Universidad Autónoma de Querétaro

Facultad de Ingeniería
Maestría en Inteligencia Artificial
Machine Learning
Cecilia Gabriela Rodríguez Flores
Sheila Leyva López

Árbol de decisión: ID3

04 de noviembre del 2022

Objetivo

Desarrollar un algoritmo de árboles de decisión, mediante el lenguaje de programación de python, capaz de realizar la tarea de clasificación de enfermedad cardíaca.

Introducción

Dentro del área de la Inteligencia Artificial (IA) existe una sub-área llamada aprendizaje automático o Machine Learning (ML), cuyas herramientas permiten a un sistema aprender patrones y comportamientos de los datos en lugar de aprender mediante la programación explícita. Asimismo, existen distintos tipos de ML: aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi-supervisado y aprendizaje por refuerzo. Específicamente, hay un tipo de algoritmo de aprendizaje supervisado no paramétrico utilizado para tareas de clasificación como de regresión, llamado árboles de decisión. Básicamente, son modelos predictivos formados por reglas binarias (si/no) con las que se logra repartir las observaciones en función de sus atributos y predecir así el valor de la clase del atributo de decisión. Este método es útil para minería de datos y tareas de descubrimiento de conocimiento.

En el presente trabajo se desarrollan las funciones, en lenguaje de Python, necesarias para la predicción de clases de cualquier base de datos, sin embargo, toma como referencia la base de datos: Iris.

Marco Teórico

Es importante definir conceptos básicos implementados en este trabajo a fin brindar un mejor entendimiento del contexto en el que se trabaja.

Árboles de decisión

Los árboles de decisión son una técnica de aprendizaje automático supervisado y ha sido utilizada en múltiples ámbitos desde la inteligencia artificial hasta la economía. Esta técnica va tomando las decisiones siguiendo la estructura de un árbol. Los nodos intermedios representan soluciones y las hojas la predicción o clasificación que se está buscando [1].

Las principales ventajas de este algoritmo es que es fácil de entender y explicar a personas que no están familiarizadas con técnicas de inteligencia artificial, se adapta a cualquier tipo de datos y descubre cuales son los atributos relevantes. Por otro lado, sus desventajas son que no extrapolan bien fuera de su rango de entrenamiento y tienden al sobreajuste [1].

ID3

ID3 es el acrónimo de “Iterative Dichotomiser 3” o dicotimizador iterativo en español. Fue desarrollado por J. Ross Quinlan. La salida del algoritmo se puede presentar como como un grafo en forma de árbol cuyos componentes son:

- Nodo raíz: Es el nodo principal localizado en la parte superior. De este nodo parten ramas hacia otros nodos inferiores, de los cuales pueden salir más ramas hacia otros nodos.
- Nodos terminales: Como su nombre lo indica, son nodos donde termina el flujo y que ya no son raíz de ningún otro nodo, también puede denominarse hoja. Estos nodos terminales contienen la clasificación a la cual pertenece el objeto que ha conducido hasta dicho nodo.
- Nodos intermedios: representan preguntas con respecto al valor de uno de los atributos.

- Ramas: representan las posibles respuestas que los atributos pueden tomar.

