



Detector de plagio en archivos de código Java

TC3002B: Desarrollo de aplicaciones
avanzadas de ciencias computacionales



Mónica Andrea Ayala Marrero
A01707439
Olivia Araceli Morales Quezada
A01707371
Josemaría Robledo Lara
A01612376



Problemática & Objetivo

Estado del Arte

Autor	Lenguaje de Programación	Metodología	Resultados
M. Duracik, M. Callejas-Cuervo y M. Mikusova, 2020	C#	K-Means	N/A
H. Cheers y Y. Lin. 2022	Java	Puntaje de Similitud	N/A
N. Awale, M. Pandey, A. Dulal y B. Timsin, 2020	c y C++	Clasificador Xgboost, Hashes	Accuracy: 94% F1: 0.90
Y. Wu, S. Feng, D. Zou y H. Jin, 2023	Java	AST, Cadenas de Markov + Clasificador (KNN, RF, DT)	F1: 0.95
Wu, J.S., Chien, T. H., Chien, L.R y Yang, C. Y, 2021	C	AST y CNN	Accuracy: 92%
N. Viuginov, P. Grachev y A. Filchenkov, 2020	C++	AST y Clasificador (RF, Xgboost)	F1: 0.745



Elección de Datasets


FIRE14 fue originalmente diseñado para la detección de reutilización de código fuente en el evento PAN de FIRE 2014. ConPlag es un conjunto de datos específicamente diseñado para evaluar herramientas de detección de plagio en el contexto de concursos de programación Java.

Integración de Datos

Para enriquecer el entrenamiento y la evaluación de la herramienta, los datos de ConPlag se combinaron con los de FIRE14. Esto permite probar la herramienta en un espectro más amplio de situaciones de plagio, aumentando la robustez del sistema.

Preprocesamiento

Por medio de un script y archivos CSV se indican los códigos a comparar, se dividen los datos en conjuntos de entrenamiento y prueba, y almacenan los datos preparados en archivos .npy para su uso posterior en la implementación de modelos de aprendizaje automático.



Solución propuesta

Matriz de transición de estados

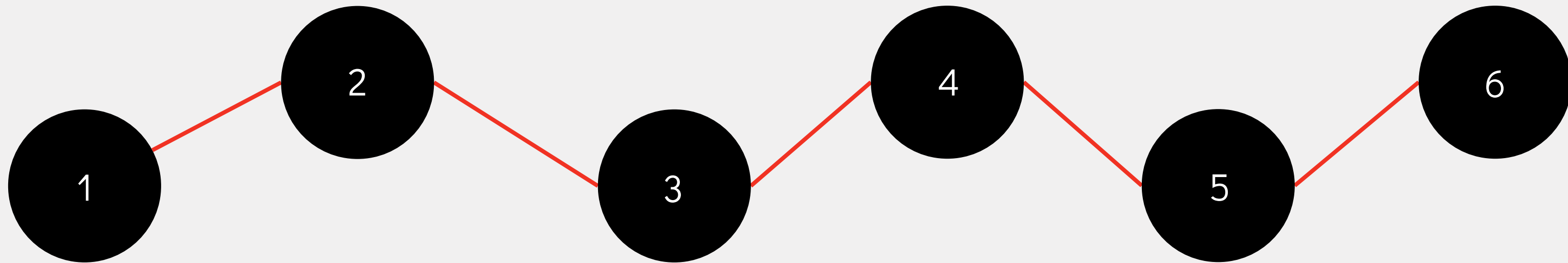
Recorremos el AST haciendo un conteo de transiciones de un tipo de nodo a otros

Métricas de distancia

Calculamos cosine similarity, pairwise distances (euc, che, man) y n_grams similarity

Clasificador

Pasamos los datos de X_train y y_train a modelos de clasificación (RF, Xgboost)



Árbol Sintáctico

Utilizando Javalang obtenemos el árbol sintáctico de ambos códigos en Java

Normalización y Vectorización

Normalizamos la matriz y la vectorizamos (teniendo en cuenta todos los nodos posibles en el dataset)

Generación del Dataset

Creamos un array X con nuestros vectores normalizados y las métricas de distancia y un array Y con el veredicto.

