Tendencias actuales y riesgos emergentes
en ciberseguridad:
Proyecto final del bloque de ML
Árbol de decisión

Flavio Rodrigues Dias (frodrd00@estudiantes.unileon.es)

Contenido

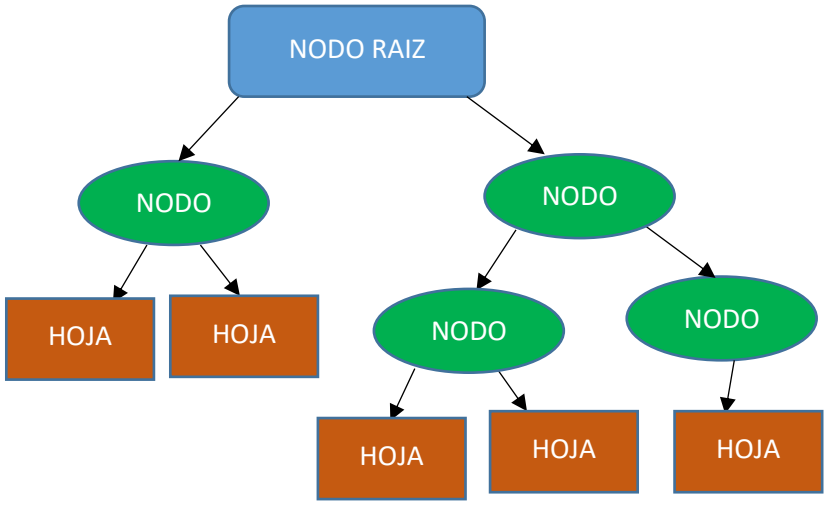
1 Teoría del modelo: árbol de decisión

El modelo de Machine Learning árbol de decisión o "Decision Tree" en inglés, es uno de los algoritmos más populares y potentes. Es popular debido a que este modelo es fácil de entender, y no se necesita ser un experto en el tema.

El árbol de decisión es un algoritmo de aprendizaje supervisado, hay varias ventajas de usar este modelo que son las siguientes:

- Se puede usar para predecir tanto valores continuos y discretos, es decir funcionan bien para las tareas de clasificación y regresión. (El que se va a desarrollar en este trabajo es el algoritmo de clasificación)
- Se requiere relativamente menos esfuerzo para entrenar el algoritmo que otros modelos.
- Se puede usar para clasificar datos no separables linealmente.
- Son muy rápidos y eficientes en comparación con KNN u otros algoritmos de clasificación.
- Es un buen modelo para detectar y eliminar "outliers", ya que la división se produce en función de la proporción de muestras dentro de los rangos divididos y no en valores absolutos.

En la siguiente imagen se puede ver la estructura de un árbol de decisión.



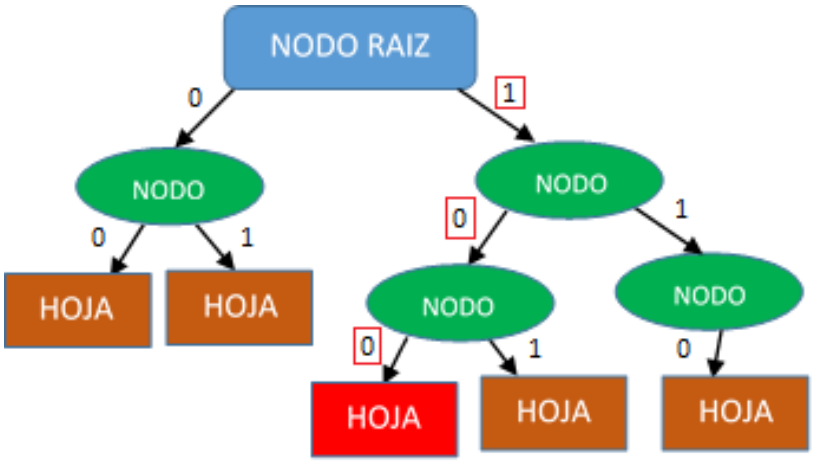
1

La idea principal del esquema anterior es encontrar aquellas características descriptivas que contengan la mayor "información" con respecto a la característica de destino y luego dividir el conjunto de datos a lo largo de los valores de estas características de manera que los valores de la característica de destino para los sub_datasets resultantes sean lo más puros posibles.

