CI-0148 Aprendizaje Máquina
Prof. Pablo Sauma Chacón

Laboratorio 2

Objetivo: Familiarizar al estudiante con la creación y el uso de árboles de decisión. Así como las diferentes métricas que existen para medir el rendimiento de los modelos implementados.

Enunciado: Lea las instrucciones del documento y resuelva el enunciado en Python. Al inicio de su documento adjunte su nombre y carnet como un comentario. Este trabajo es de carácter individual, tampoco se aceptarán códigos que no hayan sido desarrollados por su persona.

Para este laboratorio se trabajará con tres sets de datos diferentes: mushrooms.csv, iris.data y titanic.csv. El primero consiste en un set de datos con atributos estrictamente categóricos de diferentes variedades de hongo, el objetivo de dicho conjunto de datos es lograr diferenciar entre un hongo comestible y uno tóxico. El segundo set de datos consiste de atributos estrictamente numéricos de diferentes variedades de flor iris, el objetivo de dicho conjunto de datos es lograr diferenciar entre las tres diferentes especies de iris con base en las características de la flor. El último conjunto de datos consiste en atributos tanto numéricos como categóricos con respecto a la sobrevivencia de pasajeros en el Titanic, el objetivo de dicho conjunto de datos es lograr predecir si cada pasajero logró sobrevivir o no el hundimiento.

1. Función `Gini(y)` que recibe un objeto `pandas.Series` y calcula el coeficiente de Gini para dicha serie.
    a. Recuerde que los valores de y puede ser de tipos de datos diferentes como strings (mushrooms), booleanos o incluso números enteros.

2. Función `Gini_split(ys)` que recibe un arreglo con diversos objetos `pandas.Series` y calcula el coeficiente de $\text{Gini}_{\text{split}}$ para esta división.

Para trabajar con los sets de datos deberá implementar la clase **`DecisionTree`** con los siguientes métodos:

3. Método `fit(self, x, y, max_depth = None)` que recibe un objeto `pandas.DataFrame` `x`, un objeto `pandas.Series` `y` y (opcionalmente) un entero `max_depth`. El parámetro `x` contiene los datos y atributos que utilizará el modelo en sus predicciones; el parámetro `y` contiene la categoría a la que pertenece cada uno de los datos de `x`; `max_depth` es un parámetro opcional que de ser diferente a `None`, indica cuál es el valor máximo de profundidad para el árbol. Cuando se llama este método se deberán eliminar los datos previos del modelo y rehacer el árbol de decisión desde cero. Considere que para esto podría resultar útil una clase Nodo, el árbol será de tipo binario. Para cada nodo deberá analizar los siguiente:
    a. Si todos los valores restantes pertenecen a la misma clase entonces el nodo es un nodo hoja y pertenece a la clase de sus datos. Si ninguna partición

posible genera ganancia de información (si `IG == 0`) o si dividir el nodo excedería el límite de profundidad permitida, entonces el nodo es un nodo hoja y pertenece a la clase de la moda de los datos (la clase de la mayoría de los datos).

b. Si los valores restantes pertenecen a distintas clases y aún no llegamos al límite de profundidad entonces es necesario encontrar el split óptimo para el nodo: para cada uno de los atributos de la tabla de datos `x` se deberá evaluar una variedad de opciones para la división dependiendo de si los datos son categóricos o numéricos.
    i. Puede extraer las columnas numéricas de un `pandas.DataFrame` con la siguiente instrucción: `x.select_dtypes(include=['int16', 'int32', 'int64', 'float16', 'float32', 'float64']).columns`
c. Si los datos del atributo son de tipo numérico, deberá calcular el máximo y el mínimo para dicho atributo, si son iguales entonces no se debe considerar este atributo para la partición; si son diferentes entonces deberá calcular 10 puntos intermedios (para ello puede utilizar el comando `numpy.linspace`), y deberá intentar hacer la división en esos 10 diferentes puntos, separando los datos entre <= al punto y > al punto.
d. Si los datos del atributo son de tipo categórico, deberá verificar la cantidad de valores únicos para dicho atributo, si solo hay un valor entonces no se debe considerar este atributo para la partición; si hay más de una categoría entonces deberá probar separar los datos acorde a cada una de las categorías, separando los datos entre aquellos == a la categoría y los != a la categoría. Debe probar la división con cada una de las categorías.
e. Cada vez que intente una división deberá verificar si dicha división genera una ganancia de información mayor a la que se ha encontrado hasta el momento y de ser así conservar la información de dicha división. Una vez que haya intentado hacer la división con cada uno de los atributos/columnas acorde a su tipo deberá haber encontrado la división óptima para el nodo (o descubierto que ningún atributo es válido para división, en cuyo caso el nodo sería nodo hoja).
f. Si se tiene una división óptima válida, entonces se crearán dos nodos hijos: el hijo izquierdo (<= para numérico, == para categórico) e hijo derecho (> para numérico, != para categórico); los cuáles recibirán únicamente sus datos pertinentes y llamarán recursivamente a su constructor continuando la construcción del árbol.

