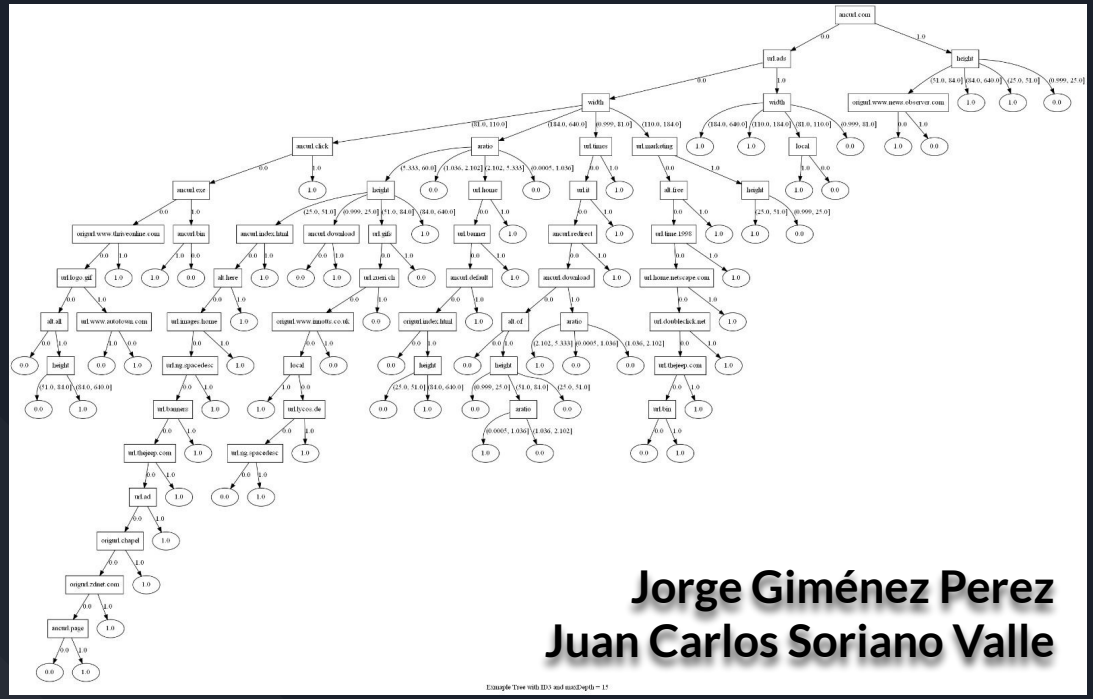
Práctica 2: Decision Trees

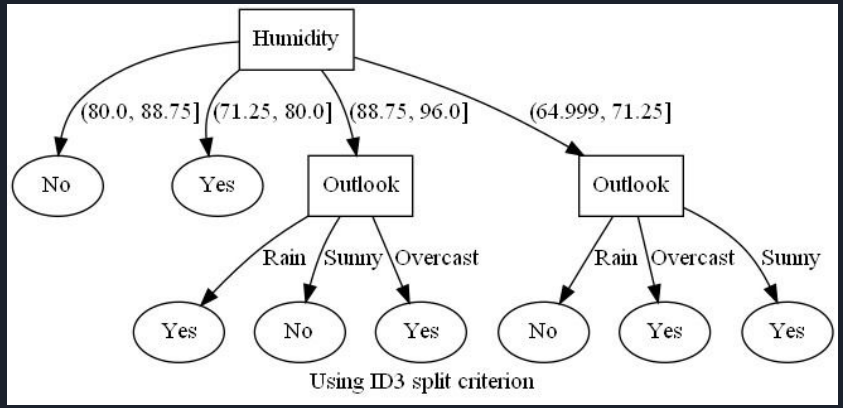


Jorge Giménez Perez
Juan Carlos Soriano Valle

17-11-20
Coneixement, Raonament i Incertesa
Grup: 8:30_3

Introducción del problema

- Generación de modelos de Árboles de Decisión para predecir si una instancia pertenece a una clase o una otra.
- Implementación de algoritmos de separación (ID3, C4.5, Gini) y algoritmos de validación (Kfold)