Árbol sintáctico

Java Code

```
public class C {
```

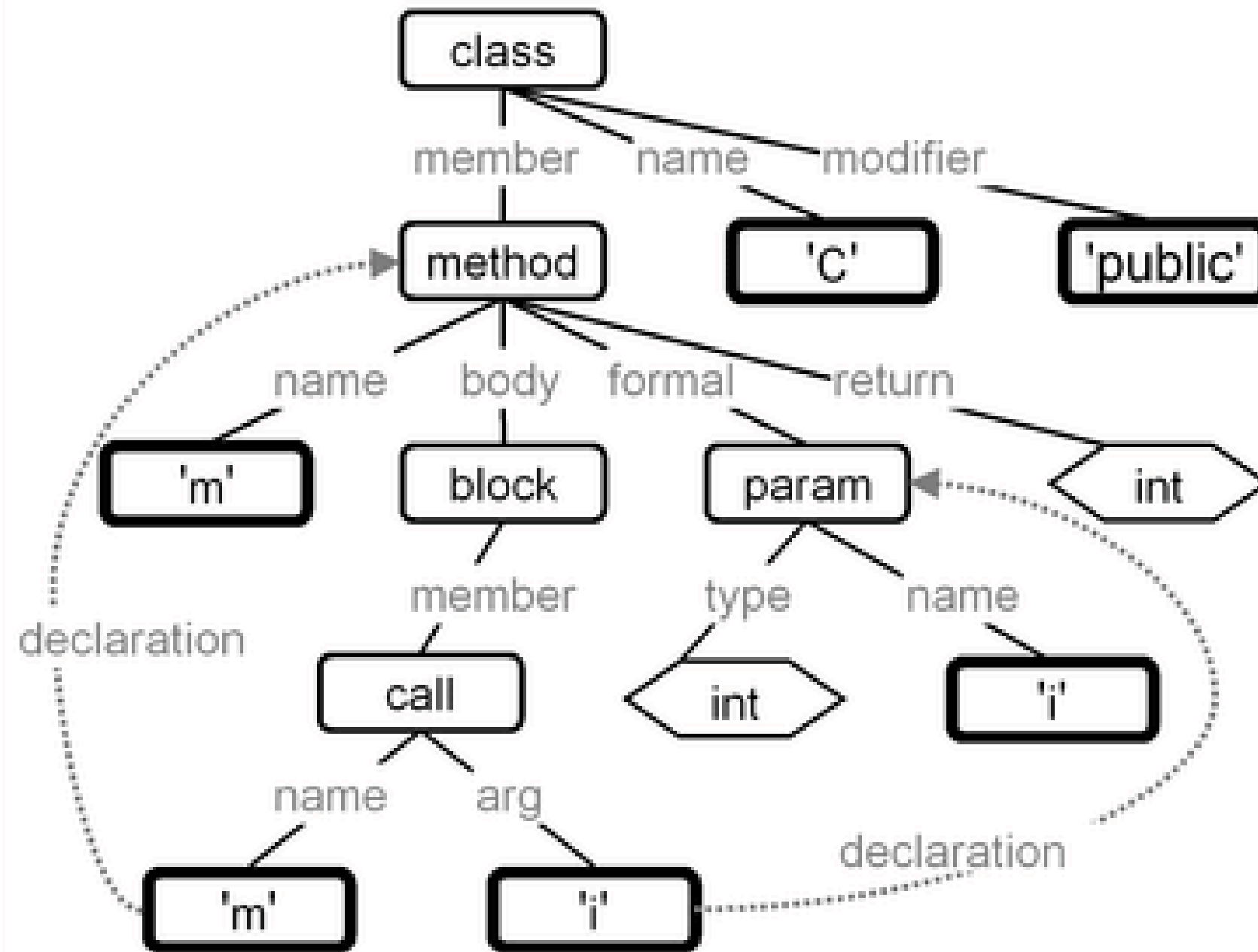
```
    int m(int i) {
```

```
        m(i);
```

