Optimizing Plagiarism Detection with Random Forest and Markov Chains via Token Preprocessing Techniques.

Ariann Fernando Arriaga Alcántara, Cristian Rogelio Espinosa Diaz, Aldo Tena García.

1. Abstract

The objective of the present project is to create a model that can identify and differentiate between codes whose sources are similar. This involves detecting the most similar source code pairs within a large collection of source codes. To achieve this, you need to develop the following:

A computational model that uses Compilers, Quantitative Methods and Machine Learning tools to detect source code pairs that have been reused from a given collection (FIRE14 and CONPLAG). As mentioned above, multiple representative data (7 different values obtained from both codes and the verdict within the data set) obtained from various processing techniques of the files received as input to the model in order to determine whether they have signs of plagiarism or not. And the effectiveness of the model will be compared with the previously proposed metrics. It is expected that the performance of the proposed model will fall within the previously established ranges of the metrics to be evaluated. In summary, the goal is to create an advanced computational system that can identify patterns and similarities between source codes, in order to improve the detection of code plagiarism.

After performing multiple tests with 3 different models with 2 different sets of preprocessed data positive results were obtained, all values that were used to evaluate (accuracy, precision, recall and the f1 score) the model are within the desired range (all with a value above of 80%)

2. Planteamiento del problema

El plagio de software, es una práctica donde se copia ilegalmente el trabajo de otros o se aprovecha del código abierto para disfrazarlo como original, representa una amenaza significativa en una industria de rápido crecimiento. Los piratas informáticos replican la lógica del software original, lo que resulta en enormes pérdidas económicas para las empresas de software. Esta actividad constituye una clara violación de los derechos de autor del software original, ya que los piratas no solo replican el producto, sino que también lo modifican y lo comercializan. A menudo, recurren a la ingeniería inversa para acceder al código fuente original, exacerbando aún más el problema.

El informe de Software Mundial de la Business Software Alliance (BSA) de 2016 reveló que la tasa de piratería de software alcanzaba el 39 %, generando daños económicos masivos estimados en alrededor de $52,200 millones de dólares. Estas cifras subrayan la urgente necesidad de implementar medios efectivos para detectar y combatir el plagio de software. Sorprendentemente, estudios han demostrado que una gran proporción de software, en algún lugar entre el 5 % y el 20 %, contiene elementos plagiados. Esto resalta la necesidad de herramientas de detección de plagio eficaces y sostenibles en el tiempo.

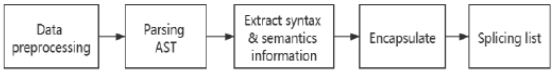
Para abordar este desafío, se han desarrollado diversas herramientas de detección de plagio, como sistemas de análisis de similitud de código fuente, soluciones para la identificación de errores de software, y métodos para la detección de clones y marcas de software. Estas soluciones se centran principalmente en el análisis exhaustivo del texto y la estructura del código, buscando identificar patrones y similitudes que indiquen posibles casos de plagio. Sin embargo, mantener y actualizar estas herramientas representa un desafío constante, dada la evolución continua de las prácticas de piratería y la complejidad creciente de los sistemas de software.

3. Antecedentes

Para abordar este desafío, se han desarrollado diversas herramientas de detección de plagio, como sistemas de análisis de similitud de código fuente. Al igual que soluciones para la identificación de errores de software, y métodos para la detección de clones y marcas de software. Estas soluciones se centran principalmente en el análisis exhaustivo del texto y la estructura del código, buscando identificar patrones y similitudes que indiquen posibles casos de plagio. Sin embargo, mantener y actualizar estas herramientas representa un desafío constante, dada la evolución continua de las prácticas de piratería y la complejidad creciente de los sistemas de software.

Después de recolectar información de distintos artículos y publicaciones científicas se observó que una gran parte de las fuentes compartían como punto de partida el preprocesamiento del set de datos por medio de la generación de un árbol de sintaxis abstracta (AST) para

poder usar posteriormente una representación vectorial del código fuente en múltiples funciones. El uso de Árboles Sintácticos Abstractos (AST) para representar la estructura del código fuente de manera abstracta. Los AST permiten superar los retos de aplicar y transformar datasets en estructuras adecuadas para entrenar modelos de machine learning, proporcionando una representación sintética de los elementos presentes en el código. Que permiten obtener datos representativos adecuados para el entrenamiento de un modelo o su análisis por medio de algoritmos como K-Means presentadas por Y. Xie et al [1].



