



## SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

B. TECH CSE-AI

**Subject: Advance Machine Learning**

**Subject Code: 23CSE514**

### **MINI PROJECT**

### **Plagiarism Detection system for academic content**

Submitted to,

Prof:- Dr. Guruvammal S

Assistant Professor,

Department of Computer Science & Engineering (AI),  
Faculty of Engineering & Technology, Jain (Deemed-  
To-Be) University.

Submitted by,

Name : GUNJI CHANIKYA

USN : 23BTRCA055

Branch & Section : CSE- AI

Date of Submission : 24<sup>TH</sup> November 2025

# TABLE OF CONTENTS

SI.NO	CONTENT	PAGE NUMBER
1	Introduction	2
2	Literature Review (Prior - Studies)	3-5
3	<b>Project Details</b> 3.1 Problem Statement 3.2 Dataset Description 3.3 Methodology 3.4 Implementation Code 3.5 Results and Analysis	5-16
4	References	17

## **1. INTRODUCTION:**

**1.1 Background:** Plagiarism has become a major concern in schools, colleges, universities, and research organizations. With the rapid growth of digital content, students and researchers now have easy access to information through websites, online journals, PDFs, and AI-generated content. This has increased the chances of copying, rephrasing, or reusing text without giving proper credit. Traditional manual checking methods are time-consuming, inconsistent, and ineffective when dealing with large volumes of academic submissions. Therefore, automatic plagiarism detection using Natural Language Processing (NLP) has become essential to maintain originality, protect academic integrity, and ensure fair evaluation.

**1.2 Motivation:** The growing ease of access to online resources and availability of sophisticated AI writing tools has made plagiarism more common and harder to detect. Teachers often receive dozens of assignments, reports, and project files, making manual verification extremely difficult. Even small modifications—like changing a few words or reordering sentences—can deceive basic comparison methods. This challenge creates a strong need for an intelligent, automated plagiarism detection system that can compare texts accurately and highlight similarities at both document and sentence levels. NLP-based methods such as TF-IDF and cosine similarity provide a reliable, fast, and scalable approach to detect similarities in different file formats.

**1.3 Objectives:** This project aims to:

- Develop an NLP-based plagiarism detection system using TF-IDF and cosine similarity
  - Support multiple document formats including PDF, DOCX, and TXT
  - Preprocess text effectively for accurate comparison
  - Detect and display similarity at both document and sentence levels
  - Provide a clear similarity report with percentages
  - Highlight matching text to help teachers identify copied content
  - Build a simple, user-friendly, and reproducible solution using Python and Streamlit
-

## **2. LITERATURE REVIEW (PRIOR STUDIES):**

### **2.1 Traditional Approaches**

Early plagiarism detection methods relied primarily on rule-based or lexical matching systems. These systems compared documents using simple text patterns such as identical words, matching phrases, or fixed substring search.

String matching algorithms, such as the Karp–Rabin algorithm and Longest Common Subsequence (LCS), were popular for identifying exact text matches.

---

### **2.2 Modern NLP and Machine Learning Approaches**

#### **Supervised Learning Approaches:**

Supervised machine learning models have been explored for plagiarism detection, although they require labeled datasets of “plagiarized” and “non-plagiarized” text.

Examples include:-

- **Support Vector Machines (SVM):** Effective for high-dimensional text features but computationally expensive for large corpora.
- **Random Forests:** Able to rank feature importance and detect structural patterns in text documents.
- **Neural Networks:** Useful for capturing complex text relationships but require large labeled datasets, which are rare in plagiarism research.

#### **Unsupervised & NLP-Based Approaches:**

Most plagiarism detection systems rely on unsupervised similarity techniques, since labeled plagiarism datasets are limited.

### **TF-IDF + Cosine Similarity:**

This is one of the most widely used techniques in academic settings. TF-IDF vectorizes text based on word importance, and cosine similarity measures how close two document vectors are. This method is fast, scalable, and works well for assignments, reports, and research papers.

### **Semantic Embedding Models (Word2Vec, GloVe):**

These models capture meaning-level similarity between words, allowing detection of paraphrased content that lexical methods miss.

### **Transformer-Based Models (BERT, SBERT):**

Deep contextual models like BERT significantly improve plagiarism detection by understanding semantic relationships between sentences. They outperform TF-IDF for paraphrasing but require large computational resources.

