Tecnológico de Monterrey

Reporte de resultados

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Adrián Matute Beltrán A01703889

Alan Fernando Razo Peña A01703350

Fermín Méndez García A01703366

Grupo 301

4 de junio del 2025

Detección de plagio en código Java

Introducción

El plagio de código es una problemática creciente en entornos académicos y profesionales, donde la reutilización no ética de software compromete la integridad intelectual y la innovación. Según estudios recientes, hasta un 34% de los casos de similitud en código Java en entornos universitarios corresponden a plagio encubierto mediante modificaciones superficiales (renombrado de variables, reordenamiento de estructuras).

Este proyecto propone un sistema de detección basado en Árboles de Sintaxis Abstracta (AST) y modelos de machine learning, que supera las limitaciones de los enfoques tradicionales (como comparación de texto plano) al analizar la estructura profunda del código. La solución se implementa en Python utilizando técnicas de procesamiento de lenguaje natural (NLP) adaptadas al código fuente.

Objetivos

Desarrollar un clasificador automático que identifique plagio en código Java mediante el análisis de similitud estructural con un umbral de precisión mayor o igual a un 85% .

Metodología

1. Data Understanding

En esta primera sección describiremos los conjuntos de datos utilizados para entrenar y evaluar cada modelo de detección de plagio en código Java. Estos datasets fueron seleccionados por su relevancia en el ámbito y su capacidad para representar casos reales de plagio.

Datasets utilizados

- IR-Plag-Dataset
    - Origen: Dataset público diseñado para la detección de plagio en código Java, utilizado como base para el entrenamiento del modelo.
    - Contenido:
        - Archivos originales: 7 ejemplos de código fuente únicos que sirvieron como base para generar casos de plagio.
        - Archivos plagiados: 355 archivos derivados de modificaciones (superficiales o estructurales) de los originales.
        - Archivos no plagiados: 105 ejemplos de código independiente, sin relación con los originales.

- FIRE14 Source Code Training Dataset
    - Origen: Dataset utilizado en competencias de detección de plagio, que incluye código fuente en múltiples lenguajes, con un enfoque en Java para este proyecto.
    - Contenido: 258 archivos de código Java, los cuales algunos contienen plagio estructural de otros archivos dentro del mismo dataset.

2. Data Preparation

Para poder entrenar un modelo capaz de detectar plagio en código fuente, fue necesario transformar los archivos Java en una representación numérica útil. Este proceso involucró varias etapas que permitieron capturar tanto el contenido textual como la estructura sintáctica del código.

Carga y combinación de datasets

Se trabajó con los conjuntos de datos principales mencionados anteriormente:

- IR-Plag-Dataset, que contiene archivos etiquetados como “original”, “plagio” y “no plagio”.
- FIRE14 Source Code Dataset, el cuál se integró en una segunda etapa del proyecto para incrementar la diversidad de ejemplos y enriquecer el aprendizaje del modelo.

Al integrar ambos conjuntos se aumentó el volumen de datos y se mejoró la generalización del modelo, en lo que se conoce como Data Augmentation.

Creación de pares etiquetados

Se construyeron pares de archivos bajo dos condiciones:

- Pares positivos: archivos donde uno es el original y el otro ha sido plagiado con alteraciones (renombrado de variables, reordenamiento de funciones, etc.).
- Pares negativos: combinaciones de archivos que no tienen relación entre sí.

Cada par fue etiquetado con un 1 si representaba plagio, y 0 en caso contrario. Esto permitió transformar el problema en una tarea de clasificación binaria.

Análisis estructural del código

Para no depender únicamente del texto plano, se aplicó un análisis estructural utilizando Árboles de sintaxis abstracta (AST). Este enfoque permite entender la lógica y jerarquía del código, independientemente de nombres de variables u otros cambios superficiales.

A partir de cada AST se extrajo una secuencia de tokens que representa la estructura del programa.