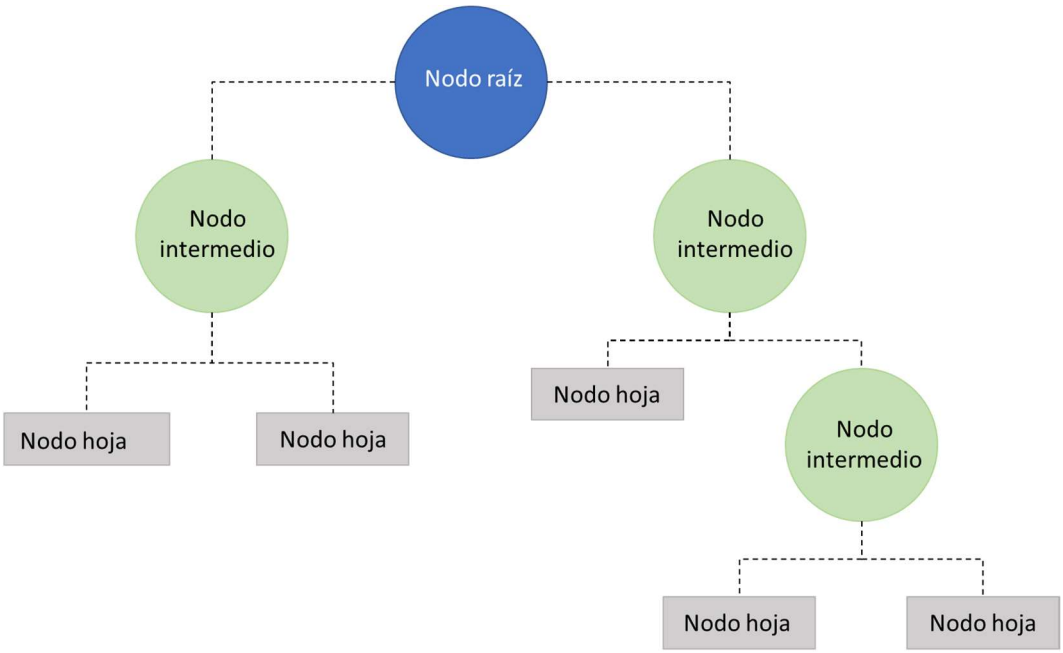

Ilustración 1. Estructura general de un árbol de decisión.

El objetivo principal del algoritmo ID3 es determinar, para un conjunto de datos, el atributo más importante, es decir, aquel que posea el mayor poder discriminatorio para dicho conjunto, el cual será denominado nodo raíz; este atributo es usado para la clasificación de la lista de objetos, basados en los valores asociados con él mismo. Después de haber hecho la primera prueba de atributo se arrojará un resultado, el cual es en sí mismo un nuevo problema de aprendizaje de árbol de decisión, con la diferencia que contará con menos ejemplos y un atributo menos, por lo que, cada atributo que se selecciona se descarta para la siguiente prueba [2].

Este proceso se realiza mediante el apoyo de dos operaciones de probabilidad, la entropía y la ganancia. La entropía es una medida de incertidumbre y es usado para ayudar a decidir qué atributo debe ser el siguiente en seleccionarse. Se calcula de la siguiente forma:

$$\text{Entropia}(S) = \sum_{i=1}^{n} -p_i \log_2 p_i \qquad (1)$$

Mientras que la ganancia de información es una medida de discriminación e indica el siguiente atributo que debe ser seleccionado y se calcula de la siguiente forma:

$$\text{Gan Inf}(S, A) = \text{Entropia}(S) - \sum_{v \in V(A)} \frac{|Sv|}{|S|} \text{Entropia}(Sv) \quad (2)$$

Prueba Cobertura

La prueba de cobertura es una de las pruebas más utilizadas en la evaluación del desempeño de los árboles de decisión, dado que, se pretende conocer si se recorren todos los posibles caminos pasando por todos los nodos intermedios y finales. Idealmente se espera una cobertura igual o cercana al 100%[3].

Etapas en la implementación de modelos

Es importante definir conceptos básicos del aprendizaje máquina que permitan entender el contexto en el que se trabaja, como son las etapas fundamentales para la implementación de técnicas de aprendizaje máquina: Preparación de datos, la cual es una etapa fundamental en cualquier proyecto, debido a que por medio de ella se pueden inicializar correctamente los datos para su posterior procesamiento y análisis; Analizar, en esta sección se utilizaran (que dato se van a utilizar) o dividirá los datos ya preparados para el siguiente paso, para este trabajo se optó por crear subconjuntos de datos, en los cuales el 80% de los datos se considera para el entrenamiento, mientras que un 20% corresponde a pruebas del modelo de clasificación.

Métricas

Exactitud

La exactitud es la relación entre los simples correctamente clasificados y el número total de simples en el conjunto de datos de evaluación. Esta métrica es conocida por ser engañosa en el caso de diferentes proporciones de clases, ya que asignar simplemente todos los simples a la clase predominante es una forma fácil de lograr una alta precisión [3].

$$ACC = \frac{\#\ correctly\ classified\ samples}{\#\ all\ samples} = \frac{TP+TN}{TP+FP+TN+FN} \quad (3)$$

Precisión

La precisión es la capacidad de un modelo para identificar sólo objetos relevantes, es decir, es el porcentaje de predicciones positivas correctas entre todas las verdades básicas dadas [3].

$$P_r = \frac{\sum_{n=1}^{S} TP_n}{\sum_{n=1}^{S} TP_n + \sum_{n=1}^{N-S} FP_n} = \frac{\sum_{n=1}^{S} TP_n}{all\ detection} \quad (4)$$

Materiales y métodos

Herramientas utilizadas

- Google Collaboratory
- Conjunto de datos: *Indicadores personales clave de enfermedad cardíaca*
- AMD Ryzen 9 5900HS with Radeon Graphics 3.30 GHz
- RAM 16.0 GB
- 64-bit operating system, x64-based processor

Conjunto de datos

En este trabajo, se utilizará la base de datos Iris, la cual contiene tres especies de Iris (Virginica, Versicolor y Setosa),en donde cada una de ellas contiene 50 muestras. Cada una de las muestras fue medida de acuerdo con cuatro características: ancho y largo de los pétalos y sépalos.