[5] Source code conversion steps

El núcleo del sistema es el procesamiento del código fuente y su conversión a un Árbol Sintáctico Abstracto (AST), posteriormente se utilizarán múltiples librerías y funciones desarrolladas por el equipo para poder convertir este árbol de sintaxis abstracta en una matriz de transiciones que permitirá obtener múltiples valores representativos que podrán ser analizados por múltiples modelos en las etapas posteriores.

El artículo científico de Y. Xie et al. [1] proporciona un ejemplo de transformación que ilustra cómo se convierten los códigos de Python en AST para los sets de datos. Esta transformación es útil para análisis como la detección de duplicación de código. Para ello, se emplean herramientas externas que transforman el código Python en bytecode interpretable por el modelo.

El artículo científico de Fahad Ebrahim et al. [3] trata sobre el uso de SOCO para la clasificación de código fuente mediante representaciones vectoriales que capturan tanto la sintaxis como la semántica del texto, conocidas como incrustaciones contextuales. Estas incrustaciones se generan utilizando modelos preentrenados (CodePTMs). La selección de clasificadores y el ajuste de parámetros se realizan con la ayuda de Machine Learning Automatizado (AutoML).

Las pruebas iniciales se realizaron con código fuente en Java. El proceso implica convertir el código fuente a vectores antes de ingresarlos al clasificador. Se han explorado varias implementaciones utilizando Redes Neuronales Artificiales (ANN), Árboles Sintácticos Abstractos (AST), y Redes Siamese, demostrando un enfoque de Machine Learning aplicado a un problema de clasificación.

El artículo científico de A. Kurtukova et al.[2] trata sobre la identificación del autor del código fuente utilizando una técnica basada en una red neuronal híbrida (HNN). Para evaluar la efectividad de diferentes arquitecturas de redes neuronales en la identificación del autor, se realizaron experimentos con un conjunto de datos compuesto por más de 200,000 muestras de código escritas por 898 autores en 13 lenguajes de programación diferentes.

Los resultados de los experimentos indicaron que la combinación de una Red Neuronal Convolucional (CNN) con una Red Neuronal Recurrente Bi-GRU (Bidirectional Gated Recurrent Unit) fue la más efectiva, alcanzando una precisión promedio superior al 80% para todos los lenguajes de programación evaluados. Esta técnica mostró ser robusta incluso en casos de ofuscación y cumplimiento de estándares de codificación.

El artículo científico de Awale N et al. [5] trata sobre la detección de plagio en código fuente mediante diversos atributos y técnicas. En primer lugar, se aborda el uso de n-gramas en el procesamiento de texto para representar datos secuenciales, como programas informáticos. Los n-gramas, que son secuencias contiguas de longitud n, se utilizan para capturar la información sobre el orden de las palabras, superando las limitaciones del modelo de bolsa de palabras.

Para representar los datos secuenciales, los n-gramas pueden calcularse a nivel de caracteres, por ejemplo, los 3-gramas del texto "range(2)" son {ran, ang, nge, ge(, e(2, (2}. Los n-gramas se asignan a índices utilizando un diccionario o mediante el “Truco del hashing”, evitando la necesidad de construir un diccionario. Además, se aplica el esquema de ponderación tf-idf para escalar las características, teniendo en cuenta la frecuencia de los términos en un documento y su frecuencia inversa en el conjunto de datos.

El artículo científico de Heres D et al. [6] trata sobre la detección de similitudes entre códigos fuente mediante diversas técnicas, incluyendo la implementación de un puntaje de similitud utilizando el algoritmo de Karp-Rabin. Se consideran varios aspectos del estilo de codificación, tales como el uso de corchetes, espacios, saltos de línea, comentarios, la cantidad de líneas de código y el número de variables y funciones sin utilizar, conocido como “código muerto”.

4. Solución propuesta

Después de hacer un extenso análisis de todas las diferencias y similitudes entre los diferentes artículos consultados se dividió el problema en 4 diferentes etapas que abarcan múltiples actividades, estas etapas son: limpieza del set de datos, preprocesado de los datos, procesamiento de los datos y la optimización del modelo.

Los datasets originales de FIRE14 y CONPLAG en los archivos tenían documentación sobre los pares de código y el criterio sobre el plagio entre ellos. No obstante, varios códigos eran utilizados varias veces en diversas parejas en la documentación de los archivos .csv.