Dataset

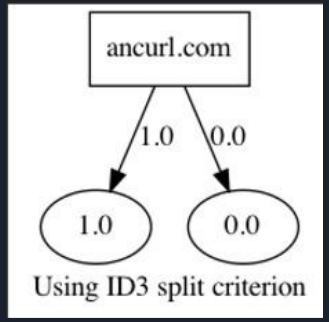
- Estudio sobre posibles anuncios en páginas web
- Atributos como las imagenes y su dimensión, palabras dentro la URL, etc.
- Gran número de filas (3279) y columnas (1558)

Data Set Characteristics:	Multivariate	**Number of Instances:**	3279	**Area:**	Computer
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	1558	**Date Donated**	1998-07-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	363522

Procesamiento de Datos

- ProcessDatasetFiles: Programa que se encarga de procesar los datos crudos del Dataset
  - Transformación de los valores “unknown”
  - Generación del Dataframe arreglado (Valores continuos a quartiles)
  - Kfold para la posterior validación (k=5)

Módulos del programa



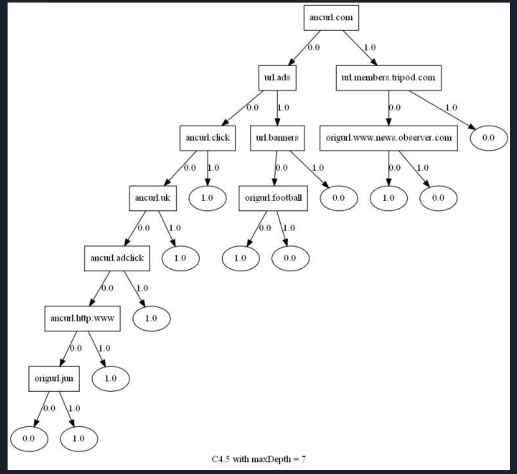
- Clase DecisionTreeNode

  Atributos de cada Nodo: Decision Tree al que pertenece, SplittingValue utilizado, raiz o hoja, ect.

  En esta clase se generan todos los nodos del Árbol de Decisión dependiendo del método de separación.

  Métodos toJSON y fromDict que nos ayudan a importar y exportar el estado actual de cada nodo para tener una rápida lectura de los Árboles y no generarlos más veces de las necesarias

Módulos del programa



- Clase DecisionTree

  Atributos del Árbol: Variable que indica el tipo con el que se representa la variable objetivo, columna de la variable objetivo, método de separación utilizado, etc.

  Método generador y predictor del Árbol que devuelve matriz con las predicciones.

  Visualizador de Árboles renderizados en archivos .PNG

  Métodos para importar y exportar en archivos .JSON recursivamente por los nodos

Módulos del programa

- Clase SplittingAlgorithm

  Algoritmos base que utilizan los 3 métodos

  Cálculo de la Entropia para cada paso del Árbol

  Algoritmos de separación (ID3, C4.5 y Gini)

Módulos del programa

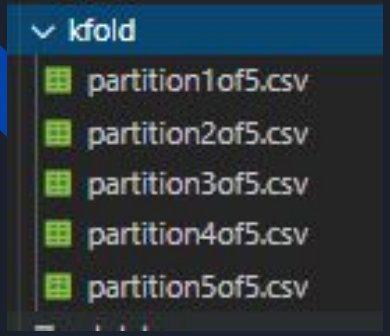
- Funciones Misceláneas:
  - DiscretizeDataframe: Separar atributos continueos en tantos cuartiles como indiquemos
  - DeleteRowsWithValues: Eliminar filas del Dataframe
  - TrainTestSplit: Separar Dataframe en datos de Train y Test
  - GetKfoldSubsets: Generación de particiones para el Kfold Validation