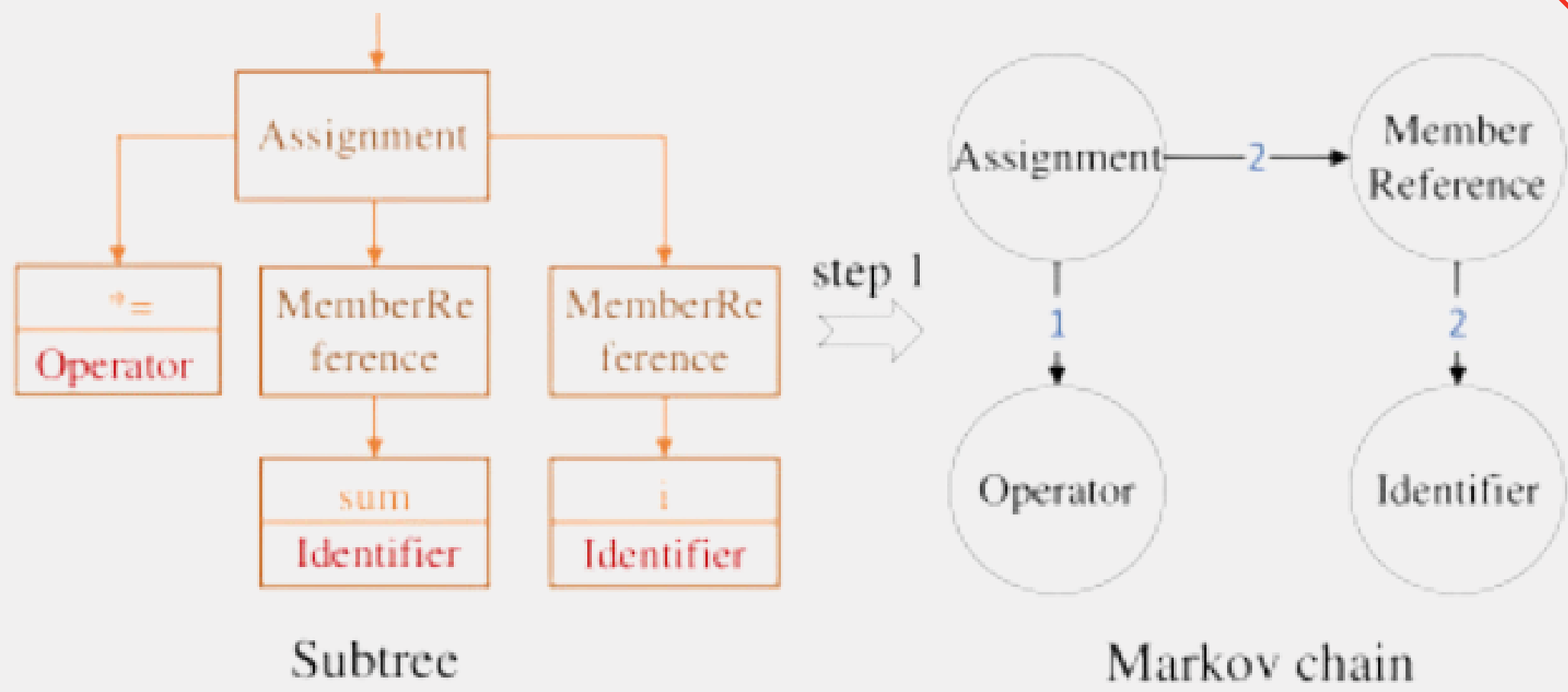
```
    }
```

```
}
```

Simplified AST



Obtener la matriz de transición



step 2

	A	I	M	...	O
A	0	0	2	...	1
I	0	0	0	...	0
M	0	2	0	...	0
⋮	⋮	⋮	⋮	⋱	⋮
O	0	0	0	0	0

step 3

	A	I	M	...	O
A	0	0	0.67	...	0.33
I	0	0	0	...	0
M	0	1	0	...	0
⋮	⋮	⋮	⋮	⋱	⋮
O	0	0	0	...	0

State Transfer Matrix

Transfer Probability Matrix

Vectorización de la matriz

all_node_types

[A, I, M, B, C, D, E, F, ..., O]

Vector con todos los tipos de nodos presentes en todo el dataset

		node types					
		A	I	M	...	O	
node types	A	0	0	0.67	...	0.33	
	I	0	0	0	...	0	
	M	0	1	0	...	0	
	⋮	⋮	⋮	⋮	⋱	⋮	
	O	0	0	0	...	0	