El proceso de encontrar la mejor característica se realiza hasta que se cumple con un criterio de parada en el que finalmente se termina con los nodos llamados hojas. Los **nodos hojas** son las clases del dataset, contienen la **predicción** para los nuevos datos que se quiera predecir con el modelo. Esto es posible ya que el modelo ha aprendido la estructura subyacente de los datos de entrenamiento y, por lo tanto, dado algunos supuestos, puede hacer predicciones sobre el valor del nodo hoja (clase) de las instancias de consulta.

Por ejemplo, dado una nueva muestra de datos que se quiera predecir a que clase pertenece, dependiendo de los valores de las características de esta muestra, va recorriendo cada nodo hasta llegar al nodo hoja roja, que es la predicción para la nueva muestra de datos.

Valores [1,0,0] -> Hoja roja (clase)



El árbol de decisión entonces contiene lo siguiente:

- Un **nodo raíz**: que es la característica que aporta más información dentro del dataset con el que se va a entrenar.
- **Nodos interiores**: que son las características del dataset, se van eligiendo las mejores características a la hora de formar un nuevo nodo del árbol. Tanto como el nodo raíz, como los nodos interiores se escogerá mediante la **métrica** que se aplique al modelo.
- **Nodos hoja**: son los nodos finales del árbol, las clases del dataset, representa el resultado final a la hora de ir tomando decisiones a lo largo del recorrido del árbol.
- **Arcos**: cada arco representa una decisión, estas decisiones son los valores que pueden tomar cada característica del dataset.

En función de la métrica que se escoja para elegir la mejor característica a lo largo del árbol, se obtendrá un algoritmo de aprendizaje u otro:

- Accuracy respecto de la clase mayoritaria (maximizar).
- Índice de Gini (minimizar).
- Entropía o ganancia de información (maximizar).

Para ver el funcionamiento el modelo se va a continuación se va a mostrar un ejemplo con un dataset muy pequeño. La métrica que se va a usar para elegir la mejor característica es la **entropía**. Este dataset contiene datos para elegir si se sale a comer fuera, se queda en casa a comer o no se come nada. Tiene el siguiente contenido:

- Las características son [‘hambre’,’dinero’,’coche’] y la clase [‘clase’].
- Los valores de las características [1,0], 1:True y 0:False
- Los valores de las clases [0,1,2], 0: salir a comer, 1: no comer, 2:comer en casa

hambre	dinero	coche	clase
1	1	1	0
0	0	0	1
0	0	1	1
1	0	0	2
1	1	0	2
1	0	1	0

Los pasos a seguir para escoger las mejores características por ejemplo para la métrica **entropía** son los siguientes:

1. Se calcula la entropía total del dataset.
2. Se coge cada característica y sus posibles valores, se dividen en datasets (sub_datasets) agrupando esos valores.
3. De cada subdataset se calcula la entropía
    a. Calculo entropía para hambre == 1
    b. Calculo entropía para hambre == 0
    c. Suma de las entropías de a) y b)
4. Se hace la resta de la entropía total y la entropía c) → información ganada
5. Se guarda el resultado y se vuelve a hacer el proceso para las demás características.
6. El mejor resultado anterior será la mejor característica que se elija para que sea el nodo raiz o los nodos interiores a la hora de ir creando nuevas ramas.