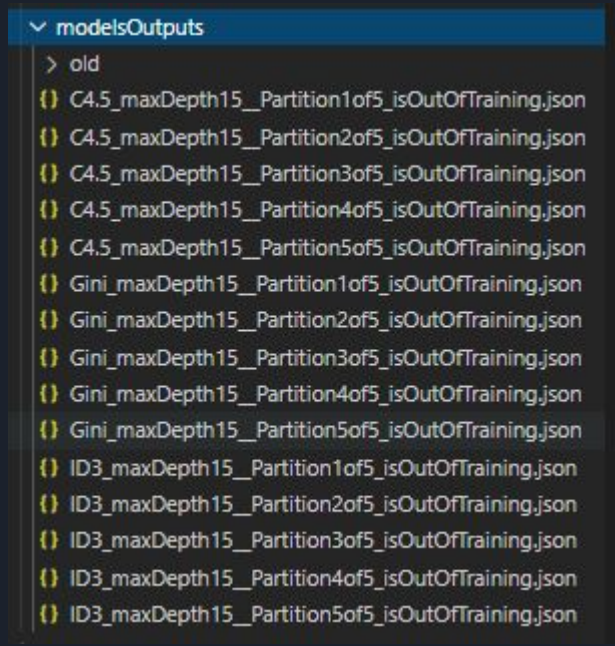
```python
def getKfoldSubsets(dataframe, k):
    """
    Esta función sirve para generar las particiones de kfold deseadas.
    dataframe: Dataframe de pandas que contiene todos los registros.
    k: Número de particiones deseadas.
    Devuelve k dataframes.
    """
    nInstances = dataframe.shape[0]
    print(nInstances)
    partitionInstances = int(round(nInstances/k))
    if partitionInstances*k > nInstances: partitionInstances -= 1
    dataframe = dataframe.sample(n=nInstances, random_state=0).reset_index(drop=True)
    subsets = []
    for i in range(k):
        subset = dataframe.iloc[partitionInstances*i:partitionInstances*i+partitionInstances, :]
        subsets.append(subset)
    return subsets
```

Entrenamiento de los modelos

Para entrenar los modelos y poder evaluarlos posteriormente utilizamos k-fold como técnica de cross validation:

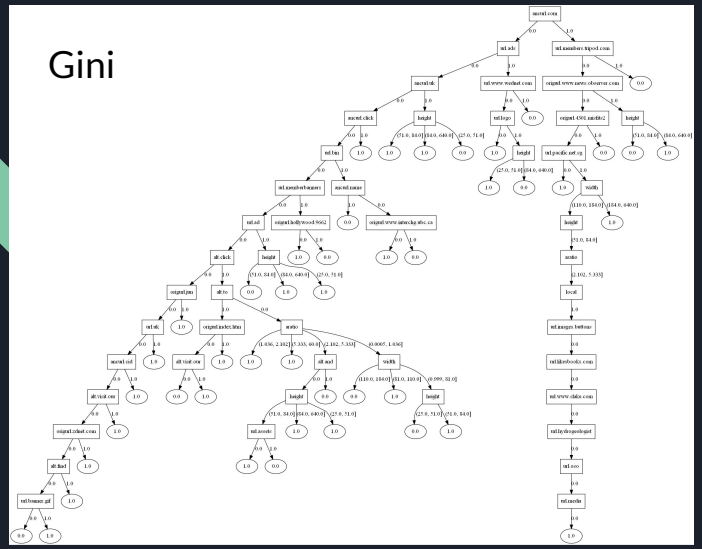
- Partimos el dataset en 5 particiones.
- Entrenamos 5 modelos de cada tipo (5 para ‘ID3’, 5 para ‘C4.5’, y 5 para ‘Gini’) con todas las particiones de k-fold menos 1.
- Estos modelos tienen la profundidad máxima limitada a 15 nodos para reducir el coste computacional.
- Evaluamos cada uno de los modelos contra aquella partición del dataset que no se había usado para entrenarlo.



