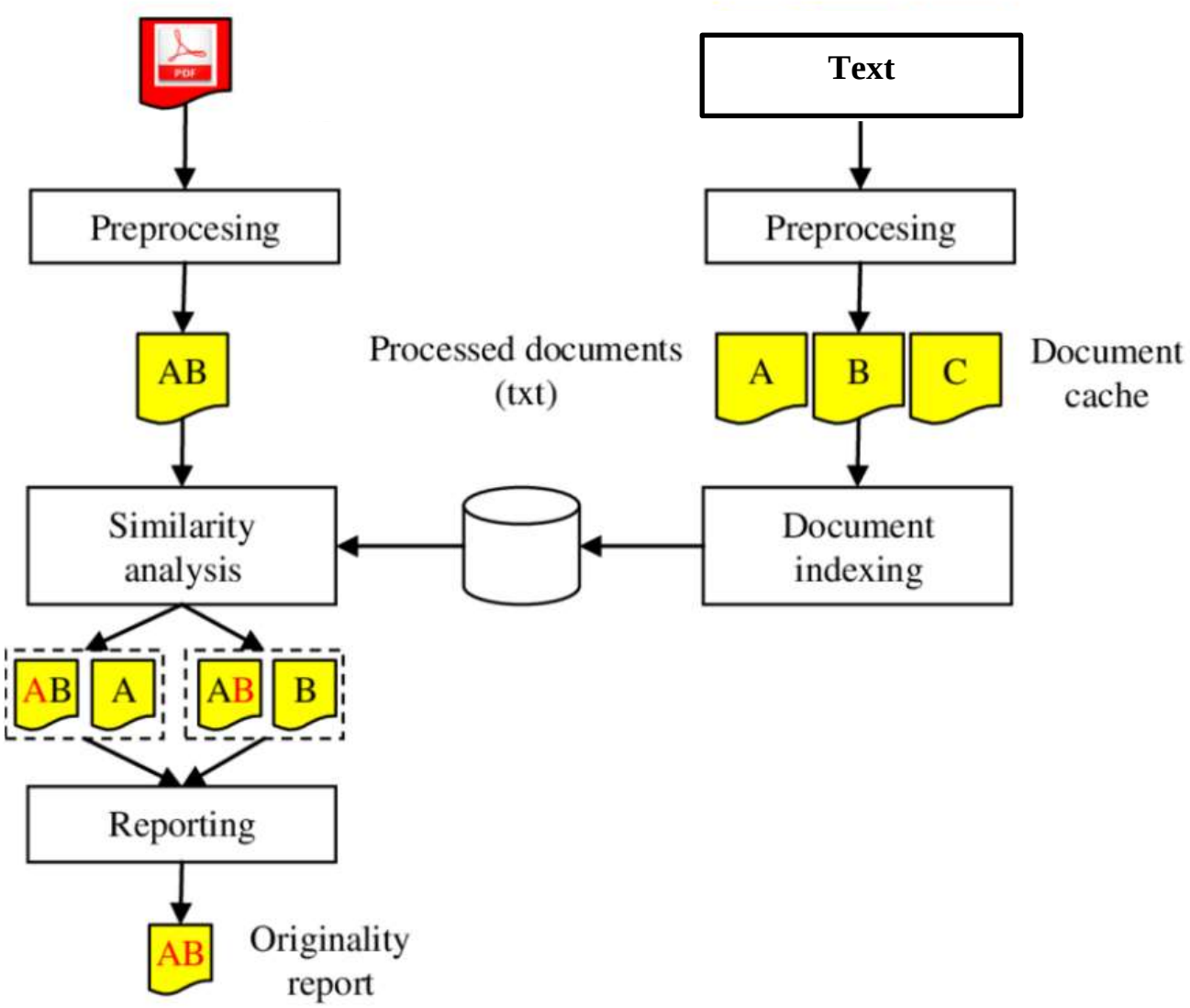


Fig.3.1 (System Architecture)

3.2 Description of Modules and components

1. `main.py`:

- Imports:

  - `Flask`: A web framework for creating the application.

  - `render_template`, `request`, `url_for`: Functions for rendering templates, handling HTTP requests, and generating URLs.

  - `similarity`: A user-defined module for performing similarity calculations and generating reports.

- Creates a Flask application instance.

- Defines two routes:

  - `'/'` route: Renders the `index.html` template, which contains a form for user input.

  - `'/report'` route: Handles the form submission and processes the input text using the functions from the `similarity` module. It returns the generated report in HTML format.