La forma de dividir el dataset en sub_datasets es la siguiente:

hambre	dinero	coche	clase
1	1	1	0
0	0	0	1
0	0	1	1
1	0	0	2
1	1	0	2
1	0	1	2

hambre == 1

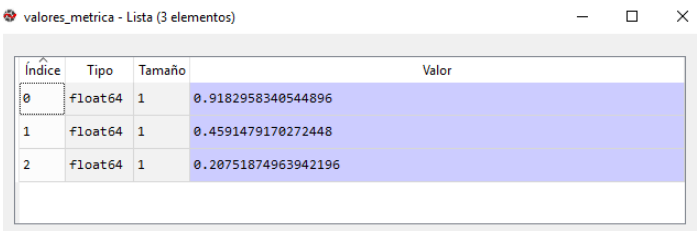
hambre	dinero	coche	clase
1	1	1	0
1	0	0	2
1	1	0	2
1	0	1	2

hambre == 0

hambre	dinero	coche	clase
0	0	0	1
0	0	1	1

Como en el primer subset los valores de las clases pueden valer 0 o 2, entonces se tendrá que seguir dividiendo en más subsets (en caso de que la característica hambre sea la elegida como mejor), borrando antes la columna 'hambre'. Mientras que para hambre == 0 las clases resultantes solo pueden tomar valor 1, entonces esto ya no se divide más y sería un **nodo final hoja** → si hambre == 0 entonces la **predicción** será 1 (no comer)

Se ha probado este dataset en el programa Python desarrollado y los resultados de las entropías en la primera ejecución son las siguientes:

