

# DETECTOR DE PLAGIO DE CÓDIGO FUENTE EN PYTHON

## TIPO 1, 2 Y 3

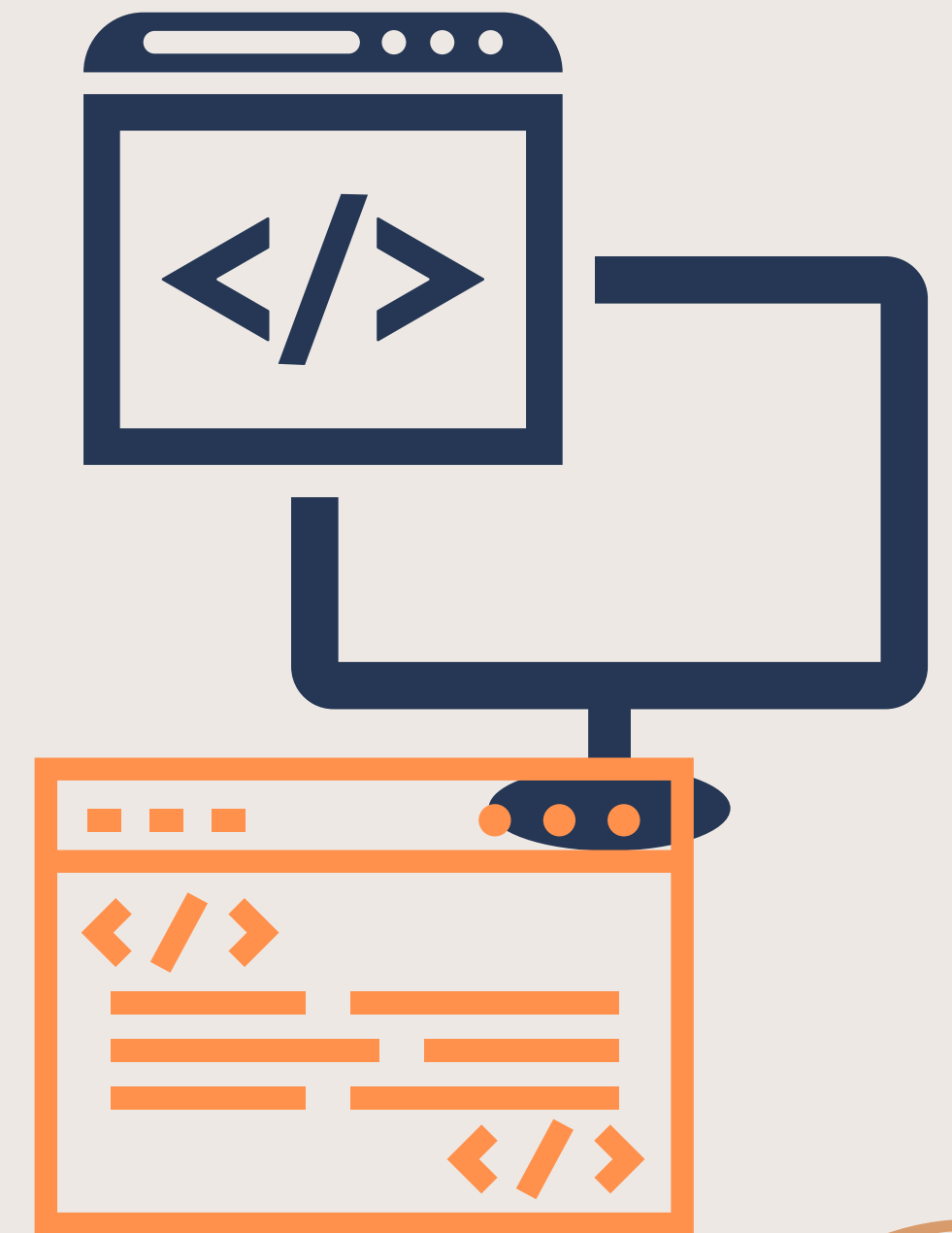
José Juan Irene Cervantes  
Alejandro Daniel Moctezuma  
Augusto Gómez Maxil  
Jacqueline Villa Asencio

# INTRODUCCIÓN AL PROBLEMA

¿COPIAR O APRENDER?

En los últimos años, el plagio en código fuente se ha vuelto un problema creciente, impulsado por el fácil acceso a plataformas como GitHub, foros de ayuda y herramientas de inteligencia artificial.