---

## **Key Studies**

- Alzahrani et al. (2012)  
Demonstrated that semantic-based methods outperform simple lexical matching for plagiarism detection.
  - Potthast et al. (2014) – PAN Plagiarism Detection Initiative  
Introduced standardized plagiarism detection datasets and benchmarks widely used in academic research.
  - Barrón-Cedeño (2010)  
Showed that combining lexical and semantic features improves detection accuracy for paraphrased plagiarism.
  - Ferrero et al. (2020)  
Highlighted the effectiveness of deep learning (BERT/SBERT) for semantic plagiarism detection.
-

## 2.3 Methods for Dealing With Paraphrasing and Content Variations

Unlike fraud detection, plagiarism detection deals primarily with paraphrasing, synonym replacement, reordering of sentences, and meaning-level similarity. Researchers have proposed techniques to address these issues:

- Character-level n-gram analysis to catch small edits in words
- Word-level n-gram analysis to detect copied phrases
- Stopword removal and lemmatization to reduce noise
- Sentence similarity algorithms (SequenceMatcher, semantic embeddings)
- Hybrid models combining lexical and semantic similarity

These methods help capture both exact matches and paraphrased content, making them suitable for academic plagiarism detection.

## 3. PROJECT DETAILS

### 3.1 Problem Statement

#### Objective:

Build an NLP-based plagiarism detection system capable of identifying similarity between academic documents and highlighting possible copied or paraphrased content using TF-IDF, cosine similarity, and sentence-level matching.

#### Challenges:

**Different File Formats:** Academic content may come from PDF, DOCX, or TXT files, requiring robust text extraction methods.

**Paraphrasing:** Students often change words or rearrange sentences, making exact matching techniques ineffective.

**Document Size Variation:** Reports and assignments may vary from a few lines to dozens of pages.

**Semantic Understanding:** Detecting meaning-level similarity is harder than direct copy-paste detection.

**Performance:** System must compute similarity quickly for multiple files.

**Noise in Text:** Issues such as formatting, line breaks, and special symbols must be handled during preprocessing.

## **Success Criteria:**

- Accurately compute similarity percentage between documents
- Extract text correctly from all supported file types (PDF/DOCX/TXT)
- Highlight top matching/similar sentences
- Generate a clear, easy-to-read plagiarism report
- Provide real-time results through a user-friendly Streamlit interface

## **3.2 Dataset Description :-**

Source: You may use any academic dataset, including sample assignments, research abstracts, or publicly available academic text datasets.

**For demonstration purposes, you used a small sample dataset stored inside the project folder.**

## **Dataset Characteristics:**

- Multiple text files (.txt)
- Academic writing (assignments, paragraphs, reports)
- Some files contain intentional plagiarism for testing
- Includes direct copy-paste and paraphrased content

## **File Types Supported:**

TXT – Raw text

DOCX – Microsoft Word documents

PDF – Scanned/typed academic PDFs

## **Important Notes:**

- PDFs may contain layout noise; extraction is handled using PyPDF2
- DOCX files are parsed using python-docx
- TXT files are read directly

## **Data Properties:**

- Text length: varies from 50–2000 words
- Domain: academic narratives, explanations, summaries ● No labels required (unsupervised similarity detection)
- Used purely for text comparison, not classification

### 3.3 Methodology

#### 3.3.1 Workflow Pipeline:

Text Extraction → Preprocessing → TF-IDF Vectorization → Cosine Similarity Calculation → Sentence Matching → Similarity Report Generation → Streamlit User Interface.

#### 3.3.2 Text Extraction

Steps performed:

1. Loaded uploaded files inside Streamlit
2. Detected file format (TXT/DOCX/PDF)
3. Extracted raw text using appropriate libraries
4. Handled encoding errors (Unicode, Latin-1, UTF-8)
5. Combined text paragraphs into a clean string

Key Findings:

- TXT extraction is the cleanest
- DOCX extraction preserves paragraph structure
- PDF extraction quality depends on document formatting

#### 3.3.3 Text Preprocessing Operations Performed:

1. Converted text to lowercase
2. Removed unnecessary whitespace
3. Removed line breaks and special characters
4. Tokenized sentences for matching
5. Prepared clean input for TF-IDF

Purpose:

To standardize text so that similarity calculations are accurate and meaningful.