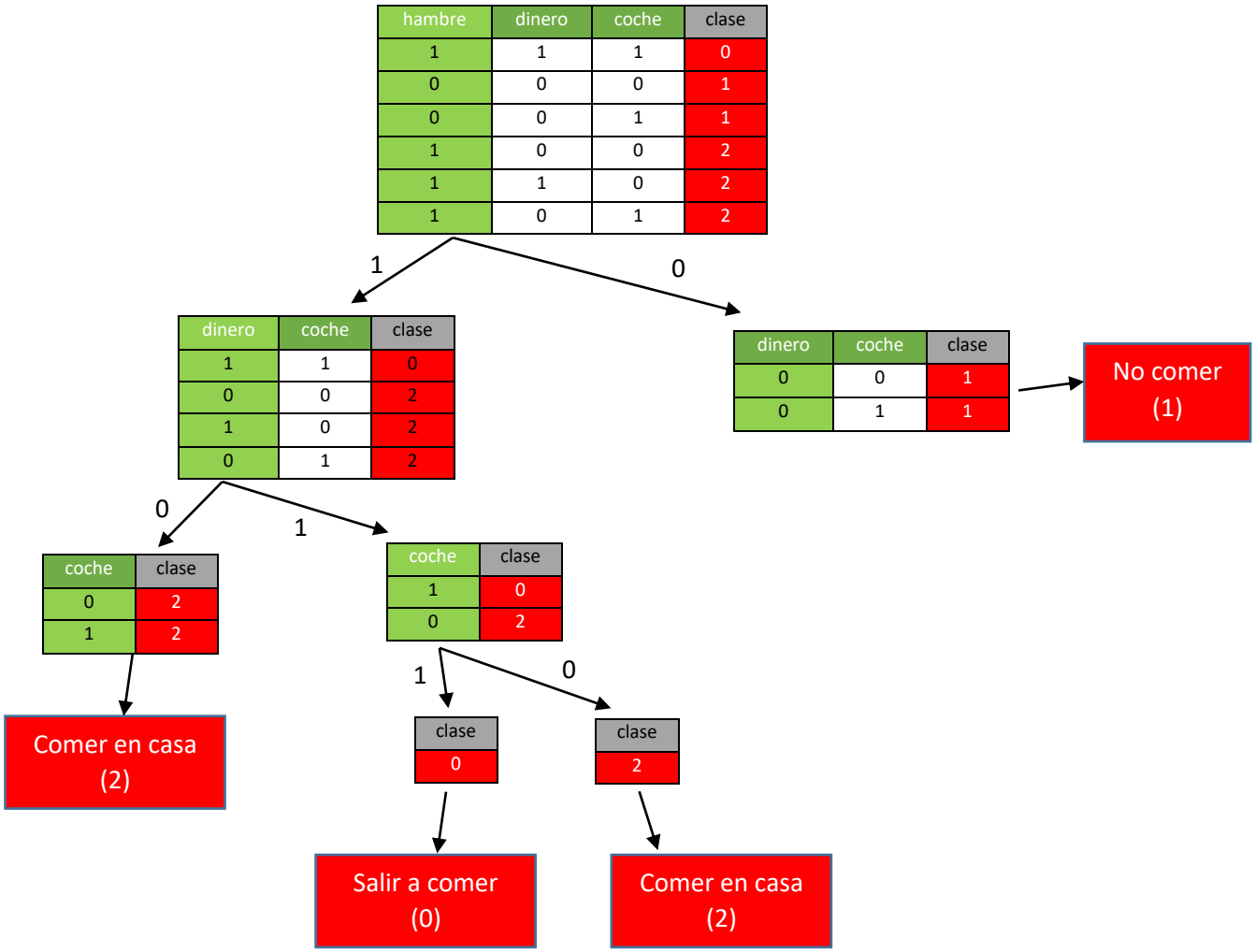


En este caso la característica que aporta más información en el datase es la de '**hambre'**.

Entonces como resultado final del algoritmo con la métrica entropía es la siguiente:

```
============================================================
El siguiente diccionario contiene el arbol de decision:
{'hambre': {0: 1.0, 1: {'dinero': {0.0: 2.0, 1.0: {'coche': {0.0: 2.0, 1.0: 0.0}}}}}}
============================================================
El accuracy del modelo con metrica entropia es: 100.00:
============================================================
Para las caracteristicas-> hambre: 1, dinero: 1, coche: 0,
Predice que es de la clase-> 2.0
============================================================
```

El resultado final en forma gráfica sería el siguiente:



2 Pseudocódigo del algoritmo árbol de decisión

A continuación, se introduce el pseudocódigo del modelo Decision Tree desarrollado:

1. Input **dataset, métrica, profundidad_maxima**
2. Se inicializa el árbol que es de tipo diccionario
3. Se definen los criterios de parada
    a. Si todas las filas pertenecen a la misma clase se devuelve un **nodo hoja**, que es la clase a la que pertenece esos valores fila.
    b. Si ya no hay características(columnas) a procesar devuelve la clase mayoritaria del dataset (sub_dataset)
    c. Si la profundidad máxima introducida por input llega a 0 también se devuelve la clase mayoritaria.

4. Si no se cumple con ningún criterio de parada el árbol puede seguir creciendo, formándose más nodos (ramas)
    a. Se selecciona la mejor característica con respecto a la métrica introducida como input.
        i. Si la métrica es ***entropía*** o ***accuracy*** se coge el valor más alto (maximizar)
        ii. Si la métrica es ***gini*** se coge el valor más bajo (minimizar)
    b. Con esta mejor característica se crea la raiz o un nuevo nodo en el árbol.
    c. Para cada valor v_j de la caracteristica anterior car_i
        i. Se divide el dataset a lo largo del valor de la car_i con la mayor ganancia de información y se crea el **sub_dataset**
        ii. Se borra la columna con mejor car_i que ya se ha procesado
        iii. Se llama al de nuevo a esta función (paso 1) de forma recursiva con los nuevos parámetros (sub_dataset, métrica, **prof_max -1**)
        iv. Se añade los nuevos nodos al árbol: **arbol[mejor_car$_i$][v$_j$] = sub_arbol**

