Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias

Evidencia 1.
Análisis de similitud empleando Inteligencia Artificial

Desarrollo de aplicaciones avanzadas de ciencias computacionales
TC3002B.301

Presentado por:
Frida Bailleres González

Fecha de entrega:
4 jun 2025

Profesor:
Pedro Oscar Pérez Murueta

Metodología

Selección de Datos

Elegimos el dataset “Programming Contest Plagiarism in Java” porque estaba bien estructurado y era manejable en tamaño. Originalmente consistía en carpetas con pares de códigos y un archivo CSV que indicaba si eran plagio o no. Lo convertimos en un solo CSV con tres columnas: `code1`, `code2` y `label`.

	A	B	C
1	code1	code2	label
2	import java.i	import java.u	0
3	import java.u	import java.u	0
4	import java.i	import java.i	1
5	import java.u	import java.i	0
6	import java.n	import java.u	0
7	import java.u	import java.i	0
8	import java.u	import java.i	0
9	import java.i	import java.i	0
10	import java.u	import java.i	0
11	import java.i	import java.i	1

El dataset estaba desbalanceado, con muchos más ejemplos de no plagio. Por eso, generamos nuevos pares de código plagiado con ayuda de IA (ChatGPT), usando como base algunos ejemplos reales del dataset. Así logramos equilibrar el conjunto con 656 pares positivos (plagio) y 656 negativos, para un total de 1312 pares. Separamos el 20% de los datos para pruebas.

Análisis de Datos

Antes de aplicar el modelo, hicimos una limpieza de los fragmentos de código. Creamos un script en Python para eliminar todos los comentarios (tanto en línea como multilínea) y reducir los espacios vacíos. Esto ayudó a que el modelo no se confundiera con contenido irrelevante.

Luego usamos TF-IDF para vectorizar el texto de los códigos. Esta técnica nos permitió identificar tokens relevantes que se repiten en un fragmento pero no en muchos otros, lo que mejora la detección de similitudes reales.

Representamos cada par como la diferencia absoluta entre sus vectores TF-IDF (abs(code1 - code2)), lo cual permite capturar similitud entre fragmentos. Esta diferencia se usó como entrada para el modelo.

Construcción del Modelo

Usamos una regresión logística binaria como clasificador. Elegimos este modelo porque es rápido, eficiente y fácil de interpretar: cada token tiene un peso que influye en la decisión del modelo, lo que facilita entender por qué algo se clasifica como plagio.

Configuramos el modelo con `max_iter=50` (que son como las epochs) y usamos `train_test_split` con 20% para prueba y `random_state=42` para replicabilidad.

Lo mejor es que no requirió muchos ajustes: primero probamos solo TF-IDF y luego mejoró notablemente al combinarlo con regresión logística.

Resultados

Presentación de hallazgos
Nuestro modelo logró una *accuracy* del 90%, con un F1-score de 0.90 para ambas clases.

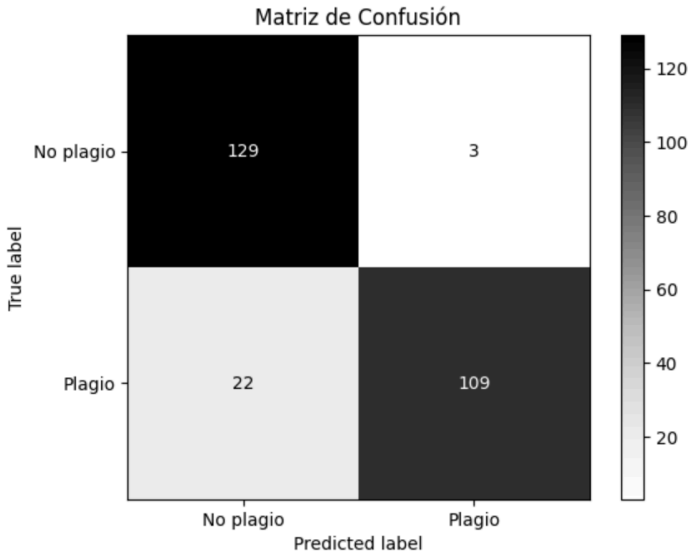
```
Reporte de clasificación:
              precision    recall  f1-score   support

   No plagio       0.85      0.98      0.91       132
      Plagio       0.97      0.83      0.90       131

    accuracy                           0.90       263
   macro avg       0.91      0.90      0.90       263
weighted avg       0.91      0.90      0.90       263
```

Esto significa que el modelo es especialmente bueno identificando correctamente casos positivos (plagio), pero se equivoca poco al clasificar como plagio algo que no lo es.

Matriz de confusión:



Comparación con el estado del arte
Lo comparamos con un enfoque del artículo “Implementation of Plagiarism Detection Using TF-IDF Vectorizer and Machine Learning Based on Flask”. Aunque también usaron TF-IDF y regresión, su accuracy fue de 68.5% y el F1-score de ~69%. Nuestro modelo superó ese rendimiento por más de 20 puntos.

Las principales razones fueron:
- Preprocesamiento mejorado
- Dataset controlado (solo código Java, sin ruido)
- Transformación simple pero efectiva

Conclusiones

Este proyecto nos dejó varios aprendizajes importantes. Aunque usamos una arquitectura sencilla, el resultado fue sólido y confiable. Demostramos que con buen preprocesamiento y un modelo interpretativo como la regresión logística, se puede lograr una herramienta eficiente y funcional.

Lo más valioso fue mantener el modelo simple pero potente. No dependimos de técnicas avanzadas ni de recursos computacionales complejos, lo cual lo hace ideal para escuelas, universidades o empresas donde se necesita verificar plagio de código de forma rápida y confiable.

Si tuviéramos más tiempo o recursos, podríamos experimentar con otros enfoques, como modelos basados en embeddings o análisis estructural del código, para ver si mejoran aún más el desempeño.

En conclusión, este modelo no solo es una buena solución técnica, sino también práctica y fácil de implementar en contextos reales donde la autenticidad del código es crítica.