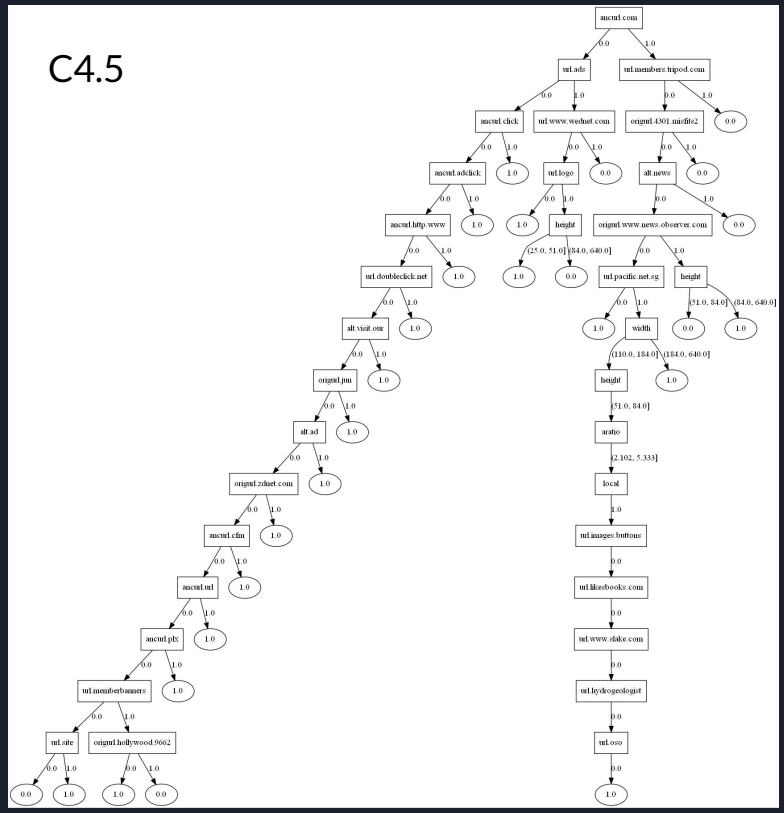



```python
MAX_DEPTH = 15

kfoldPartitions = []
for i in range (1, 6):
    kfoldPartitions.append(pd.read_csv(f'./data/kfold/partition{i}of5.csv'))

def generateModelAndSave(tree, fileName):
    tree.generate()
    tree.saveToFile(fileName)

for kIdx in range(1,6):
    print(f"PROGRESS ===> Current K index {kIdx}/5")
    trainPartitions = []
    for partitionIdx in range(5):
        if partitionIdx+1 != kIdx: trainPartitions.append(kfoldPartitions[partitionIdx])
    trainData = pd.concat(trainPartitions)

    tree1 = DecisionTree(trainData, 'class', 1.0, 0.0, 'ID3', maxDepth=MAX_DEPTH)
    tree2 = DecisionTree(trainData, 'class', 1.0, 0.0, 'Gini', maxDepth=MAX_DEPTH)
    tree3 = DecisionTree(trainData, 'class', 1.0, 0.0, 'C4.5', maxDepth=MAX_DEPTH)
    t1 = threading.Thread(target=generateModelAndSave, args=(tree1, f'ID3_maxDepth{MAX_DEPTH}__Partition{kIdx}of5_isOutOfTraining'))
    t2 = threading.Thread(target=generateModelAndSave, args=(tree2, f'Gini_maxDepth{MAX_DEPTH}__Partition{kIdx}of5_isOutOfTraining'))
    t3 = threading.Thread(target=generateModelAndSave, args=(tree3, f'C4.5_maxDepth{MAX_DEPTH}__Partition{kIdx}of5_isOutOfTraining'))
    t1.start()
    t2.start()
    t3.start()
    t1.join()
    t2.join()
    t3.join()
    print('\n')
```