Esto fue un factor a considerar debido a que si se utilizaban librerías para la división aleatoria de los datos para la fase de entrenamiento y pruebas, podría darse el

caso de que los datos de las fases sean contaminadas. En otras palabras, se podría presentar la situación de que el modelo sea expuesto al mismo código en ambas fases, lo cuál contrarresta el propósito de la fase de prueba, donde el modelo es expuesto a datos nunca antes vistos.

Para manejar esta situación, el equipo desarrolló la división de manera manual, creando un archivo csv para cada fase, asegurando que los códigos usados en entrenamiento no sean utilizados en prueba.

Para poder obtener valores representativos adecuados para entrenar los modelos de la siguiente etapa se tiene que hacer múltiples operaciones de preprocesado con los sets de datos ya adecuados. En primera instancia se usó la biblioteca javalang para poder convertir el contenido de un archivo de JAVA a una representación de los tokens contenidos en el mismo, a partir de estos tokens se puede construir un árbol de sintaxis abstracta que refleja en gran medida la estructura y el contenido del archivo.

Después de obtener el árbol de sintaxis, se usaron múltiples funciones desarrolladas con el objetivo de generar una cadena de Markov para generar una matriz de probabilidad de transiciones. Esta matriz se usará para obtener la similitud del coseno entre ambas matrices. Posteriormente, se convierte la matriz obtenida se convierte en una matriz unidimensional, es decir, un vector que en junto a su contraparte del otro archivo de JAVA pueden ser introducidos como parámetros a funciones a la librería scipy para poder obtener la distancia euclidiana, la distancia de jaccard y la distancia de Manhattan. Se realiza la resta de estos 2 vectores para obtener un nuevo vector que representa esta diferencia y se analizan todos los comentarios presentes en ambos archivos mediante una matriz de transiciones para obtener la distancia del coseno y la similitud por medio de cadenas de Markov.

Después de obtener todos los valores representativos de ambos códigos estos se almacenan en un vector, este vector tiene una contraparte en la que se almacenan los veredictos contenidos en los sets de datos iniciales, ambos serán usados para el entrenamiento de múltiples modelos de machine learning. Los modelos seleccionados para el desarrollo fueron Logistic Regression, Random Forest y XGBoost con base a lo investigado en la literatura. Estos modelos fueron seleccionados por sus ventajas a la hora de procesar vectores con datos representativos dentro de una gran parte de los artículos consultados para el desarrollo de este modelo.

Cada uno de los modelos paso por una búsqueda de hiperparametros adecuados haciendo uso de funciones como Grid Search CV para lograr encontrar hiperparametros adecuados para el correcto desarrollo y entrenamiento del modelo con base en los datos disponibles para el entrenamiento del modelo, los valores obtenidos por estas funciones fueron aplicados en las primeras etapas de entrenamiento del modelo, a pesar de tener estos hiperparámetros ya integrados en el entrenamiento se procedió a buscar más posibles hiperparámetros para obtener mejores resultados.

5. **Metodología**

Una vez definido el pre procesado de los datos y los modelos a utilizar, se realizó una segmentación de los datos para que fuesen representativos, donde se cuenta con un set de datos de entrenamiento y uno para la ejecución de pruebas, ambos se encuentran en proporciones de 80% y 20% respectivamente. Posteriormente, para optimizar el rendimiento de los modelos de machine learning propuestos, se llevó a cabo una experimentación con hiperparámetros utilizando Grid Search CV (Validación Cruzada de Búsqueda en Cuadrícula). Primero, se definió el espacio de hiperparámetros para cada modelo. Para la Regresión Logística, se consideraron parámetros como la penalización (penalty), la regularización (C), la tolerancia (tol), la interceptación (fit_intercept), el algoritmo de optimización (solver) y el número máximo de iteraciones para el modelo (max_iter).

Para el modelo Random Forest, los parámetros clave incluyeron el número de árboles (n_estimators), la profundidad máxima del árbol (max_depth), el número mínimo de muestras requeridas para dividir un nodo (min_samples_split), el número mínimo de muestras en una hoja (min_samples_leaf) y la disminución mínima de la impureza (min_impurity_decrease). En el caso de XGBoost, se exploraron parámetros como el número de árboles (n_estimators), la tasa de aprendizaje (learning_rate), la profundidad máxima (max_depth), la fracción de muestras utilizadas para entrenar cada árbol (subsample) y la regularización L2 (reg_lambda).