5. Se devuelve el diccionario árbol

3 Descripción de todas las funciones de todos los archivos de Python.

Archivo main.py

def main(argv)

```
Función main del proyecto
:param argv: elementos que se pasan al ejecutar el proyecto
argv[0] -> nombre del archivo py
argv[1] -> nombre del fichero csv con el dataset
argv[2] -> nombre de la métrica: accuracy o gini o entropía
argv[3](opcional) -> split del dataset usar 'split' o 'no_split'
argv[4](opcional) -> profundidad maxima del arbol (numero entero)
```

def inicio(datos, metrica, split=None, profundidad_maxima=None)

```
Función para iniciar iniciar el modelo y para hacer la prueba de
predicción, tambien saca el accuracy del modelo dependiendo de la
metrica elegida
:param datos: = el conjunto de datos de entrenamiento (en formato
dataframe de Pandas)
:param métrica: = nombre de la métrica: accuracy o gini o entropia
:pram split: split del dataset usar 'split' o 'no_split'
:param profundidad_maxima: profundidad máxima del arbol
```

- En esta función se separan los datos en train y test si se desea, por defecto no_split.
- Por defecto no hay profundidad.
- Se crea el algoritmo del árbol y se entrena
- Se muestra el árbol de decisión
- Se calcula el accuracy del árbol respecto a la métrica escogida
- Se prueba a predecir una fila del dataset, se ha predicho la fila 4.

Archivo funciones_auxiliares.py

def readCSV(nombre)

```
Función para leer el dataset en formato csv
:param nombre: nombre del dataset a leer, ej: dataset.csv
:return dataset: dataset en formato dataframe
```

def split_dataset(dataset)

```
Función para separar el dataset en datos de entrenamiento y test
:param dataset: dataset a utilizar
:return (training_data, test_data): datos de entrenamiento y datos
de test
```

def entropy_info_gain(dataset, feature_name)

```
Calcula la ganancia de información en la entropía, parámetros:
:param dataset: el dataset a ser utilizado con los datos
:param feature_name: el nombre de la característica para la que
se va a calcular la ganancia de información
:return info_gain: información ganada entropía
```

- Se coge el nombre de la columna clase
- Se calcula la entropía total del dataset
- Calcular las clases únicas y la cantidad de cada una para la característica (feature_name)
- Se crea una lista que contiene “series” con los sub_datasets posibles valores de las clases que puede tomar cada valor de la característica pasada como parámetro

- Se calcula la entropía para cada posible valor de la característica (feature_name)
- Se obtiene la información de cada característica.
- Finalmente se calcula la información ganada: resta de la entropía total menos la entropía para cada característica.

def entropy(columna)

```
Función para calcular la entropía: entropy = - SUM(P(clasei) * log2(clasei))
:param columna: columna del dataset a calcular su entropía
:return -e: entropía
```

def metrica_accuracy(dataset, feature_name)

```
Función para calcular el accuracy respecto a la clase mayoritaria
:param dataset: datos del dataset
:param feature_name: nombre de la característica
:return (1 - errores_totales/len(dataset)): accuracy
```

- Se coge el nombre de la columna clase.
- Se inicializa contador de errores para la característica seleccionada
- Calcular las clases únicas y la cantidad de cada una para la característica (feature_name)
- Se crea una lista que contiene "series" con los sub_datasets posibles valores de las clases que puede tomar cada valor de la característica pasada como parámetro.
- Se obtiene la clase mayoritaria respecto a los sub_datasets creados
- Se obtiene el número de errores de cada sub_dataset y se acumulan.
- Se devuelve 1 – errores totales / longitud del dataset pasado como input.

def metrica_gini(dataset, feature_name)

```
Función para calcular el índice de gini respecto de cada característica pasada por parámetro, del dataset
:param dataset: datos del dataset
:param feature_name: nombre de la caracteristica
:return gini: índice de gini para esa columna pasada feature_name
```

- Se coge el nombre de la columna clase
- Calcular las clases únicas y la cantidad de cada una para la característica (feature_name)
- Se crea una lista que contiene "series" con los sub_datasets posibles valores de las clases que puede tomar cada valor de la característica pasada como parámetro.