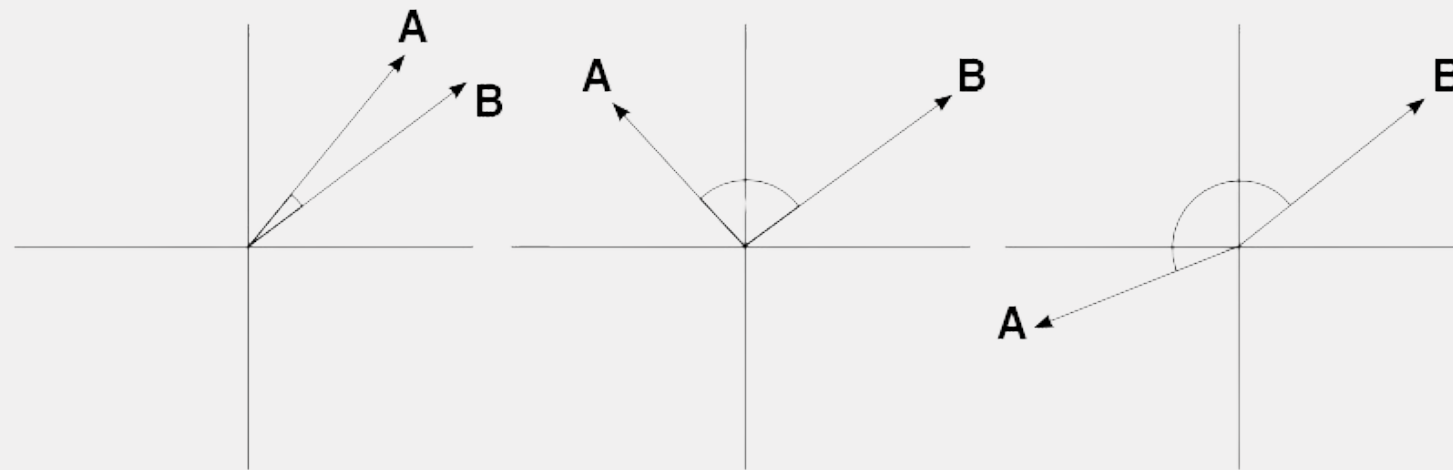
		new_matrix										
		A	I	M	B	C	D	E	F	...	O	
A	0	0	0.67	0	0	0	0	0	0	...	0.33	
I	0	0	0	0	0	0	0	0	0	...	0	
M	0	1	0	0	0	0	0	0	0	...	0	
B	0	0	0	0	0	0	0	0	0	...	0	
C	0	0	0	0	0	0	0	0	0	...	0	
D	0	0	0	0	0	0	0	0	0	...	0	
E	0	0	0	0	0	0	0	0	0	...	0	
F	0	0	0	0	0	0	0	0	0	...	0	
⋮	0	0	0	0	0	0	0	0	0	...	0	
O	0	0	0	0	0	0	0	0	0	...	0	