Se implementó Grid Search CV con validación cruzada de 5 pliegues para explorar todas las combinaciones posibles de estos hiperparámetros. Este enfoque permitió evaluar el rendimiento del modelo de manera robusta, asegurando que los resultados no dependieran de una única partición de los datos.

Tras realizar la búsqueda, se seleccionaron los mejores conjuntos de hiperparámetros basados en la métrica de validación elegida, como lo es el **Acuraccy**. Los parámetros óptimos encontrados fueron:

Modelo	**Hiperparámetro**	**Valor**
Regresión Logística	**penalty**	l2
	C	0.1

	tol	0.0001
	fit_intercept	True
	solver	lbfgs
	max_iter	100
	class_weight	None

Modelo	**Hiperparámetro**	**Valor**
Random Forest	**n_estimators**	200
	max_depth	20
	min_samples_split	2
	min_samples_leaf	1
	min_impurity_decrease	0.001

Modelo	**Hiperparámetro**	**Valor**
XGBoost	**n_estimators**	200
	learning_rate	0.1
	max_depth	6
	subsample	0.8
	reg_lambda	0.001

Con estos hiperparámetros óptimos, se reentrenaron los modelos para asegurar que proporcionarán el mejor rendimiento posible.

Después de usar múltiples hiperparámetros para poder obtener los mejores resultados posibles, se vio que los resultados no estaban dentro del umbral establecido para considerarlos adecuados. Para solucionar esto se dio un paso atrás en el proceso de análisis y se decidió cambiar la creación de la matriz de probabilidad de transiciones a partir del AST para que esta matriz fuera generada a partir de los tokens obtenidos del código.

Después de realizar este cambio en la etapa de preprocesamiento se pudo observar un aumento del 2-4% dentro de múltiples métricas generales.

6. Evaluación de solución

Para determinar la efectividad del modelo propuesto, se evaluarán las siguientes métricas, las cuales aparecen de forma recurrente en la literatura investigada:

Accuracy: Esta métrica ayuda a determinar el porcentaje total de predicciones correctas. Es relevante para tener un panorama general sobre el porcentaje de predicciones correctas en modelos de clasificación. De acuerdo con A. Kurtukova et al. [2], la fórmula para determinar esta métrica es:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

donde:

- **TP** es positivo verdadero, el número de casos donde un modelo predijo correctamente la clase positiva.
- **FP** es falso positivo, el número de casos donde un modelo predijo la clase negativa incorrectamente.
- **TN** es verdadero negativo, el número de casos donde un modelo predijo correctamente la clase negativa.
- **FN** es falso negativo, el número de casos donde un modelo predijo la clase negativa incorrectamente.

Precision: Esta métrica sirve para evaluar la exactitud de un modelo para la predicción de las clases, normalmente usada en modelos de clasificación binaria. De acuerdo con Fahad Ebrahim et al. [3], la fórmula para determinar esta métrica es:

$$\text{Precision} = \frac{TP}{TP+FP}$$

donde:

- **TP** es positivo verdadero, el número de casos donde un modelo predijo correctamente la clase positiva.
- **FP** es falso positivo, el número de casos donde un modelo predijo la clase negativa incorrectamente.

Recall: Esta métrica mide la capacidad de un modelo para identificar correctamente todos los casos positivos reales. Evalúa la habilidad de un modelo para detectar casos positivos reales sobre el total de casos positivos disponibles. De acuerdo con Fahad Ebrahim et al. [3], la fórmula para determinar esta métrica es:

$$\text{Recall} = \frac{TP}{TP+FN}$$

donde:

- **TP** es positivo verdadero, el número de casos donde un modelo predijo correctamente la clase positiva.
- **FN** es falso negativo, se refiere a los casos donde un modelo predice la clase negativa incorrectamente.

F1-score: Esta métrica es relevante en la evaluación de un modelo cuando el conjunto de datos es desbalanceado, es decir, cuando un modelo de clasificación binaria fue entrenado con un conjunto de datos donde una clase es mucho más frecuente que la otra. De acuerdo con Fahad Ebrahim et al. [3], la fórmula para determinar esta métrica es:

$$\text{F1-score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Debido a esta fórmula, se considera al F1-score como la media armónica de precision y recall.

La solución propuesta será evaluada con respecto a estas métricas para determinar su calidad en las predicciones de clasificación sobre si se trata o no de plagio en los códigos analizados.