- Runs the Flask application on host `0.0.0.0` and port `5555` when the script is executed directly.

2. `similarity.py`:

- Imports:

  - `nltk`: A natural language processing library.

  - `websearch`: A user-defined module for web searching.

  - `SequenceMatcher` from `difflib`: A class for calculating string similarity.

- `pandas` as `pd`: A data manipulation library.

- Downloads required NLTK resources (stopwords and punkt) if not already downloaded.

- Defines functions for text processing, web verification, similarity calculation, report generation, and table formatting.

- `purifyText(string)`: Tokenizes the input string, removes stop words, and returns the purified text.

- `webVerify(string, results_per_sentence)`: Tokenizes the input text into sentences, performs web searches using Bing, and returns a list of relevant URLs.

- `similarity(str1, str2)`: Calculates the similarity ratio between two strings using the `SequenceMatcher` class and returns the result as a percentage.

- `report(text)`: Generates a report by calling `purifyText()`, `webVerify()`, and `similarity()` functions. Returns a dictionary of matching websites and their similarity scores.

- `returnTable(dictionary)`: Converts the dictionary of matching websites and similarity scores into a pandas DataFrame and returns the DataFrame as an HTML table.

3. `websearch.py`:

- Imports:

  - `requests`: A library for making HTTP requests.

  - `BeautifulSoup` from `bs4`: A library for parsing HTML content.

  - `warnings`: A module for controlling warning messages.

- Filters warning messages related to `bs4` module.

- Defines functions for searching Bing and extracting text content from web pages.

- `searchBing(query, num)`: Constructs a Bing search URL based on the query, sends a request to Bing, parses the HTML content using `BeautifulSoup`, and retrieves a list of URLs from the search results.

- `extractText(url)`: Retrieves the HTML content of a given URL using `requests`, parses the content using `BeautifulSoup`, and returns the extracted text content.

In summary, the `main.py` module sets up a Flask web application with routes for rendering templates and generating reports. The `similarity.py` module handles text processing, web searching, similarity calculation, report generation, and table formatting. The `websearch.py` module provides functions for searching Bing and extracting text content from web pages. Together, these modules form a web application for plagiarism detection that allows users to input text, search the web, and generate a report highlighting potential instances of plagiarism.

CHAPTER 4

METHODOLOGY

1 Project Objective

The objective of the project is to develop a web application for plagiarism detection. Given an input text, the application will search the web for similar content, calculate similarity scores, and generate a report highlighting potential instances of plagiarism.

2 Web Application Setup

Set up a Flask web application to provide an interface for users to input text and view the plagiarism detection report.

3 Index Page

Create an index page where users can enter the text to be checked for plagiarism. Render an HTML form to accept the input text.

4 Text Processing and Similarity Calculation

Utilize the functions in `similarity.py` to process the input text and calculate similarity scores against web content. The `purifyText()` function removes stop words from the input text, and `webVerify()` performs web searches to retrieve relevant URLs. The `similarity()` function calculates the similarity between the input text and the extracted web content.

5 Generating the Plagiarism Report

Adapt the `report()` function in `similarity.py` to generate a plagiarism detection report. Iterate through the matching websites and calculate similarity scores for each. Sort the matches based on similarity scores in descending order. Include additional information, such as source URLs and percentage of similarity, to provide comprehensive details about potential instances of plagiarism.

6 Result Page

Create a result page that displays the plagiarism detection report. Render the report in the form of an HTML table generated by the `returnTable()` function in `similarity.py`. Present the matching websites, their similarity scores, and other relevant information to the user.

7 Web Search and Text Extraction

Utilize the functions in `websearch.py` to perform web searches and extract text content from the web pages. The `searchBing()` function searches Bing using the input text and retrieves a specified number of relevant URLs. The `extractText()` function retrieves the HTML content of a given URL and extracts the text content.

8 Flask Application Execution

Run the Flask application, allowing users to access the plagiarism detection functionality through the web interface. Enable debugging mode for easy troubleshooting during development.