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

Ilustración 2. Visualización de los atributos e instancias del conjunto de datos Iris.

Así bien, este repositorio suele utilizarse ampliamente en ejemplos de minería de datos, clasificación y agrupamiento, por lo cual, es posible encontrarlo en el repositorio de aprendizaje automático UCI Machine Learning[4], Kaggle[5], como también de Scikit Learn[6].

En la siguiente figura se muestran las tres especies de Iris, mencionadas con anterioridad a fin de que el lector tenga un conocimiento previo sobre la apariencia de las clases a tratar en este trabajo.

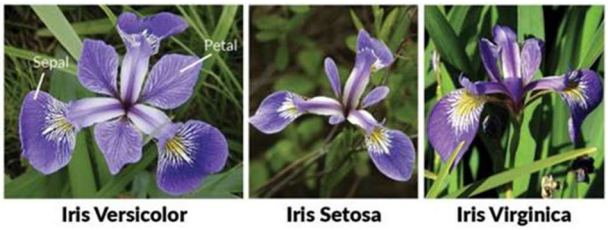

Ilustración 3. Especies de la flor Iris.

Métodos

Para la realización de esta tarea, fue necesario importar las librerías de Pandas, numpy, etc., así como el montaje de Google Drive en el entorno de ejecución a fin de poder hacer uso de la base de datos propuesta.

Posteriormente, se optó por crear diversas funciones a fin de distribuir las tareas de una manera más eficiente como se muestra a continuación:

- Función contar_clases: obtiene el número total de valores en cada una de las clases inmersas en la base de datos en cuestión. Esta función requiere como datos de entrada el atributo de decisión y las clases.

```
def contar_clases(atributo_decision, clases):
    cont_clases = np.zeros((len(clases)))
    for n, c in enumerate(clases):
        cont_clases[n] = int(np.sum(atributo_decision == c))
    return cont_clases
```

Ilustración 4. Función propuesta para la obtención del total de cada clase.

- Función calcular_entropia: realiza el cálculo de la entropía, a fin de determinar atributo debe ser el siguiente en seleccionarse.

```
def calcular_entropia(cont_clases):
    entropia_decision = 0
    total = int(sum(cont_clases))
    for d in cont_clases:
        if d == 0: entropia_decision += 0
        else: entropia_decision -= (d/total)*np.log2(d/total)
    return entropia_decision
```

Ilustración 5. Función propuesta para la obtención de la entropía.

- Función ganancia_atributo: se enfoca en la obtención de la ganancia de cada atributo. Esta función requiere de como entradas los datos, etiquetas, número total de cada clase, valor de entropía y las clases en cuestión al conjunto de datos en cuestión.

```
def ganancia_atributo(x, y, cont_clases, entropia, clases):
    Total = int(sum(cont_clases))
    ganancia = {}
    for a in x.columns:
        ganancia[a] = 0
        observaciones = x[a].unique()
        for o in observaciones:
            y_a_o = y.loc[x[a] == o]
            cont_clases_y_a_o = contar_clases(y_a_o, clases)
            entropia_a_o = calcular_entropia(cont_clases_y_a_o)
            total = int(sum(cont_clases_y_a_o))
            ganancia[a] += (total/Total) * entropia_a_o
        ganancia[a] = entropia - ganancia[a]
    return ganancia
```

Ilustración 6. Función propuesta para la obtención de la ganancia.

- Función nodo_raiz: se enfoca en seleccionar el nodo raíz para la realización del árbol de decisión. Esta función requiere de como entrada únicamente las ganancias de los atributos.

```
def nodo_raiz(ganancia):
    raiz = max(ganancia.items(), key=operator.itemgetter(1))[0]
    entropia_r = ganancia[raiz]
    return raiz, entropia_r
```

Ilustración 7. Función propuesta para la obtención del nodo raíz.