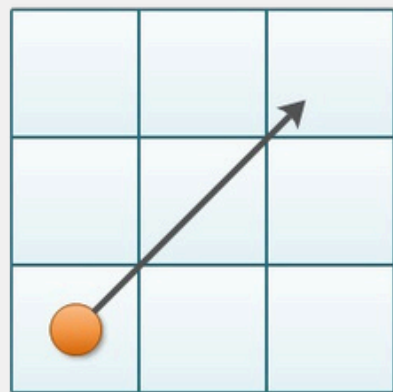
Similar

Unrelated

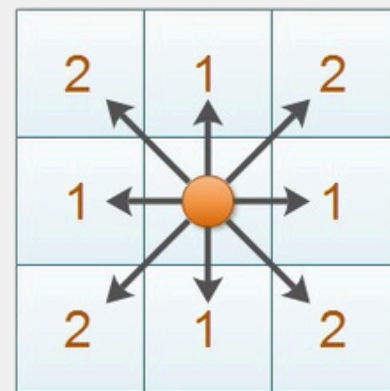
Opposite



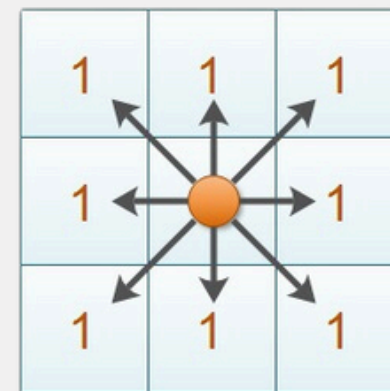
Euclidean Distance



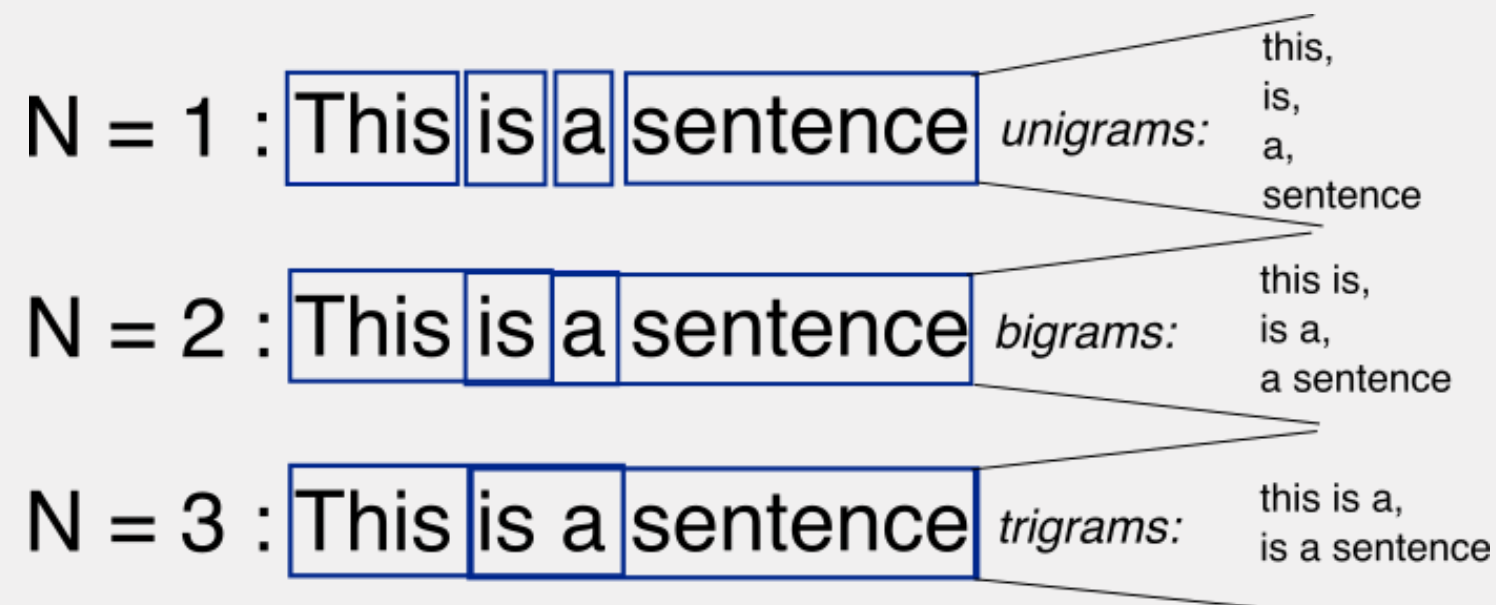
Manhattan Distance



Chebyshev Distance



$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad |x_1 - x_2| + |y_1 - y_2| \quad \max(|x_1 - x_2|, |y_1 - y_2|)$$



Cosine Similarity

vector generado de matriz de transición

Pairwise distances

vector generado de matriz de transición

euc (euclidean)

man (manhattan)

che (chebyshev)

Similitud de N-grams (4)

genera los n-grams del texto directo de los archivos y calcula cuantos hay en comun entre todos los n-grams únicos que hay

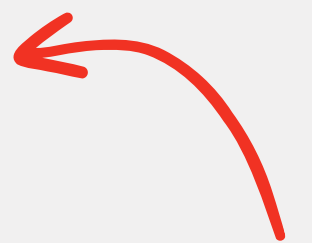
Datos al final

Features (X_{train} , X_{test})

[**vector_difference**,
csim, **euc**, **man**, **che**,
ngram_sim, **diff_len**,
diff_avg_len_line]

Target (y_{train} , y_{test})