Las herramientas tradicionales como MOSS o JPlag detectan similitudes textuales, pero fallan cuando se modifica la estructura del código o se renombra contenido. Esto hace urgente el desarrollo de soluciones más precisas, adaptables y centradas en el contexto académico actual.






# ALCANCE DEL PROYECTO

Este proyecto se enfoca en detectar tres tipos de plagio frecuentes en el ámbito académico:

- **Tipo I: Copia exacta** del código sin modificaciones.
- **Tipo II: Renombramiento** de identificadores conservando la lógica.
- **Tipo III: Reestructuración** del código sin alterar funcionalidad.

El sistema está diseñado para identificar estos casos, sin depender de coincidencias literales, utilizando técnicas léxicas, estructurales y de clasificación automática.



---

# OBJETIVO DEL PROYECTO

- Desarrollar un sistema híbrido capaz de detectar automáticamente plagio en código fuente Python.
- Enfocarse en tres tipos mencionados:
  - Exacto
  - Renombramiento
  - Estructural.

Por medio de:

- **Análisis léxico**, como *Suffix Array* y *Difflib*
- **Análisis estructural**, a través del uso de *Árboles de Sintaxis Abstracta (AST)* y *distancias de edición jerárquicas*
- **Clasificación automática** mediante un modelo *Random Forest binario y multietiqueta*.



# ESTADO DEL ARTE

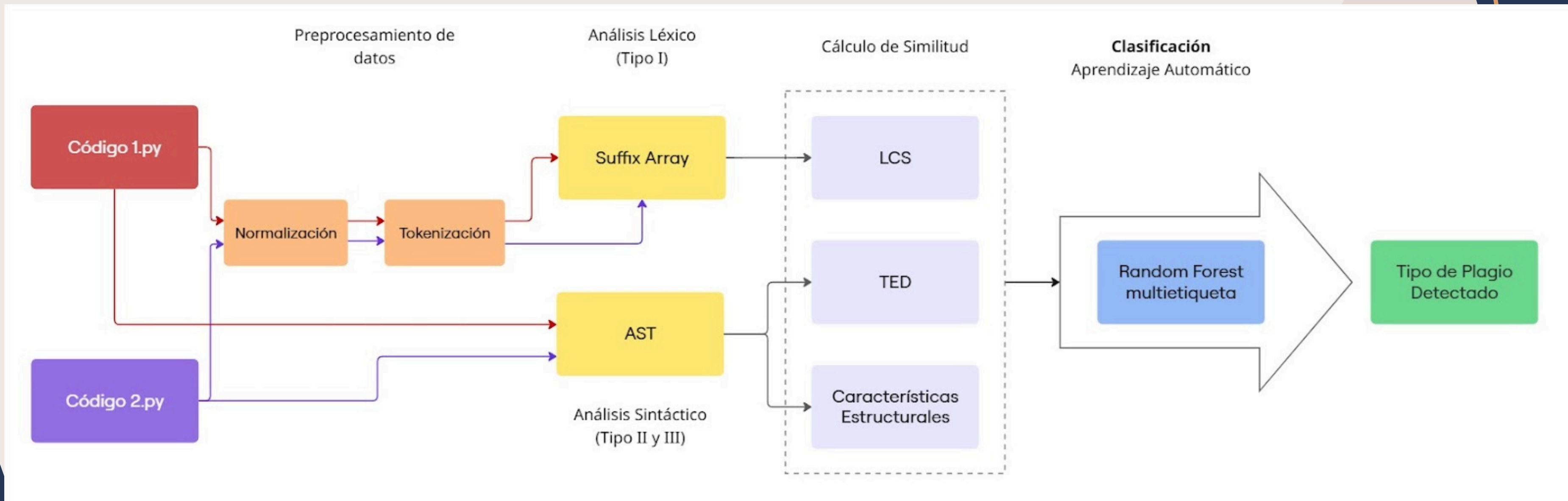
Diversos trabajos han abordado el problema del plagio en código fuente desde enfoques distintos.