- Función nodos_intermedios: se enfoca en obtener los nodos intermedios para la realización del árbol de decisión. Esta función requiere como entradas los datos, las etiquetas, nodo raíz y las observaciones.

```
def nodos_intermedios(x, y, atributo, obs):
    subx = []
    suby = []
    for o in obs:
        pos_coinciden = x[atributo] == o
        subx.append(x.loc[pos_coinciden])
        suby.append(y.loc[pos_coinciden])
    return subx, suby
```

Ilustración 8. Función propuesta para la obtención del nodo intermedios.

- Función rama_final: se enfoca en obtener la rama final para la realización del árbol de decisión. Esta función requiere como entradas las posiciones de cada elemento, así como su correspondiente clase y las clases totales en cuestión.

```
def rama_final(y_fr, obs):
    hoja = 0
    clase_fr = 0
    n = len(y_fr)
    for o in obs:
        coincidencias = int(np.sum(y_fr == o))
        if coincidencias == n:
            hoja = 1
            clase_fr = o
    return hoja, clase_fr
```

Ilustración 9 9.Función propuesta para la obtención del nodo final.

- Función arbol_id3: se encarga en llamar a cada una de las funciones anteriormente presentadas en este trabajo, a fin de obtener un árbol de decisión. Esta función requiere como entradas los datos, las etiquetas, las clases totales en cuestión y la creación de un diccionario para el árbol.

```
def arbol_id3(x, y, clases, arbol):
    conteo_clases = contar_clases(y, clases)
    entropia_decision = calcular_entropia(conteo_clases)
    ganancia = ganancia_atributo(x, y, conteo_clases, entropia_decision, clases)
    nodo_raiz_, entropia_raiz = nodo_raiz(ganancia)
    arbol[nodo_raiz_] = {}
    ramas = x[nodo_raiz_].unique()
    sub_x, sub_y = nodos_intermedios(x, y, nodo_raiz_, ramas)
    print(sub_y)
    print(clases)
    print('')
    for r, sx, sy in zip(ramas, sub_x, sub_y):
        arbol[nodo_raiz_][r] = {}
        hoja, clase_fr = rama_final(sy, clases)
        if hoja == 1:
            arbol[nodo_raiz_][r] = clase_fr
        else:
            arbol_id3(sx, sy, clases, arbol[nodo_raiz_][r])
    return arbol
```

Ilustración 10. Función propuesta para la obtención del árbol de decisión.

- Función division_80_20: realiza la separación del conjunto de datos de acuerdo al porcentaje de 80% para el conjunto de entrenamiento y 20% para el conjunto de prueba. Esta función requiere como entradas los datos y las etiquetas.

```
def division_80_20(datos, etiquetas):
  n = len(datos)
  indices = np.arange(n)
  np.random.shuffle(indices)
  lim = int(n * 0.80) + 1
  x_train = datos.loc[indices[0:lim]]
  x_test = datos.loc[indices[lim:]]
  y_train = etiquetas.loc[indices[0:lim]]
  y_test = etiquetas.loc[indices[lim:]]
  return x_train, x_test, y_train, y_test
```

Ilustración 11. Función propuesta para la división del conjuntos de datos.

- Función clasificar_id3: se encarga de realizar la prueba de cobertura de acuerdo al árbol de decisión.

```
def clasificar_id3(x):
  indices = x.index
  clasificaciones = []
  for i in indices:
    clasifica = x.loc[i]
    resultado_i = comprobacion(clasifica)
    clasificaciones.append(resultado_i)
  return clasificaciones
```

Ilustración 12. Función propuesta para la obtención de la métrica de exactitud.

Además, una vez realizado el entrenamiento y prueba de los datos, es necesario visualizar el comportamiento del modelo ID3 por medio de las métricas de precisión y exactitud, las cuales se muestran en la Ilustración 13 y 14.

```python
def exactitud(y_calculada, y_real):
  coincidencias = np.equal(y_calculada, y_real)
  total_coincidencias = np.sum(coincidencias)
  porcentaje = (total_coincidencias / len(y_calculada))
  return porcentaje
```

Ilustración 13. Función propuesta para la obtención de la métrica de exactitud.

```python
def precision(y_class, y_real):
    TP = FP = 0
    for c, r in zip(y_class, y_real):
        if r == 'Setosa':
            if ((r == c) and r == 'Setosa'): TP += 1
            elif ((r != c) and ( r == 'Virginica' or r == 'Versicolor')): FP += 1
        elif r == 'Virginica':
            if ((r == c) and r == 'Virginica'): TP += 1
            elif ((r != c) and ( r == 'Setosa' or r == 'Versicolor')): FP += 1
        elif r == 'Versicolor':
            if ((r == c) and r == 'Versicolor'): TP += 1
            elif ((r != c) and ( r == 'Virginica' or r == 'Setosa')): FP += 1
    print('TP:',TP)
    print('FP:',FP)
    Pr = TP / (TP + FP)
    return Pr
```

Ilustración 1410. Función propuesta para la obtención de la métrica de precisión.