- Para cada sub_dataset se calcula el índice $Gini(\mathcal{D}) := 1 - \sum_{i=1}^{n} p_i^2$
- Se devuelve $Gini(\mathcal{D}, car_i) := \sum_{j=1}^{l} \frac{\#\mathcal{D}_{v_j}^{car_i}}{\#\mathcal{D}} Gini(\mathcal{D}_{v_j}^{car_i})$

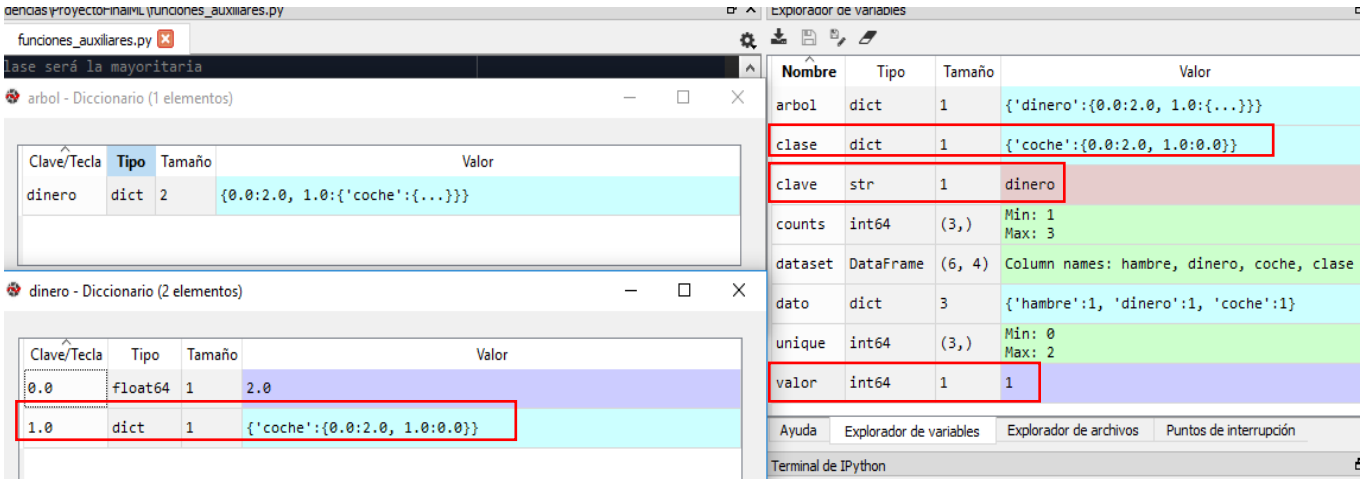
def prediccion(arbol, dato, dataset)

```
Función para predecir una muestra con los valores de las características
:param árbol:  árbol de decisión
:param dato: una fila de la base de datos sobre la que queremos predecir la clase.
:param dataset: dataset original usado para calcula clase mayoritaria
```

- Se coge cada característica/clave del conjunto de dato dict.
- Se coge cada clave del conjunto de datos del arbol tipo dict.
- Se recorre la lista de claves de dato.
- Si la clave esta es el nodo raíz del árbol.
- Intenta lo siguiente
  - Coger el valor1 de la clase en el dict dato pasado como parámetro a la función.
  - Coger el valor2 (puede ser otro dict) en el árbol con clave:valor1
- Si no se devuelve la clase mayoritaria del dataset.
- Si el contenido de variable 'clase' tiene un diccionario sigue buscando dentro de este, que pasa a ser la nueva variable árbol, hasta que la clase sea un nodo hoja.
- Si no retorna la variable 'clase': nodo hoja
- Si al final no puede llegar a predecir por algún motivo, retorna None, dado que es una función de tipo void.