Vectorización del código

Tanto el texto del código como la secuencia de tokens obtenida de AST fueron procesados mediante técnicas de vectorización TF-IDF (Term Frequency-Inverse Document Frequency), que

permiten convertir cadenas de texto en vectores numéricos que capturan la relevancia de términos dentro del conjunto.

Extracción de características (features)

A partir de los vectores obtenidos para cada archivo del par, se calcularon métricas de similitud para construir los features del modelo. Las principales fueron:

- Similitud del coseno: mide la orientación entre vectores (aplicada en el texto y al AST)
- Similitud de Jaccard: mide la intersección sobre la unión de tokens únicos (también aplicada al texto y al AST).

De esta forma se obtuvieron cuatro características para cada par de archivos, las cuales fueron usadas como entrada para algunos de los modelos de clasificación.

División del conjunto de datos

El dataset completo fue dividido en:

- 80% para entrenamiento
- 20% para prueba

Esta división fue hecha con estrategia, de manera que ambas clases (plagio y no plagio) estuvieran representadas de manera proporcional en cada subconjunto.

Modelado

Una vez procesados los datos y extraídas las características relevantes, se procedió a construir modelos de aprendizaje automático para resolver la tarea de clasificación binaria: determinar si un par de archivos de código representa un caso de plagio o no.

Selección de algoritmos

Durante el proyecto se experimentó con distintos algoritmos de clasificación supervisada, entre los cuales están:

- Random Forest: elegido como el modelo principal por su capacidad para manejar relaciones no lineales, su resistencia al sobreajuste y su buen desempeño con múltiples características.
- XGBoost: probado como una alternativa más compleja basada en ensamblado de árboles.
- Regresión Logística: utilizado como modelo base para comparación, dada su simplicidad.

Entrenamiento de modelos

Los modelos fueron entrenados con los features derivados de las métricas de similitud entre los pares de archivos:

- Similitud de coseno entre textos planos
- Similitud de coseno entre AST
- Similitud de Jaccard entre textos planos
- Similitud de Jaccard entre AST

Cada modelo fue ajustado y validado utilizando el conjunto de entrenamiento. Posteriormente, se evaluó su rendimiento con el conjunto de pruebas para asegurar que los resultados fueran generalizables.

Evaluación del modelo

Una vez entrenados los modelos, procedimos a evaluarlos utilizando el conjunto de prueba. El objetivo fue verificar su capacidad para generalizar a nuevos datos y contrastar su rendimiento de manera objetiva.