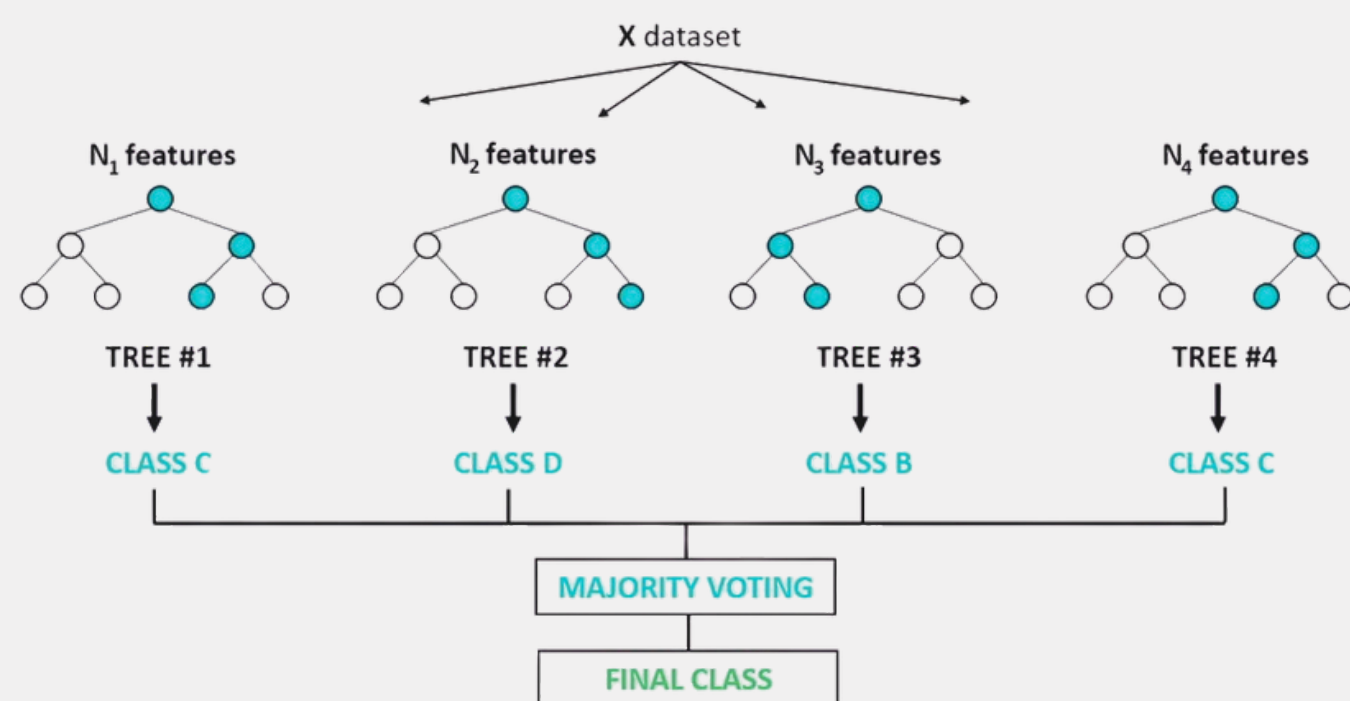
0 o **1**



plagio o no plagio, label
identificada del dataset

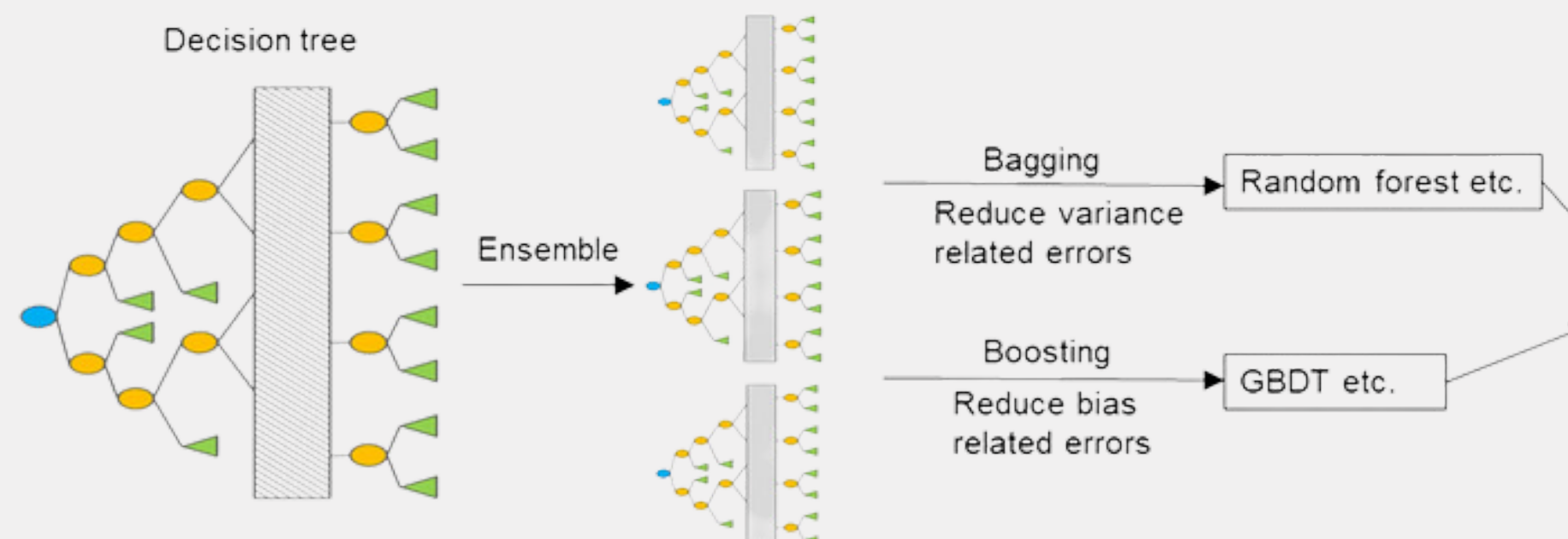
Modelos de clasificación

Para mejorar el accuracy, se optó por el uso de machine learning.
Los métodos utilizados fueron random forest (RF) y XGBoost.



Random Forest

Los hiperparámetros utilizados en el modelo RF fueron; número de árboles de 100 y un random state de 42.



XGBoost

En el modelo de XGBoost; profundidad máxima de 6, número de árboles de 100 y learning rate de 0.3.

Resultados del modelo Random Forest



Accuracy de 86%

En el dataset de prueba el modelo presenta un 86.1 por ciento de accuracy

Matriz de confusión

T/F	0	1
0	143	3
1	27	43

Reporte de clasificación

	Precision	Recall	F1
0	0.84	0.98	0.91
1	0.93	0.61	0.74

Resultados del modelo Xgboost



Accuracy de 90%

En el dataset de prueba el modelo presenta un 89.8 por ciento de accuracy

Matriz de confusión

T/F	0	1
0	139	7
1	15	55

Reporte de clasificación

	Precision	Recall	F1
0	0.90	0.95	0.93
1	0.89	0.79	0.83

Posibles mejoras al modelo



SNNs

CNNs

Probar el uso de
LSTMs

REDES NEURONALES

Analizar frecuencia de
comentarios vs código

Análisis de
espacios vs tabs

Análisis de indentación

MÁS FEATURES



Dataset más grande y
diverso

Dataset más grande y
diverso

Quitar comentarios,
espacios e imports

DATASET