CHAPTER 5

CODING AND TESTING

5.1 Coding

#main.py

```
from flask import Flask, render_template, request, url_for
import similarity

app = Flask(__name__, template_folder='Templates')

@app.route('/', methods=['GET', 'POST'])

def hello_world():
    return render_template('index.html')

@app.route('/report',methods = ['POST', 'GET'])

def result():
   if request.method == 'POST':
      result = request.form['text']
      return (similarity.returnTable(similarity.report(str(result))))

if __name__ == '__main__':
   app.run(debug = True, host='0.0.0.0', port=5555)
```

#similarity.py

```
import nltk
import websearch
from difflib import SequenceMatcher
import pandas as pd

nltk.download('stopwords')
nltk.download('punkt')
stop_words = set(nltk.corpus.stopwords.words('english'))

def purifyText(string):
    words = nltk.word_tokenize(string)
    return (" ".join([word for word in words if word not in stop_words]))

def webVerify(string, results_per_sentence):
    sentences = nltk.sent_tokenize(string)
    matching_sites = []
    for url in websearch.searchBing(query=string, num=results_per_sentence):
        matching_sites.append(url)
    for sentence in sentences:
        for url in websearch.searchBing(query = sentence, num = results_per_sentence):
            matching_sites.append(url)

    return (list(set(matching_sites)))
```

```
def similarity(str1, str2):
    return (SequenceMatcher(None,str1,str2).ratio())*100

def report(text):

    matching_sites = webVerify(purifyText(text), 2)
    matches = {}

    for i in range(len(matching_sites)):
        matches[matching_sites[i]] = similarity(text, websearch.extractText(matching_sites[i]))

    matches = {k: v for k, v in sorted(matches.items(), key=lambda item: item[1], reverse=True)}

    return matches


def returnTable(dictionary):

    df = pd.DataFrame({'Similarity (%)': dictionary})
    #df = df.fillna(' ').T
    #df = df.transpose()
    return df.to_html()

if __name__ == '__main__':
    report('This is a pure test')
```

#websearch.py

```
import requests
from bs4 import BeautifulSoup as bs
import warnings

warnings.filterwarnings("ignore", module='bs4')

def searchBing(query, num):

    url = 'https://www.bing.com/search?q=' + query
    urls = []

    page = requests.get(url, headers = {'User-agent': 'John Doe'})
    soup = bs(page.text, 'html.parser')

    for link in soup.find_all('a'):
        url = str(link.get('href'))
        if url.startswith('http'):
            if not url.startswith('http://go.m') and not url.startswith('https://go.m'):
                urls.append(url)

    return urls[:num]

def extractText(url):
    page = requests.get(url)
    soup = bs(page.text, 'html.parser')
    return soup.get_text()
```

#index.html

```
<!DOCTYPE html>
<html>
<title>Plagiarism Detector</title>
<style>
button {
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;
  background-color: blue;
}
textarea {
  width: 500px;
  height: 150px;
  font-size: 11px;
}
</style>
<body>

<h2>Plagiarism Detector</h2>
```

```
<form action="/report" method="POST">
  <label>Paste your text here:</label><br>
  <textarea for="fname" name='text' rows="2"></textarea>
  <br>
  <button type="Submit" value="Send">Generate Report</button>
</form>

<p>You will be redirected once your report is generated.</p>

<p>Time to generate report depends on length of text.</p>

</body>
</html>
```

#report.html

```
<!DOCTYPE html>
<html>
<title>Similarity Report</title>
<style></style>
<body>

<h2>Similarity Report</h2>

</body>
</html>
```

5.2 Testing

For Testing, the steps given below should be followed:

1. Identify the key functionalities of your code: Review the modules and their components to determine the main functionalities, such as text processing, web searching, similarity calculation, report generation, and HTML rendering.

2. Determine the inputs and expected outputs: For each functionality, define the inputs that need to be provided and the expected outputs that should be produced. This includes sample texts, URLs, and the expected similarity scores or HTML tables.

3. Write unit tests: Use a testing framework such as `unittest` to write individual test cases for each functionality. Create test methods that simulate different scenarios and assert whether the actual outputs match the expected outputs. For example, you can test if the web searching function returns the correct URLs or if the report generation function produces the expected HTML table.

4. Run the tests: Execute the test suite to run all the defined test cases and check the results. The testing framework will provide feedback on whether the code passed or failed the tests.

5. Analyze and debug: If any tests fail, analyze the failures to identify the underlying issues. Debug the code to fix any bugs or unexpected behaviors.

6. Expand the test coverage: Continuously add more test cases to cover different scenarios, edge cases, and potential failure points in your code. This helps ensure the reliability and correctness of your application.