1. **Métricas utilizadas**

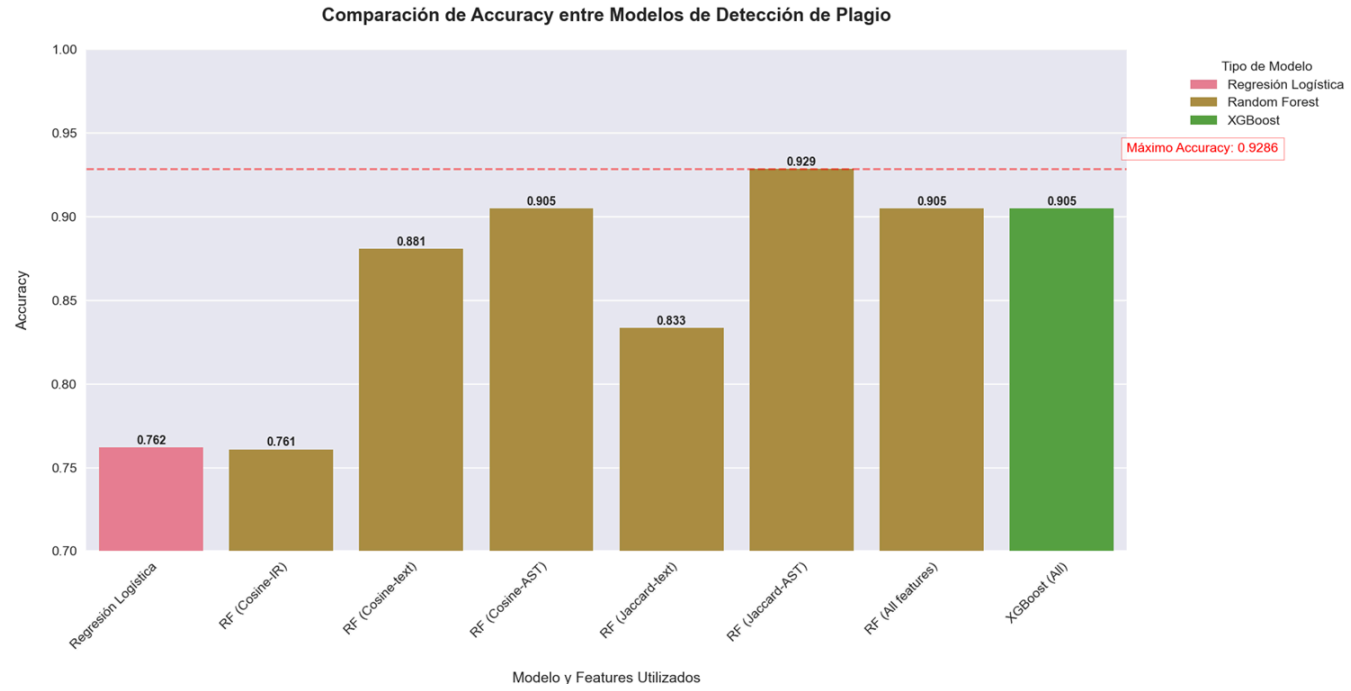
Para evaluar el desempeño de cada modelo se consideraron las siguientes métricas estándar en clasificación binaria:

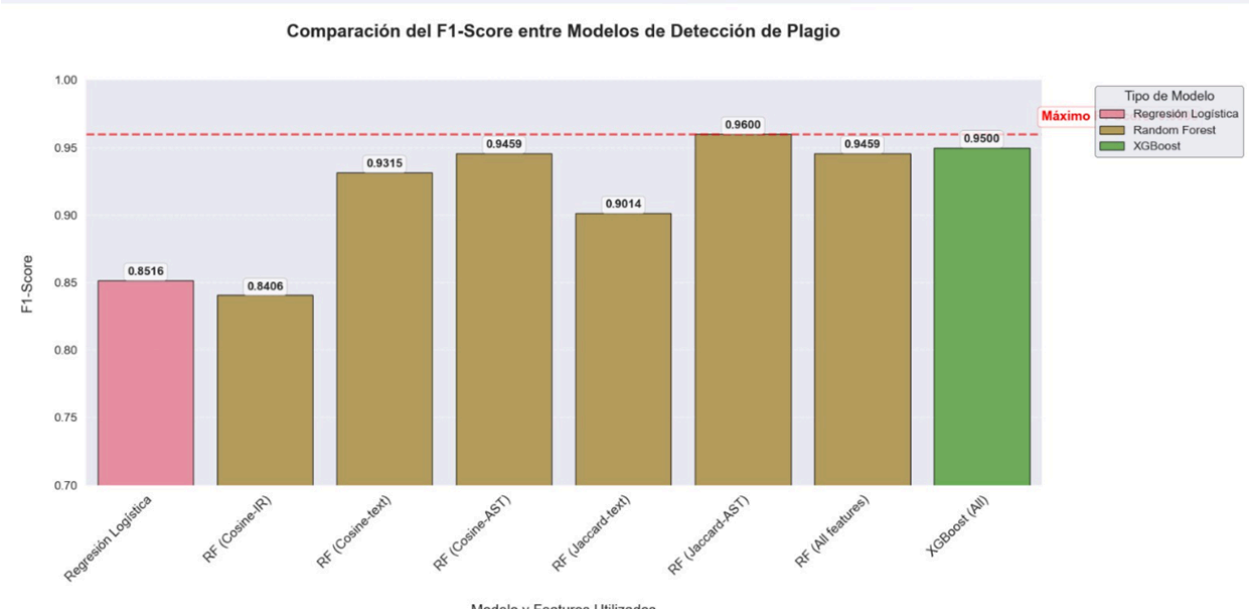
- Accuracy: proporción de predicciones correctas en total.
- Precision: proporción de verdaderos positivos entre todos los casos predichos como positivos (evita falsos positivos).
- Recall: proporción de verdaderos positivos detectados entre todos los positivos reales (evita falsos negativos).
- F1 Score: promedio ponderado entre precisión y recall, útil cuando las clases están desbalanceadas.