#### 3.3.4 Feature Representation – TF-IDF

The system uses a combined TF-IDF vectorizer:

**Word-level TF-IDF:** Captures important keywords

**Character-level TF-IDF (char n-grams):**

Captures small edits, reordering, and paraphrasing

Feature vector = [word\_ngrams + char\_ngrams]

### 3.3.5 Similarity Calculation – Cosine Similarity

Cosine similarity measures the angle between two vectors. Output ranges from 0% (no match) to 100% (exact match).

Approach provides:

- Document-to-document similarity
- Ranking of most similar document pairs
- Threshold-based plagiarism flagging

### 3.3.6 User Interface (Streamlit)

Streamlit provides:

- File upload interface
- Real-time similarity table
- Sentence-level evidence display
- Downloadable CSV report
- Interactive threshold adjustment

## 3.4 Implementation CODE:

```
# app.py (UI/UX improved) - Chanikya Plagiarism Checker

import streamlit as st
import os
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from difflib import SequenceMatcher
import PyPDF2
import pandas as pd
import numpy as np
from io import BytesIO

# ----- Page config & custom CSS -----

st.set_page_config(page_title="Chanikya Plagiarism Checker", layout="wide",
initial_sidebar_state="expanded")
```

```

# ----- Header -----

col1, col2 = st.columns([3,1])

with col1:

    st.image("logo.png", width=199)

    st.title("Chanikya Plagiarism Checker")

    st.markdown("<div class='small'>Simple, fast plagiarism checking for academic content — upload  
TXT, DOCX, PDF</div>", unsafe_allow_html=True)

with col2:

    st.markdown("<div style='text-align:right'><span class='badge blue'>Demo</span><br><span  
class='small'>v1.0</span></div>", unsafe_allow_html=True)

# ----- Sidebar (controls, help, dataset) -----

with st.sidebar:

    st.header("Controls")

    uploaded = st.file_uploader("Upload files (txt / docx / pdf)", accept_multiple_files=True,  
type=["txt", "docx", "pdf"])

    threshold = st.slider("Flag threshold (%)", 10, 100, 70, 1)

    show_sentences = st.checkbox("Show similar sentence pairs", value=True)

    st.markdown("---")

    st.header("Quick actions")

    if st.button("Load sample dataset (kaggle_dataset):"):

        st.session_state.load_samples = True

    st.markdown("**Tips:**\n- Use 3–8 files for fast demo\n- Short paragraphs work best for demo")

    st.markdown("---")

    st.header("About")

    st.markdown("**Chanikya Plagiarism Checker**\n\nBuilt with Streamlit + scikit-learn.\nSimple UI  
for college demos.")

    st.markdown("")

# ----- Optional sample screenshots -----

# local screenshot paths

s1 = "/mnt/data/9aad0a13-7caf-4644-924c-f80bf8c1766f.png"
s2 = "/mnt/data/3b33e639-6988-4350-9dcb-f2b59d3f29d7.png"
s3 = "/mnt/data/a753fbdd-4302-4fe3-a21f-a8e0e6097f38.png"

img = [p for p in (s1, s2, s3) if os.path.exists(p)]

```

```

if imgs:

    st.markdown("### Screenshots / Demo Preview")

    cols = st.columns(len(imgs))

    for c, img in zip(cols, imgs):

        c.image(img, caption=os.path.basename(img), use_column_width=True)

st.markdown("---")

# ----- Helper functions -----

def read_txt_stream(f):

    raw = f.read()

    if isinstance(raw, bytes):

        try:

            return raw.decode("utf-8")

        except:

            try:

                return raw.decode("latin-1")

            except:

                return raw.decode(errors='ignore')

    return str(raw)

def read_docx_stream(f):

    if Document is None:

        st.error("python-docx not installed. Install with: pip install python-docx")

        return ""

    tmp = "temp_docx.docx"

    with open(tmp, "wb") as out:

        out.write(f.read())

    doc = Document(tmp)

    full = [p.text for p in doc.paragraphs]

    try:

        os.remove(tmp)

    except:

        pass