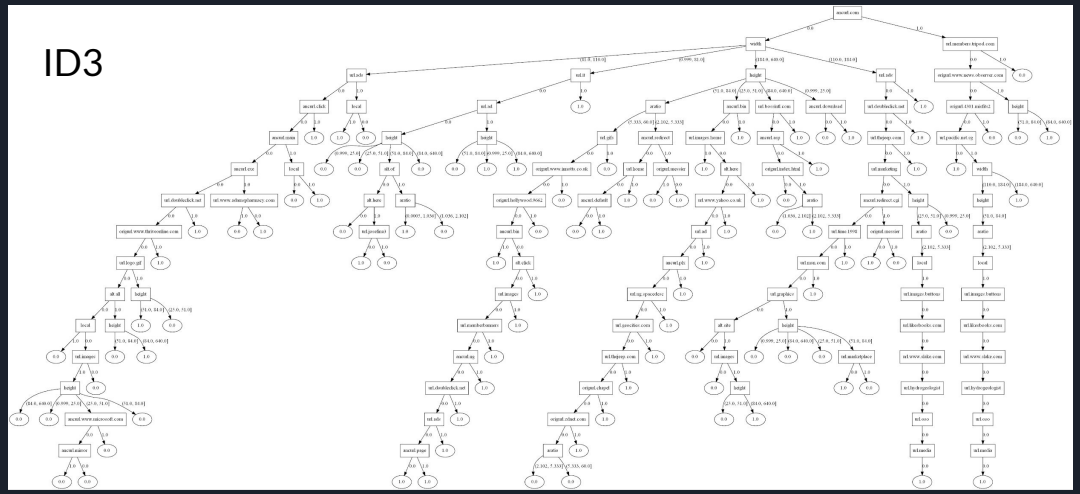
Gini



C4.5

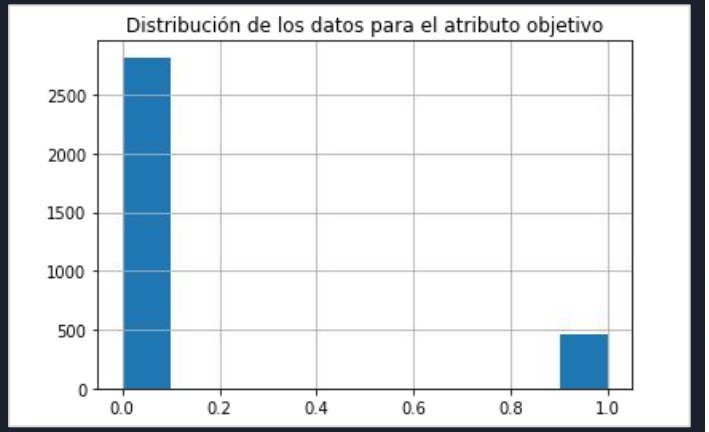


ID3



Evaluación de los modelos

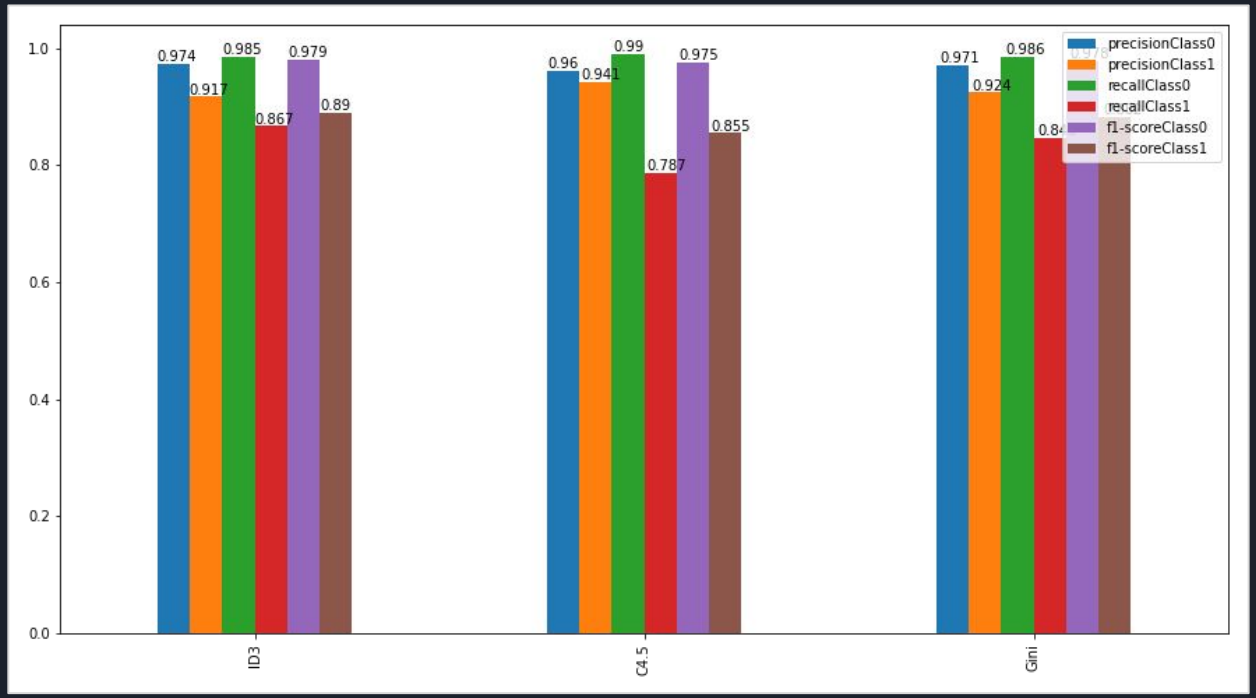
- Dado el desbalance de los datos, no nos aportará valor la accuracy, optamos por usar como métricas: precision, recall y f1-score.
- En especial nos interesan las métricas para la clase 1.0-‘ad’ ya que es más relevante poder identificar si un registro es anunico.