- 1 Gandhi et al. emplearon SVM junto con representaciones AST para detectar estructuras similares, pero su solución no logra superar casos de renombramiento superficial.
- 2 Por su parte, Mehse et al. propusieron un modelo basado en TF-IDF y clustering que agrupa códigos por similitud estadística, aunque sin considerar la estructura lógica del programa.
- 3 Perilla y colaboradores utilizaron tokenización y SVM para identificar plagio en Java, lo que limita su aplicabilidad a otros lenguajes



# SOLUCIÓN PROPUESTA

## Arquitectura general del sistema





# BASE DE DATOS

```
giamism_detector.py  adjacency_list_tipo2.py  adjacency_list.py X  plagiarism_detector.py  adjacency_list_tipo2.py X

> algorithms > adjacency_list > adjacency_list.py > Node
class Node:
    def __init__(self, vertex):
        self.vertex = vertex
        self.next = None

def create_graph(adj_list, edges):
    no_of_nodes = len(adj_list)
    for i, neighbors in edges.items():
        last = None
        for val in neighbors:
            if val >= no_of_nodes or val < 0:
                raise ValueError(f"Invalid node value {val}. It must be between 0 and {no_of_nodes-1}")
            new_node = Node(val)
            if adj_list[i] is None:
                adj_list[i] = new_node
            else:
                last.next = new_node
            last = new_node

def display_graph(adj_list):
    result = []
    for i, node in enumerate(adj_list):
        node_list = [i]
        ptr = node
        while ptr:
            node_list.append(ptr.vertex)
            ptr = ptr.next
        result.append(node_list)
    return result

Data > algorithms > adjacency_list > adjacency_list_tipo2.py > Nodo
1 class Nodo:
2     def __init__(self, vertice):
3         self.vertice = vertice
4         self.siguiente = None
5
6 def generar_rede(lista_adj, aristas):
7     no_de_nodos = len(lista_adj)
8     for i, vecinos in aristas.items():
9         ultimo = None
10        for val in vecinos:
11            if val >= no_de_nodos or val < 0:
12                raise ValueError(f"Valor de nodo inválido {val}. Debe estar entre 0 y {no_de_nodos-1}")
13            nuevo_nodo = Nodo(val)
14            if lista_adj[i] is None:
15                lista_adj[i] = nuevo_nodo
16            else:
17                ultimo.siguiente = nuevo_nodo
18            ultimo = nuevo_nodo
19
20 def mostrar_rede(lista_adj):
21     resultado = []
22     for i, nodo in enumerate(lista_adj):
23         lista_nodos = [i]
24         ptr = nodo
25         while ptr:
26             lista_nodos.append(ptr.vertice)
27             ptr = ptr.siguiente
28     return resultado
```

# DESGLOSE

## Comparadores

1

### **Comparador léxico (Suffix Array):**

Busca fragmentos de texto similares entre dos archivos, aunque hayan cambiado el orden o partes del código. Sirve para detectar coincidencias no tan obvias.

2

### **Comparador estructural (AST + TED):**

Analiza la forma en que está construido el código por dentro, como si viera su “esqueleto”. Así puede identificar cuando dos programas son iguales aunque estén escritos diferente.

3

### **Comparador línea a línea (Difflib):**

Revisa el código como lo haría un editor, línea por línea, para encontrar textos casi idénticos. Es muy útil cuando el plagio es directo.

---



# DESGLOSE

## Código principal

### **compare\_files**

Compara dos archivos y obtiene un conjunto de características como similitud léxica, diferencias estructurales, nodos AST y distancia de edición.

### **generate\_training\_data\_from\_leaf\_dirs**

Recorre todas las carpetas de archivos .py, compara pares y genera automáticamente un dataset para entrenar el modelo.

### **predict\_plagiarism**

Usa el modelo entrenado para predecir si un par de archivos tiene plagio y en qué grado.

### **algorithm**

Entrena un modelo Random Forest o carga uno ya guardado. Evalúa su rendimiento y ejecuta una predicción entre dos archivos de prueba.

### **compare\_all\_pairs**

Compara todos los archivos dentro de una carpeta, y muestra los resultados en consola con detalles de similitud y tipo de plagio estimado.



# BASE DE DATOS

## Código original

La base de datos original fue generada después de considerar la del profesor. Al no ser suficiente, se buscó una con mayor complejidad, y se expandió.

## Variedad de tipos

La base de datos está subdividida en carpetas, siendo que en cada una de estas se encuentra la versión original y las variaciones.

## Generación de variaciones

Para esta base de datos fue necesario entrenar el modelo con distintas variaciones de plagio de los códigos originales, mismas que se generaron.