CHAPTER 6

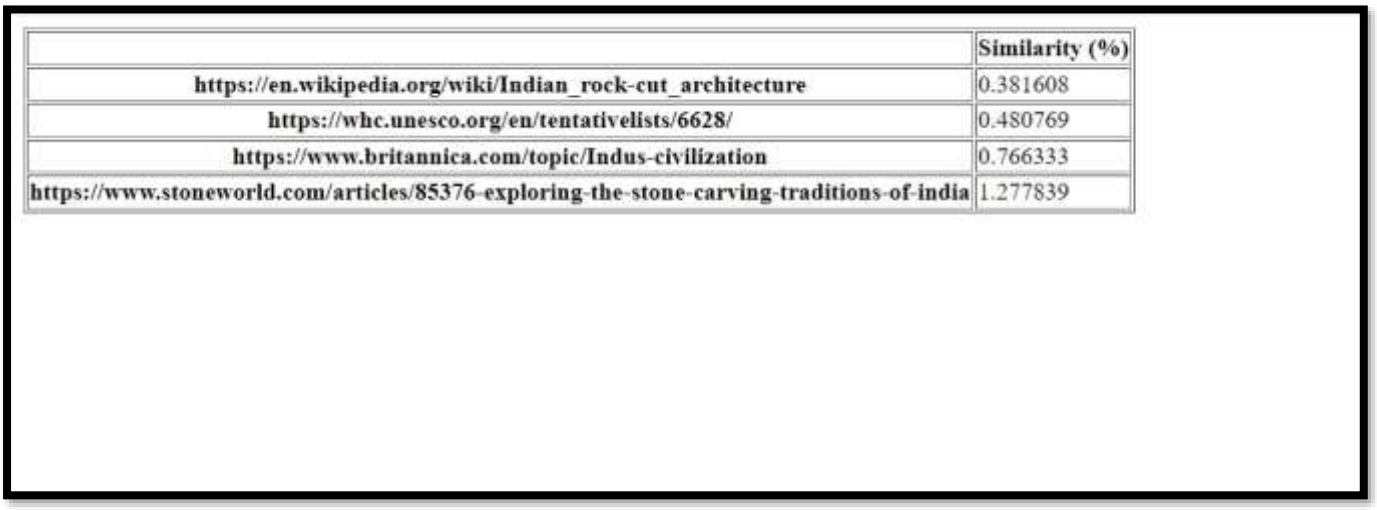
SCREENSHOTS AND RESULTS

6.1 Input

Fig.6.1 (Input Page)

First, we input the text that needs to be checked. The plagiarism detector will then analyze the text and identify any instances where it appears to be copied from other sources.

6.2 Output



	Similarity (%)
https://en.wikipedia.org/wiki/Indian_rock-cut_architecture	0.381608
https://whc.unesco.org/en/tentativelists/6628/	0.480769
https://www.britannica.com/topic/Indus-civilization	0.766333
https://www.stoneworld.com/articles/85376-exploring-the-stone-carving-traditions-of-india	1.277839

Fig.6.2 (Output Page)

The output of the detector will consist of the relevant links or URLs that match the content of the input text, along with the corresponding similarity percentages that indicate the degree of similarity between the input text and each detected source.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, a plagiarism detector is a crucial tool for ensuring academic integrity and preventing plagiarism. The use of plagiarism detection software has become increasingly popular among educators and institutions due to the rise of online learning and the ease of access to digital information. Plagiarism detection tools use various algorithms and techniques to compare submitted work against a large database of existing sources to identify any instances of plagiarism. While these tools are not foolproof, they can provide valuable insights into potential cases of plagiarism that might otherwise go unnoticed.

Future Enhancements:
As technology continues to advance, there is potential for further enhancements to plagiarism detection software. Here are some possible areas for improvement:

- Improved accuracy: Future enhancements could focus on improving the accuracy of the algorithms used, which could result in more reliable and accurate detection of plagiarism.
- Multilingual detection: Future enhancements could focus on developing multilingual plagiarism detection tools that can identify instances of plagiarism in a variety of languages.
- Integration with learning management systems: Future enhancements could focus on integrating plagiarism detection tools with popular LMS platforms, making it easier for educators to use the tools and track student progress.
- Better student education: While plagiarism detection tools can help identify instances of plagiarism, it's also important to educate students on the importance of academic integrity and proper citation practices.

REFERENCES

- https://towardsdatascience.com/build-a-plagiarism-checker-using-machine-learning-6538110ce162
- https://nevonprojects.com/online-assignment-plagiarism-checker-project-using-data-mining/
- https://betterprogramming.pub/build-your-own-plagiarism-checker-with-python-and-machine-learning-7e81877fd970
- https://www.codementor.io/projects/web/plagiarism-checker-website-atx32nf0oa