Diagrama de metodología

Para el análisis de cualquier base de datos se deben seguir los siguientes pasos:

1. Cargar base de datos.
2. Definir el conjunto de decisión.
3. Separar el atributo de decisión del resto de atributos.
4. Dividir los conjuntos de entrenamiento y prueba del modelo ID3, asignando el 80% de ellos para el entrenamiento y el 20% restante a la etapa de prueba.
5. Calcular la entropía total de la base de datos acorde a la columna decisión.
6. Conocer las observaciones que contiene cada atributo.
7. Calcular la entropía de cada observación con respecto al atributo en cuestión
8. Calcular la probabilidad de la entropía de cada atributo en cuestión.
9. Calcular la ganancia por medio de la entropía total y la probabilidad obtenida del paso anterior.
10. Comparar las ganancias obtenidas en el paso anterior a fin de determinar el nodo raíz
11. Se calcula de la misma manera la ganancia del resto de los atributos, de tal forma, que se van asignando y generando nodos intermedios y nodos hoja.
12. Una vez diseñado el árbol de decisión, se entrena el modelo.

13. Por último, se calculan las métricas de exactitud y precisión para conocer el desempeño del árbol de decisión.

A continuación, se muestra el diagrama de flujo que representa el proceso de desarrollo del algoritmo de árbol de decisión ID3, el cual fue implementado en un programa basado en lenguaje Python.

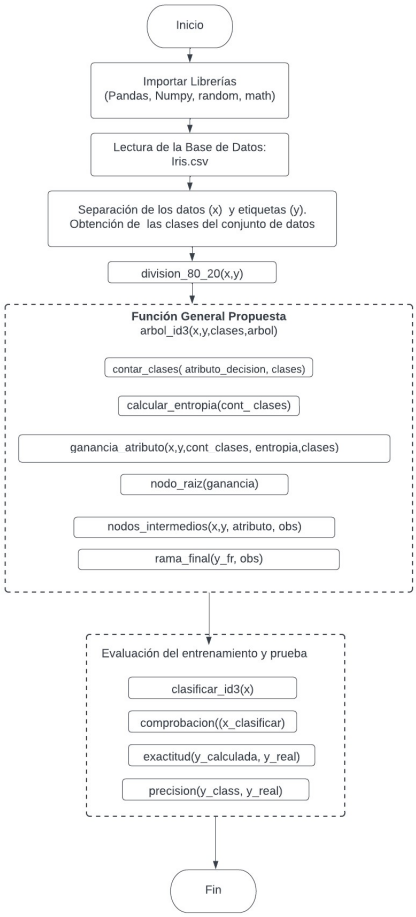

Ilustración 15. Diagrama de flujo para el análisis del conjunto de datos.

Resultados y discusión

A partir de las funciones desarrolladas, en donde se calcularon tanto las entropías como ganancias correspondientes, se obtuvo como resultado un árbol de decisión capaz de clasificar el conjunto de datos de Iris. En la Ilustración 16 se puede observar la arquitectura del árbol de decisión obtenido, su nodo raíz, nodos intermedios y hojas.