7. Contribuciones Esperadas

El trabajo de investigación realizado tiene como fin el desarrollo de un modelo de machine learning que realice una detección de plagio en dos códigos presentados escritos en lenguaje JAVA. A partir de estos dos códigos, el sistema desarrollado preprocesará los códigos para que el modelo realice una predicción a partir de esto. Con la predicción realizada por el modelo entrenado se van a clasificar los pares de códigos como plagio o no plagio.

El producto de esta investigación será un sistema desarrollado en python que puede preprocesar un par de códigos y a partir de un modelo desarrollado de machine learning realiza una clasificación para el plagio con base a una predicción.

8. Resultados

En la primera iteración, se utilizó la primera versión del preprocesado de los datos. A su vez, se implementaron los algoritmos sin el uso de hiperparámetros, se obtuvieron los siguientes resultados:

Metrics	Logistic Regression	Random Forest	XGBoost
Accuracy (train)	76%	97%	100%
Accuracy (test)	72%	80%	80%
Accuracy (validation)	75%	80%	80%
Precision	72%	83%	80%
Recall	72%	80%	80%
F1-score	70%	78%	78%
True positives	34	42	48
True negatives	150	162	155
False positives	15	3	10
False negatives	56	48	42

En la segunda iteración, se utilizó la primera versión del preprocesado de los datos. De igual forma, se implementaron los algoritmos con el uso de hiperparámetros, donde se escogieron los valores con mejor rendimiento acorde al uso de Grid Search CV:

Metrics	Logistic Regression	Random Forest	XGBoost
Accuracy (train)	75%	99%	95%
Accuracy (test)	73%	82%	80%

Accuracy (validation)	75%	81%	81%
Precision	76%	85%	82%
Recall	73%	82%	73%
F1-score	68%	80%	75%
True positives	26	46	45
True negatives	160	163	158
False positives	5	2	7
False negatives	64	44	45

En la tercera iteración, se realizó una experimentación con ligeras modificaciones a los hiperparámetros implementados en los modelos.

Metrics	Logistic Regression	Random Forest	XGBoost
Accuracy (train)	75%	83%	83%
Accuracy (test)	72%	76%	81%
Accuracy (validation)	75%	78%	79%
Precision	85%	86%	88%
Recall	61%	66%	73%
F1-score	59%	67%	75%
True positives	20	30	43
True negatives	165	165	164
False positives	0	0	1
False negatives	70	60	47

En la última iteración, se realizaron modificaciones en la generación de las características relevantes. No obstante, se mantuvieron los mismos hiperparámetros de la tercera iteración.

Metrics	Logistic Regression	Random Forest	XGBoost
Accuracy (train)	76%	97%	100%
Accuracy (test)	72%	86%	84%
Accuracy (validation)	76%	80%	81%
Precision	81%	86%	84%
Recall	72%	83%	84%
F1-score	66%	84%	83%
True positives	19	65	60
True negatives	165	154	154
False positives	0	11	11
False negatives	71	25	30

Los resultados mostrados, así como los códigos desarrollados para la solución propuesta, se encuentran en el siguiente repositorio: https://github.com/AriannFernando/tc3002b-Plagiarism-Detector

6. Bibliografía

[1] Y. Xie, W. Zhou, H. Hu, Z. Lu, and M. Wu, "Code Similarity Detection Technique Based on AST Unsupervised Clustering Method," 2020 IEEE 6th International Conference on Computer and Communications (ICCC), pp. 101–114, Dec. 2020.

[2] A. Kurtukova, A. Romanov, and A. Shelupanov, "Source Code Authorship

Identification Using Deep Neural Networks," Symmetry, vol. 12, no. 12, p. 2044, Dec. 2020.

[3] Fahad Ebrahim and Mike Joy. Source Code Plagiarism Detection with Pre-Trained Model Embeddings and Automated Machine Learning. In Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing, pages 301–309, 2023.

[4] M. Duracik, P. Hrkut, E. Krsak and S. Toth, "Abstract Syntax Tree Based Source Code Anti Plagiarism System for Large Projects Set," in IEEE Access, vol. 8, pp. 175347-175359, 2020

[5] Awale N, Pandey M, Dulal A, Timsina B. Plagiarism detection in programming assignments using machine learning. Journal of Artificial Intelligence and Capsule Networks. vol. 2, n.º 3, July 2020

[6] Heres D. Source Code Plagiarism Detection using Machine Learning. Utrecht University. Aug