Por ejemplo para valor == 1 de dato['dinero'] y clave == 'dinero'

Se obtiene la clase == {'coche': {0.0: 2.0, 1.0: 0.0}} de arbol[clave][valor]



def accuracy_del_arbol(dataset,arbol)

```
Función para comprobar el accuracy del modelo árbol que se ha creado antes
:param dataset: dataset cargado
:param árbol: modelo con el árbol creado
:return accuracy: accuracy del árbol
```

- Se coge el nombre de la columna clase.
- Se crea un array con las filas y borra la columna clase del dataset que se le pasa por parámetro to_dict: Convertir el dataframe a un diccionario, records -> [{column -> value}, … , {column -> value}]
- Se crea un dataframe llamado predicho con la columna de predicción
- Se calcula la predicción de cada fila del dataset.
- Se calcula el accuracy total.

Archivo modelo.py

def DT(dataset, metrica, profundidad_maxima=None)

```
Función para crear el algoritmo del árbol de decisión, toma 4 parámetros:
:param dataset: los datos con los que crear el árbol de decisión, en la primera iteracion se usa el dataset entero.
:param metrica: metica elegida-> entropia, gini o accuracy
param profundidad_maxima: profundidad del arbol maxima
:return arbol: arbol de decision generado
```

El pseudocódigo del modelo ya se ha explicado en el **apartado 2** de este documento.

4 Ejecución del script Python

Se deberá ejecutar el script Python main.py. Los inputs que recibe este archivo están descritos en el **apartado 3 -> archivo main.py -> función main**

- Por ejemplo, serán válidas las siguientes ejecuciones:
  Con métrica = accuracy, gini o entropia

  - python main.py archivo.csv métrica
  - python main.py archivo.csv métrica profundidad_max
  - python main.py archivo.csv métrica no_split
  - Python main.py archivo.csv métrica split
  - python main.py archivo.csv métrica no_split profundidad_max
  - Python main.py archivo.csv métrica split profundidad_max

- Se ha probado con el dataset de **zoo.csv**[1] que tiene la siguiente forma:

```
pelo,plumas,huevos,leche,volador,acuatico,predador,dentadura,columna,respira,veneno,aletas,patas,cola,domestico,tamano,clase
1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,mamifero
1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,mamifero
```

	pelo	plumas	huevos	...	domestico	tamano	clase
0	1	0	0	...	0	1	mamifero
1	1	0	0	...	0	1	mamifero
2	0	0	1	...	0	0	pez
3	1	0	0	...	0	1	mamifero
4	1	0	0	...	0	1	mamifero
5	1	0	0	...	0	1	mamifero

Por ejemplo, para la ejecución→ **python main.py zoo.csv entropia 10** tiene la siguiente salida:

(Para la predicción se ha cogido una fila del dataset original)

```
==========================================================
El siguiente diccionario contiene el arbol de decision:
{'patas': {0: {'aletas': {0.0: {'dentadura': {0.0:
'invertebrado', 1.0: 'reptil'}}, 1.0: {'huevos': {0.0:
'mamifero', 1.0: 'pez'}}}}, 2: {'pelo': {0.0: 'ave', 1.0:
'mamifero'}}, 4: {'pelo': {0.0: {'acuatico': {0.0: 'reptil', 1.0:
{'dentadura': {0.0: 'invertebrado', 1.0: 'anfibio'}}}}, 1.0:
'mamifero'}}, 6: {'acuatico': {0.0: 'insecto', 1.0:
'invertebrado'}}, 8: 'invertebrado'}}
==========================================================
El accuracy del modelo con metrica entropia es: 100.00%:
==========================================================
Para las caracteristicas-> pelo: 1, plumas: 0, huevos: 0, leche:
1, volador: 0, acuatico: 0, predador: 1, dentadura: 1, columna:
1, respira: 1, veneno: 0, aletas: 0, patas: 4, cola: 1, domestico:
0, tamano: 1,
Predice que es de la clase-> mamifero
==========================================================
```

[1] http://archive.ics.uci.edu/ml/datasets/zoo