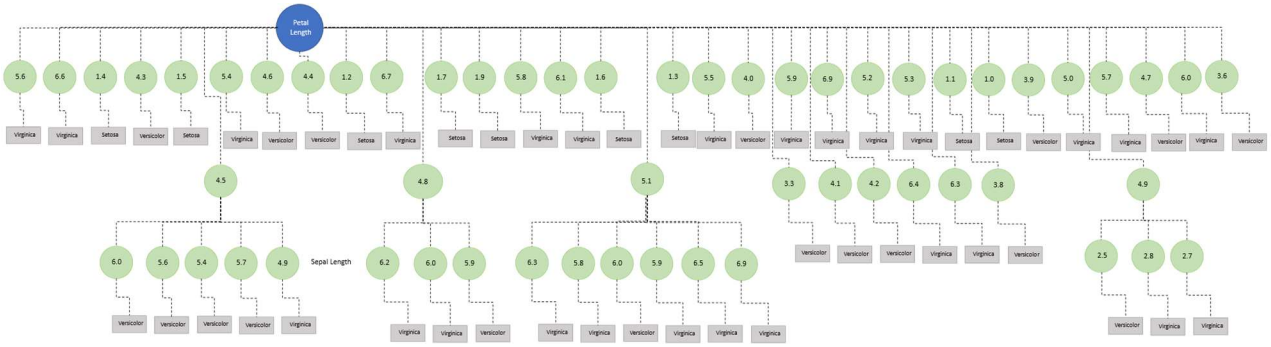

Ilustración 16. Estructura del árbol de decisión para la clasificación del conjunto de datos Iris.

Así bien, se puede observar el nodo raíz correspondiente a petal.length, así como los nodos intermedios asignados a sepal.length y sepal.width; terminando en los nodos correspondientes al atributo 'variety' de la flor.

En la Tabla 1 se presentan los valores del desempeño del modelo de árbol de decisión: ID3. Resultado de haber utilizado el popular conjunto de datos: Iris. Como era de esperarse el valor de exactitud durante la etapa de prueba es menor que durante el entrenamiento del modelo.

Tabla 1. Métricas de evaluación del rendimiento de ID3.

	Exactitud	Precisión
Entrenamiento	0.9917	1.0000
Prueba	0.6896	1.0000

Durante el entrenamiento el algoritmo obtuvo excelentes resultados. Sin embargo, el valor de la exactitud durante la etapa de prueba es menor en 0.31 con respecto al entrenamiento, este valor indica realmente cual es el desempeño del modelo si se pone a prueba con valores distintos a

los entrenados. Asimismo, la precisión tanto en entrenamiento como en prueba es de 1.0000, lo cual indica un alto grado de cercanía de los valores predichos en un punto.

De acuerdo a lo anterior, se puedo observar que al realizar la ejecución de las líneas de código relacionadas con la separación de los datos de entrenamiento y prueba, así como las posteriores líneas de código, los elementos del árbol se modificaban de igual manera.

Conclusiones

En este programa generalizado inicialmente se generaron funciones que permitieran conocer el comportamiento de los datos mediante la implementación del árbol de decisión, así como funciones que permiten obtener métricas estadísticas de los datos.

Afortunadamente, la aplicación de un ID3 a la base de datos Iris no resultó en un árbol con cientos de ramas, sin embargo, este algoritmo por su naturaleza recursiva suele volverse complejo cuando se aumenta la cantidad de instancias, observaciones y atributos, puesto que esto ocasiona que aumente el número de ramas y profundidad del árbol lo que a su vez requiere mayor tiempo y costo computacional para el entrenamiento. Además, se tuvo la ventaja de contar con una base de datos con clases balanceadas, sin datos faltantes o un gran número de outliers o algún otro factor que pudiera alterar los valores de entropía o ganancia del conjunto de datos.

Referencias

[1] Martinez, J. (2020, September 19). Árboles de Decisión con ejemplos en Python. IArtificial.net. 2022, from https://www.iartificial.net/arboles-de-decision-con-ejemplos-en-python/

[2] Rahman, T. (2021, March 27). Step by Step Decision Tree: ID3 Algorithm From Scratch in Python [No Fancy Library]. Medium.2022, from https://medium.com/geekculture/step-by-step-decision-tree-id3-algorithm-from-scratch-in-python-no-fancy-library-4822bbfdd88f

[3] M. Antonio and A. Fernández, *Inteligencia artificial para programadores con prisa by Marco Antonio Aceves Fernández - Books on Google Play*. Universo de Letras. [Online]. Available: https://play.google.com/store/books/details/Inteligencia_artificial_para_programadores_con_pri?id=ieFYEAAAQBAJ&hl=en_US&gl=US

[4] UCI Machine Learning (s.f) Iris Data Set. Recuperado el día 19 de Agosto de 2022 de http://archive.ics.uci.edu/ml/datasets/Iris

[5] UCI Machine Learning (2016) Iris Species. Recuperado el día 19 de Agosto de 2022 de https://www.kaggle.com/datasets/uciml/iris

[6] Scikit-learn(s.f) The Iris Dataset. Recuperado el día 19 de Agosto de 2022