The background features four abstract geometric patterns in the corners. Top-left: A series of parallel black diagonal lines and a light blue curved line. Top-right: A cluster of black and red semi-circles. Bottom-left: A cluster of red and black semi-circles. Bottom-right: A light blue curved line and a series of parallel black diagonal lines.

Gracias por su
atención
¿Preguntas?

Referencias

- Yueming Wu, Siyue Feng, Deqing Zou, and Hai Jin. 2022. Detecting Semantic Code Clones by Building AST-based Markov Chains Model. In 37th IEEE/ACM International Conference on Automated Software Engineering (ASE'22), October 10–14, 2022, Rochester, MI, USA. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3551349.3560426>
- Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam and B. Maqbool, "A Systematic Review on Code Clone Detection," in IEEE Access, vol. 7, pp. 86121-86144, 2019, doi: 10.1109/ACCESS.2019.2918202. <https://ieeexplore.ieee.org/abstract/document/8719895>
- N. Awale, M. Pandey, A. Dulal y B. Timsina, "Plagiarism Detection in Programming Assignments using Machine Learning", Sept. 2020, vol. 2, n.º 3, pp. 177–184, julio de 2020. Accedido el 4 de abril de 2024. [En línea]. Disponible: <https://doi.org/10.36548/jaicn.2020.3.005>
- XGBoost Parameters — xgboost 2.0.3 documentation. (s. f.-b). <https://xgboost.readthedocs.io/en/stable/parameter.html#general-parameters>