Num Modelo	Versión del modelo	Accuracy	Precision	Recall	F1 Score
1	Regresión Logística	0.7619	0.9667	0.7632	0.8516
2	Random Forest - Cosine similarity - AST (Solo IR - PLAG)	0.7609	0.8657	0.8169	0.8406
3	Random Forest - Cosine similarity text	0.8810	0.9714	0.8947	0.9315
4	**Random Forest - Cosine similarity - AST**	**0.9048**	**0.9722**	**0.9211**	**0.9459**
5	Random Forest - Jaccard similarity text	0.8333	0.9697	0.8421	0.9014
6	**Random Forest - Jaccard similarity - AST**	**0.9286**	**0.9730**	**0.9474**	**0.9600**
7	Random Forest- All features	0.9048	0.9722	0.9211	0.9459
8	Modelo XGboost	0.9048	0.9048	1.000	0.9500

Los mejores resultados globales se obtuvieron en el modelo Random Forest con Jaccard Similarity sobre AST, con una F1 Score de 0.96, precisión de 0.9730 y recall de 0.9474.

Tabla comparativa de resultados de los diversos modelos en términos de Accuracy y F1-Score:





Enter accuracy y F1 Score podemos ver que RF jaccard AST es el mejor modelo y también podemos destacar el alto desempeño que tienen lo AST-based models

Bitácora de ejecución.

Para encontrar el mejor de los modelos intentamos con dos features importantes cosine similarity tokenizando con TF - IDF y jaccard similarity. Estas medidas fueron aplicadas a los códigos directamente y después a las cadenas tokenizadas de los AST. De esta manera tenemos disponibles 4 features:

1- Cosine similarity text
2- Cosine similarity - AST
3- Jaccard similarity text
4- Jaccard similarity - AST

MODELO RF- Jaccard similarity AST

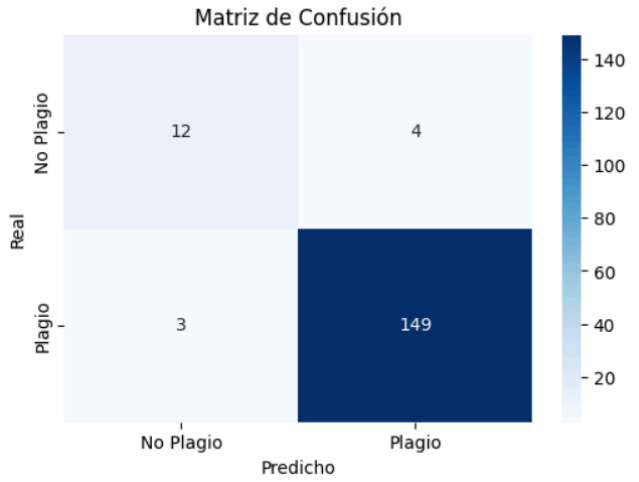
Evaluación en conjunto de entrenamiento:

Accuracy: 0.9583

Precision: 0.9739

Recall: 0.9803

F1 Score: 0.9770



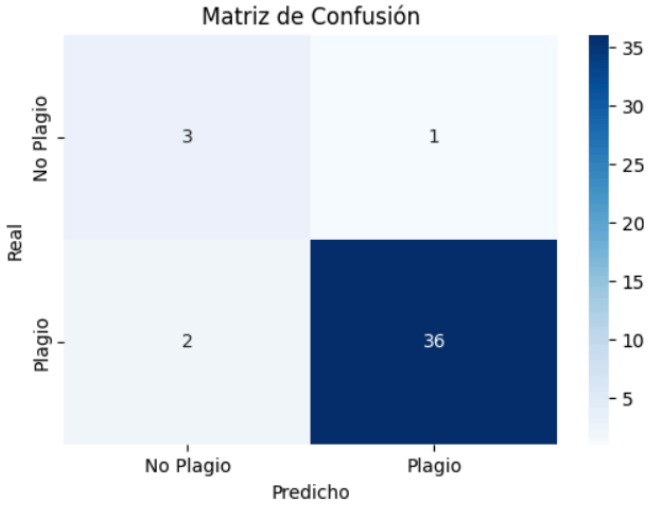
Evaluación en conjunto de prueba:

Accuracy: 0.9286

Precision: 0.9730

Recall: 0.9474

F1 Score: 0.9600



MODELO RF- COSINE SIMILARITY raw text

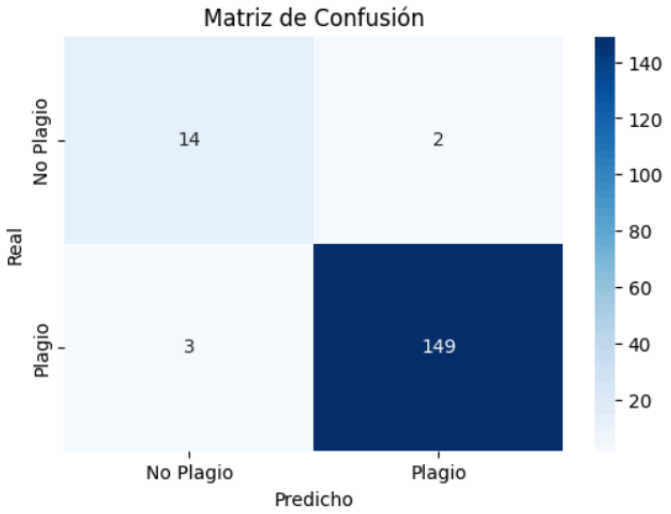
Evaluación en conjunto de entrenamiento:

Accuracy: 0.9702

Precision: 0.9868

Recall: 0.9803

F1 Score: 0.9835



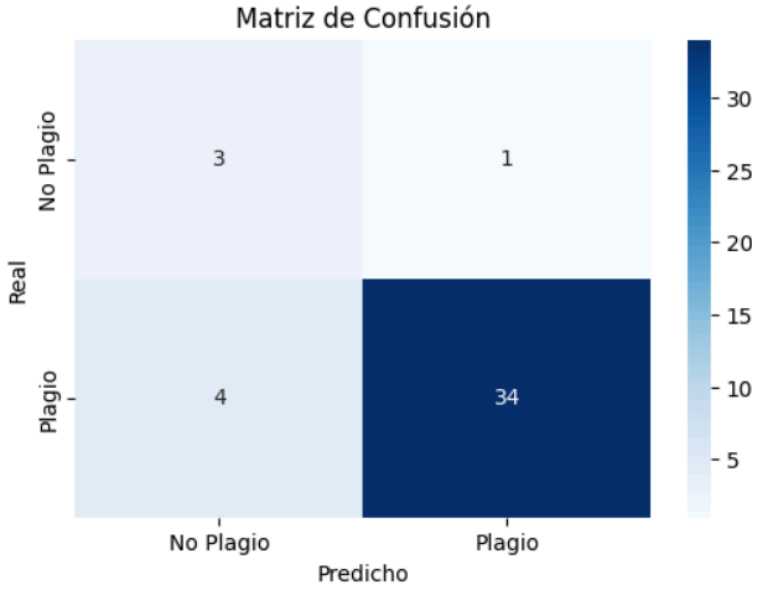
Evaluación en conjunto de prueba:

Accuracy: 0.8810

Precision: 0.9714

Recall: 0.8947

F1 Score: 0.9315



MODELO RF- COSINE SIMILARITY AST

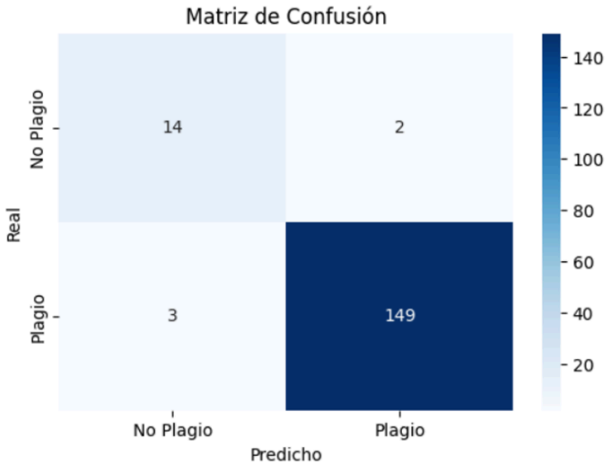
Evaluación en conjunto de entrenamiento:

Accuracy: 0.9702

Precision: 0.9868

Recall: 0.9803

F1 Score: 0.9835



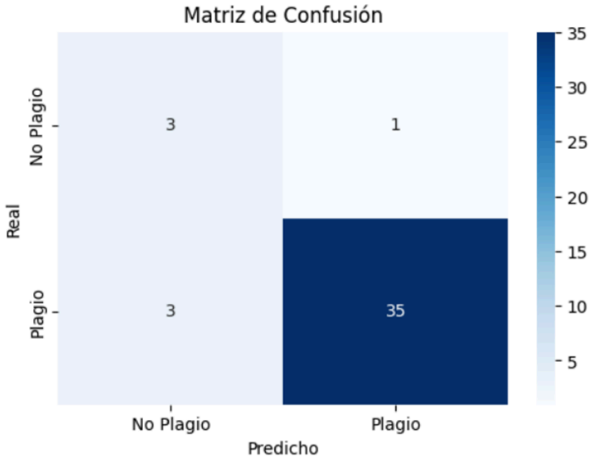
Evaluación en conjunto de prueba:

Accuracy: 0.9048

Precision: 0.9722

Recall: 0.9211

F1 Score: 0.9459



MODELO RF- Jaccard similarity raw text

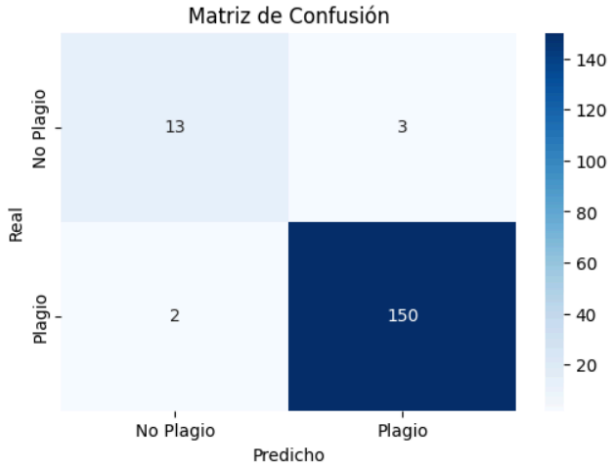
Evaluación en conjunto de entrenamiento:

Accuracy: 0.9702

Precision: 0.9804

Recall: 0.9868

F1 Score: 0.9836



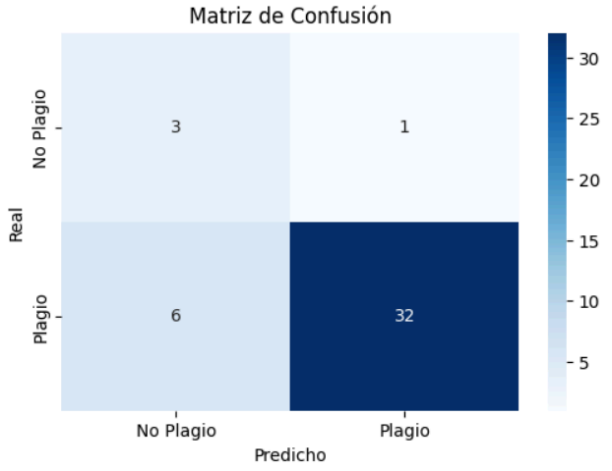
Evaluación en conjunto de prueba:

Accuracy: 0.8333

Precision: 0.9697

Recall: 0.8421

F1 Score: 0.9014



MODELO RF- 4 features

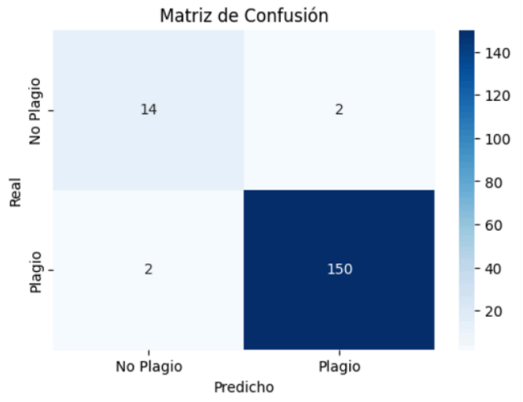
Evaluación en conjunto de entrenamiento:

Accuracy: 0.9762

Precision: 0.9868

Recall: 0.9868

F1 Score: 0.9868



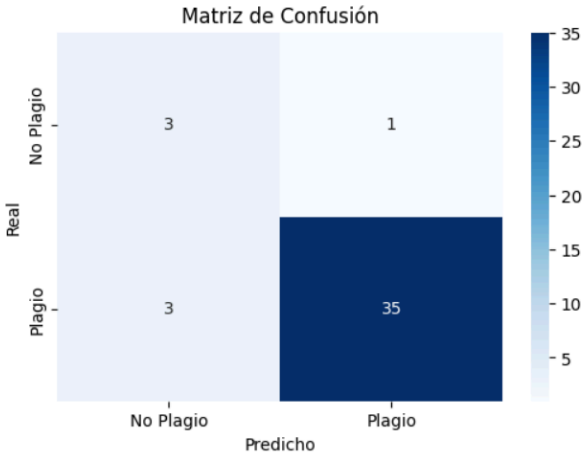
Evaluación en conjunto de prueba:

Accuracy: 0.9048

Precision: 0.9722

Recall: 0.9211

F1 Score: 0.9459



Modelo XGboost

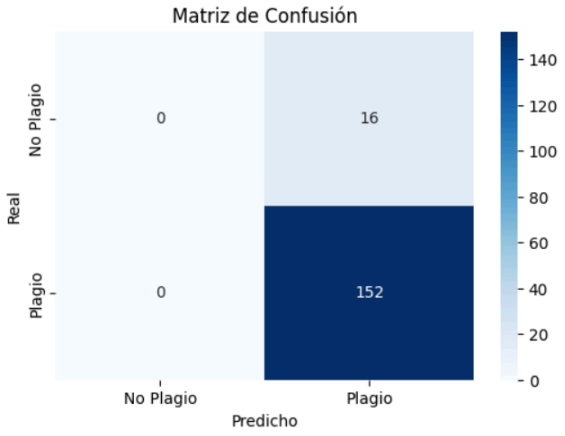
Evaluación en conjunto de entrenamiento:

Accuracy: 0.9048

Precision: 0.9048

Recall: 1.0000

F1 Score: 0.9500



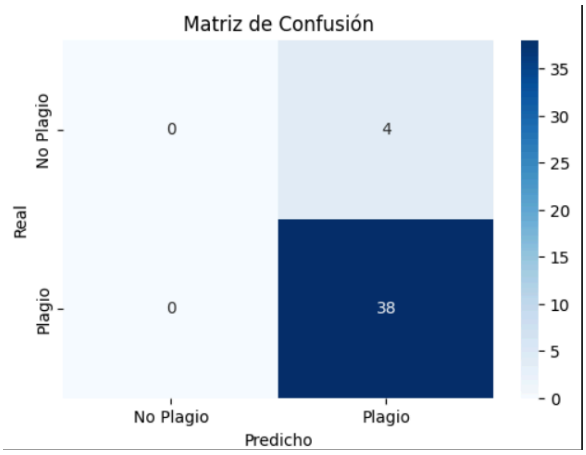
Evaluación en conjunto de prueba:

Accuracy: 0.9048

Precision: 0.9048

Recall: 1.0000

F1 Score: 0.9500



Regresión logística simple con los cuatro features

```
Modelo: LogisticRegression - Versión: v1_0
[[ 3  1]
 [ 9 29]]
              precision    recall  f1-score   support

           0     0.2500    0.7500    0.3750         4
           1     0.9667    0.7632    0.8529        38

    accuracy                         0.7619        42
   macro avg     0.6083    0.7566    0.6140        42
weighted avg     0.8984    0.7619    0.8074        42
```

De acuerdo con Anti-plagiarism tool helping developers to generate authentic code un punto de comparación del estado del arte es:

Algorithm	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Lexical Similarity	75%	70%	65%	67.5%	0.75
Cosine Similarity	80%	75%	70%	72.5%	0.80
AST-based Similarity	90%	85%	80%	82.5%	0.90

Conclusiones

El desarrollo del modelo de detección de plagio en código fuente Java demostró que es posible identificar similitudes estructurales y semánticas con alta precisión, incluso en casos donde se aplican técnicas comunes para disfrazar el plagio(como renombrado de variables o reordenamiento de funciones). Superando el umbral propuesto del 85% de precisión y mostrando métricas destacables en múltiples configuraciones del modelo.

Entre los principales hallazgos se concluyó lo siguiente:

- Los modelos basados en AST (Abstract Syntax Tree) mostraron un desempeño superior, tanto en nuestra implementación como en los enfoques revisados en el estado del arte. Esta representación estructural del código permite detectar plagio más allá de simples coincidencias textuales.
- En particular, la combinación de Jaccard similarity con AST fue la más efectiva, superando incluso a combinaciones con cosine similarity. Esto sugiere que el análisis de intersección de estructuras sintácticas es más representativo para este tipo de tarea
- El modelo desarrollado de Random Forest con estas características obtuvo una F1 Score de 0.96 y un recall de 94.74%, lo que refleja su alta capacidad para detectar casos reales de plagio sin generar excesivos plagios positivos.
- Se identificaron áreas de oportunidad en el dataset, especialmente en la representación de la clase “no plagio”, que se encuentra poco representada. Esto puede llegar a impactar el balance del modelo en ciertos contextos reales.

Bibliografía

[1] T. Pradeep, A. K. Gonepally, A. K. Pusala, and C. T. Singarapu, “Anti-plagiarism tool helping developers to generate authentic code,” *International Journal of Science and Research Archive*, Jan. 2025. [Online]. Available: https://journalijsra.com

[2] F. Ebrahim and M. Joy, “Semantic similarity search for source code plagiarism detection: An exploratory study,” in *Proc. 2024 on Innovation and Technology in Computer Science Education (ITiCSE)*, V. 1, pp. 360–366, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3649217.3653622

[3] N. Sharma, S. Shinde, S. Bhosale, and S. Patil, “SourcePlag – Source code plagiarism detection based on abstract syntax trees,” in *Proc. 2024 IEEE Int. Conf. Blockchain and Distributed Systems Security (ICBDS)*, Pune, India, 2024, pp. 1–5, Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10837209