## Banderas y nombres

Se usaron banderas (como por ejemplo archivos .txt) y el nombre de los archivos para entrenar el modelo adecuadamente. Esto permite capturar un porcentaje de correctitud.

# USO DE DOS MODELOS

## BINARIO

Este modelo se encarga de responder una pregunta directa:

¿existe plagio entre los dos archivos?

Toma como entrada un vector de características derivado de métricas léxicas y estructurales, y predice un valor binario (sí o no).

Se entrena con etiquetas generales (plagio = 1, no plagio = 0), y su objetivo es ofrecer una evaluación rápida y confiable.

## MULTIETIQUETA

Este modelo va un paso más allá:  
Identifica el tipo de plagio.

Usa el mismo vector de características, pero se entrena con etiquetas múltiples para clasificar en tipo I, II o III.

Permite una salida más explicativa para docentes, ya que distingue entre copia exacta, renombramiento y reestructuración.

# ENTRENAMIENTO



## DEL MODELO

El entrenamiento se realizó por medio de distintos algoritmos, siendo que se tiene un modelo de **MultiLabelBinarizer**, comparadores mediante difflib y SA, entre otros.

A diferencia de otros modelos, el nuestro usa un **RandomForestClassifier** para poder entrenar otro modelo, el cual predice los tipos de plagio que se esperan.

Ambos modelos son guardados con el fin de no tener que reprocesar todos los archivos después de la primera ejecución.

En caso de existir, son cargados.

# RESULTADOS DEL MODELO

## Umbrales de Clasificación

Score < 0.35

Score >= 0.9

0.35 < Score < 0.9

	Precision	Recall	F1-Score	Support
No Plagio	0.69	0.77	0.73	160
Plagio	0.89	0.76	0.79	226


# RESULTADOS DEL MODELO

## Umbrales de Clasificación

Score < 0.35

Score >= 0.9

0.35 < Score < 0.9

 Distribución de clases en el dataset:


Tipo 0: 25.17% (924 muestras)


Tipo 1: 19.31% (709 muestras)

Tipo 2: 23.40% (859 muestras)

Tipo 3: 32.12% (1179 muestras)

 Entrenando modelo...

 Tiempo de entrenamiento: 4.23 segundos

 Evaluación del modelo:

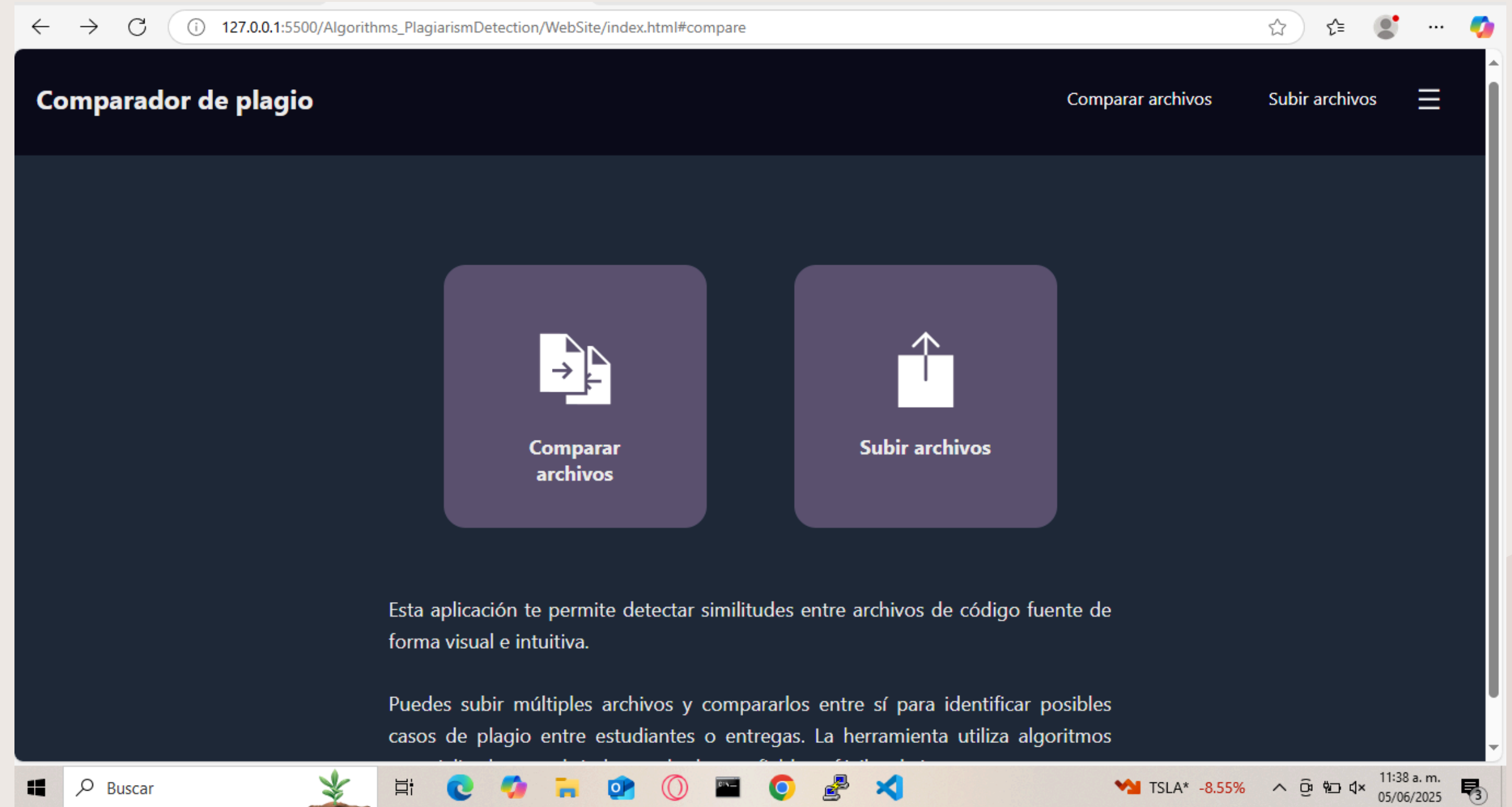
	precision	recall	f1-score	support
Tipo 0	0.45	0.35	0.39	175
Tipo 1	0.22	0.18	0.20	131
Tipo 2	0.54	0.47	0.50	172
Tipo 3	0.78	0.66	0.71	253



# IMPLEMENTACIÓN DEL PROTOTIPO

Como se puede ver en la siguiente captura, el prototipo presenta la opción de subir archivos o de comprarlos.

De igual forma se brinda una pequeña explicación sobre el sitio.



# IMPLEMENTACIÓN

Comparador de plagio

Comparar archivos Subir archivos

## Resultados de Comparación de Archivos

Archivo 1	Archivo 2	¿Plagio?	Tipos
bellman_ford.py	bellman_ford_dfs_tipo0.py	No	-
bellman_ford.py	bellman_ford_tipo2.py	Sí	Tipo 2 o 3
bellman_ford.py	bellman_ford_tipo3.py	Sí	Tipo 2 o 3
bellman_ford.py	mean_median_mode_tipo3_copiatipo1.py	No	-
bellman_ford.py	testcase1.py	No	-
bellman_ford.py	testcase1_tipo2.py	No	-
bellman_ford.py	testcase1_tipo3.py	No	-

# IMPLEMENTAR LA INFRAESTRUCTURA

Se imprimen múltiples resultados que permiten interpretar si es plagio o no: banderas de AST para definir si es plagio, los valores del modelo, la similitud SA y TED

Los resultados impresos incorporan ya la normalización y refactorización de variables y funciones en los códigos.

Se imprime el nombre de los archivos comparados, si se considera un tipo de plagio, la similitud que existe y los tipos predichos.

Se utilizan los archivos subidos a la carpeta “/uploads” para realizar las comparaciones, lo que permite que el sistema sea personalizable.

# CONCLUSIONES

El sistema demostró ser una herramienta eficaz para detectar plagio **tipo I, II y III** en código **Python**, combinando **análisis léxico, estructural y aprendizaje automático**, ideal para entornos académicos por su implementación local.

Sin embargo, aunque los resultados son prometedores, el modelo aún presenta **margen de mejora**, especialmente en la detección de plagio tipo II, donde los cambios son más sutiles y difíciles de identificar.

Mejorar la calidad y el balance del dataset, así como incorporar modelos semánticos avanzados, representa el siguiente paso para aumentar la precisión del sistema y cubrir escenarios más complejos de plagio como el tipo IV.



**MUCHAS  
GRACIAS**