```

```

return "\n".join(full)

def read_pdf_stream(f):
    if PyPDF2 is None:
        st.error("PyPDF2 not installed. Install with: pip install PyPDF2")
        return ""
    reader = PyPDF2.PdfReader(f)
    pages = []
    for p in reader.pages:
        try:
            pages.append(p.extract_text() or "")
        except:
            pages.append("")
    return "\n".join(pages)

def preprocess(text):
    if text is None:
        return ""
    text = text.lower().replace("\n", " ").strip()
    return text

def compute_combined_tfidf(docs):
    vec_word = TfidfVectorizer(stop_words="english", ngram_range=(1,2))
    vec_char = TfidfVectorizer(analyzer='char_wb', ngram_range=(3,5))
    tf_word = vec_word.fit_transform(docs)
    tf_char = vec_char.fit_transform(docs)
    from scipy.sparse import hstack
    combined = hstack([tf_word, tf_char])
    sim = cosine_similarity(combined)
    return sim

def top_similar_sentences(a_text, b_text, topn=5):
    a_sents = [s.strip() for s in a_text.split('.') if s.strip()]

```

```

b_sents = [s.strip() for s in b_text.split('.') if s.strip()]

pairs = []

for sa in a_sents:
    for sb in b_sents:
        ratio = SequenceMatcher(None, sa, sb).ratio()
        pairs.append((ratio, sa, sb))

pairs.sort(reverse=True, key=lambda x: x[0])

return pairs[:topn]

# ----- Load sample dataset (optional) -----
if st.session_state.get("load_samples", False):
    # try to load files from kaggle_dataset folder if exists
    sample_dir = "kaggle_dataset"
    if os.path.exists(sample_dir):
        st.success("Loaded files from kaggle_dataset/")
    # mimic upload by creating 'uploaded' list from file paths
    uploaded = []
    for fname in sorted(os.listdir(sample_dir))[:8]:
        path = os.path.join(sample_dir, fname)
        uploaded.append(open(path, "rb"))

else:
    st.warning("kaggle_dataset folder not found in project directory.")

# ----- Main processing & UI -----
if uploaded and len(uploaded) >= 2:
    # read files
    names = []
    texts = []
    progress = st.progress(0)
    total = len(uploaded)
    for idx, f in enumerate(uploaded, start=1):
        name = getattr(f, "name", f.name if hasattr(f, "name") else f"file_{idx}")
        names.append(name)

```

```

if name.lower().endswith(".txt"):

    txt = read_txt_stream(f)

elif name.lower().endswith(".docx"):

    txt = read_docx_stream(f)

elif name.lower().endswith(".pdf"):

    txt = read_pdf_stream(f)

else:

    txt = ""

texts.append(preprocess(txt))

progress.progress(int(idx/total * 100))

progress.empty()

# compute similarity

with st.spinner("Computing combined TF-IDF & similarity..."):

    sim = compute_combined_tfidf(texts)

# build table

rows = []

n = len(names)

for i in range(n):

    for j in range(i+1, n):

        score = float(sim[i,j]) * 100.0

        rows.append({"Doc A": names[i], "Doc B": names[j], "Similarity (%)": round(score,2)})

df = pd.DataFrame(rows).sort_values("Similarity (%)", ascending=False).reset_index(drop=True)

# dashboard summary cards

c1, c2, c3 = st.columns(3)

c1.markdown("<div class='card'><b>Total files</b><div class='small'>%d files uploaded</div></div>" % n, unsafe_allow_html=True)

top_sim = df["Similarity (%)].max() if not df.empty else 0

c2.markdown("<div class='card'><b>Top similarity</b><div class='small'>%s%%</div></div>" % round(top_sim,2), unsafe_allow_html=True)

flagged_count = len(df[df["Similarity (%)] >= threshold])