	height	width	aratio	local	url.images.buttons	url.likesbooks.com	url.www.slake.com	url.hydrogeologist	url.oso	url.media	...	caption.home	capti
0	125	125	1.0	1	0	0	0	0	0	0	...	0	
1	57	468	8.2105	1	0	0	0	0	0	0	...	0	
2	33	230	6.9696	1	0	0	0	0	0	0	...	0	
3	60	468	7.8	1	0	0	0	0	0	0	...	0	
4	60	468	7.8	1	0	0	0	0	0	0	...	0	
5	60	468	7.8	1	0	0	0	0	0	0	...	0	
6	59	460	7.7966	1	0	0	0	0	0	0	...	0	
7	60	234	3.9	1	0	0	0	0	0	0	...	0	
8	60	468	7.8	1	0	0	0	0	0	0	...	0	
9	60	468	7.8	1	0	0	0	0	0	0	...	0	
10	unknown	unknown	unknown	1	0	0	0	0	0	0	...	0	
11	90	52	0.5777	1	0	0	0	0	0	0	...	0	
12	90	60	0.6666	1	0	0	0	0	0	0	...	0	
13	90	60	0.6666	1	0	0	0	0	0	0	...	0	
14	33	230	6.9696	1	0	0	0	0	0	0	...	0	

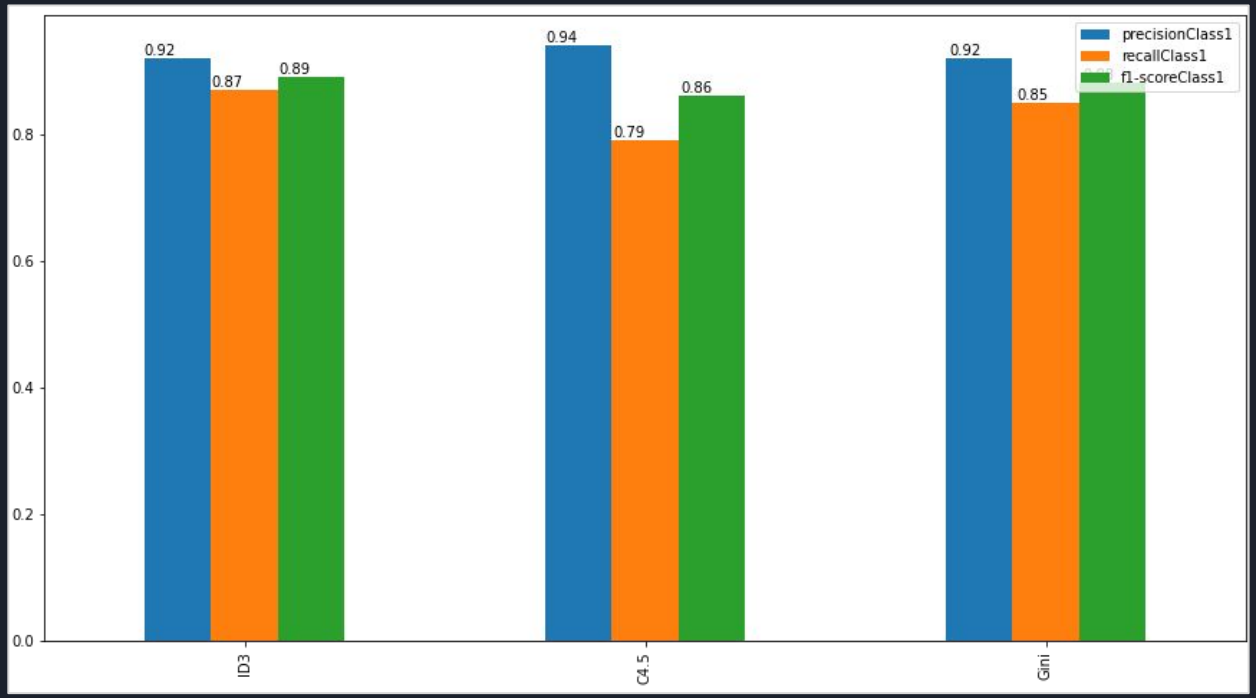
15 rows × 1559 columns

Evaluación de los modelos



	precisionClass0	precisionClass1	recallClass0	recallClass1	f1-scoreClass0	f1-scoreClass1
ID3	0.974	0.917	0.985	0.867	0.979	0.890
C4.5	0.960	0.941	0.990	0.787	0.975	0.855
Gini	0.971	0.924	0.986	0.846	0.978	0.882

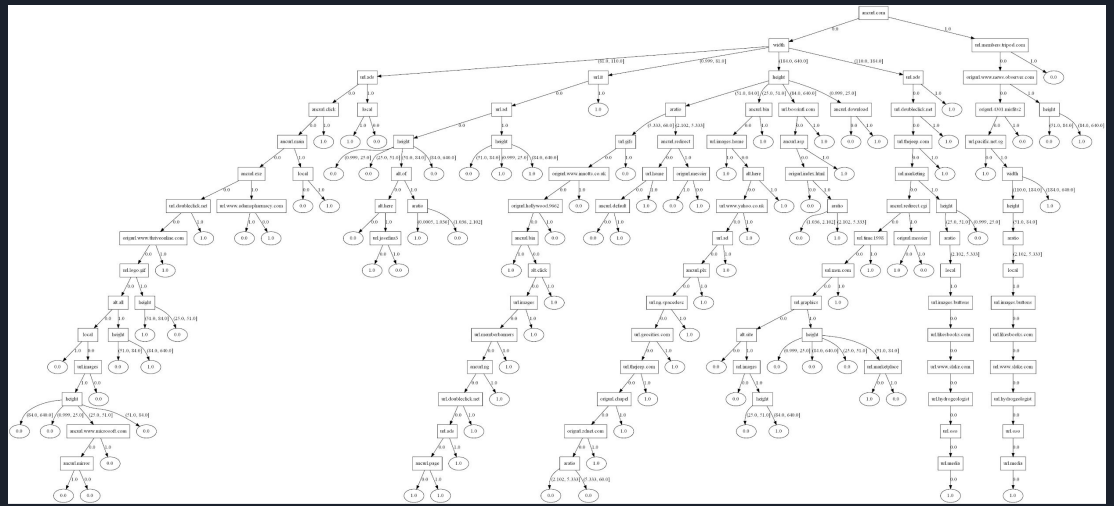
Evaluación de los modelos



	precisionClass1	recallClass1	f1-scoreClass1
ID3	0.92	0.87	0.89
C4.5	0.94	0.79	0.86
Gini	0.92	0.85	0.88

Evaluación de los modelos

ID3 nos arroja unos valores de precisión iguales a los de Gini y únicamente un poco por debajo de C4.5. Además, nos otorga los mejores valores para recall. Siendo además muy equilibrado obtiene como es de esperar, el mejor valor para la métrica f1-score tanto en la clase 0 como en la clase 1 (a la que le damos la mayor importancia). Es por esto que si tuviéramos que usar alguno de nuestros modelos entrenados para clasificar muestras de este dataset, nos inclinaríamos por usar ID3 como criterio de partición en nuestros árboles.



Problemas y conclusiones

- Dataset con demasiadas columnas.
- Tiempo de cómputo muy alto.
- Nos hubiera gustado hacer más, pero no hemos tenido el tiempo suficiente.
- Suponemos que con un uso más eficiente de la librería Numpy, se podría haber optimizado el cómputo con el alto número de columnas (atributos)

- Mejor compresión de los árboles de decisión y de los algoritmos para dividir en cada nodo.
- Implementación funcional y usable en datasets reales.
- Resultados satisfactorios desde nuestro punto de vista y satisfacción con el código desarrollado.