```

```

c3.markdown("<div class='card'><b>Flagged pairs</b><div class='small'>%d pairs
(>%d%%)</div></div>" % (flagged_count, threshold), unsafe_allow_html=True)

st.subheader("Pairwise Similarity")
st.dataframe(df, use_container_width=True)

st.markdown("### Flagged Pairs (evidence)")

flagged = df[df["Similarity (%)"] >= threshold]

if flagged.empty:
    st.info("No pairs exceeded the threshold.")

else:
    for idx, row in flagged.iterrows():

        a = row["Doc A"]; b = row["Doc B"]; score = row["Similarity (%)"]

        st.markdown(f"<div class='card'><b>{a}</b> — <b>{b}</b> <span style='float:right'
class='badge red'>{score}%</span></div>", unsafe_allow_html=True)

        if show_sentences:

            i = names.index(a); j = names.index(b)

            top = top_similar_sentences(texts[i], texts[j], topn=5)

            for r, sa, sb in top:
                st.markdown(f"> **Ratio:** {r:.2f} \n> A: {sa} \n> B: {sb}")

            st.markdown("---")

    csv = df.to_csv(index=False).encode('utf-8')

    st.download_button("Download similarity report (CSV)", csv, file_name="similarity_report.csv",
    mime="text/csv")

else:
    st.info("Upload at least 2 files (txt / docx / pdf) to compare. Use the left sidebar to upload files or
load sample dataset.")

    st.markdown(""""

    **Demo instructions (simple):**

    1. Click 'Browse files' and select multiple files (txt/docx/pdf).
    2. Set the threshold slider (default 70%).
    3. upload 'Load sample dataset' to quickly test.
    4. Wait for the app to compute and then view results table and flagged pairs.

    """")

```

## 3.5 Results and Analysis :

### 3.5.1 System Output Interpretation

The plagiarism checker generates:

The plagiarism checker generates:	
Metric	Meaning
Similarity (%)	How similar two documents are
Flagged Pairs	Files above the threshold
Sentence Matches	Strongest matching lines
Final Report	Downloadable CSV

### 3.5.2 Example Similarity Table :

Document A	Document B	Similarity
file1.txt	file2.txt	78.34%
report.docx	summary.pdf	42.10%
assignment1	assignment2	89.20%

### 3.5.3 Sentence Match Analysis:

For each flagged pair, the system displays:

- Matching sentence from Document A
- Matching sentence from Document B
- Sentence similarity ratio (0–1)

This helps visually confirm plagiarism.

### 3.5.4 System Accuracy & Performance Strengths:

- Highly accurate for direct copy–paste
- Detects paraphrasing using character-n-grams
- Works for long academic paragraphs
- Faster than deep-learning approaches

### Limitations:

- Cannot fully detect meaning-level plagiarism
- PDF extraction may skip some text
- Not trained like a classifier—pure similarity-based system

### 3.5.5 Comparison with Other Approaches:

3.5.5 Comparison with Other Approaches		
Approach	Pros	Cons
TF-IDF + Cosine	Fast, simple, accurate	Limited semantic detection
String Matching	Easy	Fails on paraphrasing
BERT Models	High accuracy	Slow, GPU required

Your chosen method offers the **best balance of accuracy + speed** for a mini project.

## **4. REFERENCES :-**

1. Alzahrani, S. M., Salim, N., & Abraham, A. (2012). "Understanding Plagiarism Linguistic Patterns, Textual Features, and Detection Methods." *IEEE Transactions on Systems, Man, and Cybernetics*.
2. Potthast, M., Hagen, M., & Stein, B. (2014). "The PAN Plagiarism Detection Shared Task." *CLEF Conference on Multilingual and Multimodal Information Access*.
3. Barrón-Cedeño, A. (2010). "Plagiarism Detection Across Distant Language Pairs." *European Chapter of the ACL*.
4. Ferrero, J., Rangel, F., & Rosso, P. (2020). "A Survey on Paraphrase and Semantic Text Similarity Detection." *Springer NLP Series*.
5. Scikit-learn Documentation. "TF-IDF Vectorizer."  
[https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.TfidfVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)
6. Scikit-learn Documentation. "Cosine Similarity."  
[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine\\_similarity.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html)
7. Python-docx Library Documentation. "Reading DOCX Files."  
<https://python-docx.readthedocs.io/en/latest/>
8. PyPDF2 Documentation. "PDF Text Extraction."  
<https://pypdf2.readthedocs.io/>
9. Difflib — Python Standard Library. "SequenceMatcher for Text Comparison."  
<https://docs.python.org/3/library/difflib.html>
10. Streamlit Official Documentation.  
<https://docs.streamlit.io/>
11. Kaggle Dataset: "Text Similarity / Academic Text Samples" (used for demonstration).  
<https://www.kaggle.com/>
12. Google Research. "Semantic Textual Similarity Benchmark (STS)."
13. Manning, C. D., Raghavan, P., & Schütze, H. (2009). *Introduction to Information Retrieval*. Cambridge University Press.