4. Método `predict(self, x)` que recibe un objeto `pandas.DataFrame x` que contiene los valores de un conjunto de datos a predecir. El método debe retornar un objeto de tipo `pandas.Series` (si se tiene un arreglo array basta con ejecutar el llamado `pandas.Series(array)`) con las clases a las que pertenece cada uno de los datos de `x`.
    a. Esto equivale a iterar a lo largo de la tabla de datos (esto se puede hacer mediante un ciclo `for index,row in x.iterrows()`) y para cada fila realizar el proceso de clasificación descendiendo por el árbol.
    b. Se inicia en el nodo raíz, y para cada nodo de tipo *split* se evalúa conforme a su división (columna/atributo + el valor con el que se evalúa). Dependiendo

del resultado se desciende por el hijo izquierdo o derecho. Al llegar a un nodo hoja se retorna la clase de dicho nodo y se considera que esa es la clase de los datos evaluados.

5. Método `to_dict(self)` que retorna el árbol en formato de diccionario.
    a. Si un nodo es nodo hoja su diccionario debe contener la siguiente información, dónde `<clase>` es la clase a la que pertenecen los elementos de dicha hoja y `<conteo>` la cantidad de elementos que conforman dicha hoja:

```
{
      "type": "leaf",
      "class": <clase>,
      "count": <conteo>
}
```

    b. Si un nodo es nodo no hoja su diccionario debe contener la siguiente información, dónde `<conteo>` es la cantidad de elementos que forman parte de dicho nodo, `<gini>` es el coeficiente de Gini para dicho nodo, `<tipo>` es el tipo de split (`categorical/numerical`), `<columna>` el nombre de la columna sobre la que opera el split, value el valor que se utiliza para separar los datos y recursivamente los hijos derecho e izquierdo del nodo:

```
{
      "type": "split",
      "gini": <gini>,
      "count": <conteo>,
      "split-type": <tipo>,
      "split-column": <columna>,
      "split-value": <valor>,
      "child-left": <diccionario hijo izquierdo>,
      "child-right": <diccionario hijo derecho>
  }
```

6. Método `calculate_confusion_matrix(predict, real)` que recibe dos objetos `pandas.Series` `predict` y `real`, que corresponden a las clases estimadas por el modelo de árbol de decisión y los valores reales para un subconjunto de datos. Teniendo los datos estimados y los datos reales deberá calcular y retornar la matriz de conclusión de dichos datos.

7. Utilice los datasets provistos para verificar el correcto funcionamiento de su árbol. Además puede utilizar el método `train_test_split` de la biblioteca `sklearn.model_selection` para separar un conjunto de datos en un conjunto de datos de entrenamiento y otro de prueba, para verificar el rendimiento de los árboles generados.
    a. Documente sus resultados obtenidos.
    b. En sus propias palabras: ¿qué mejoras o funcionalidades adicionales se le ocurre que podría incluir en su clase `DecisionTree`? Mencione al menos 2.

c. Compare el rendimiento de su árbol con el de un árbol de la biblioteca `sklearn`, ¿qué tanto varían los resultados para el mismo set de datos de entrenamiento-prueba?

Para hacer la verificación de su sistema más fácil puede utilizar el siguiente código de prueba que verifica el funcionamiento de su código: la función de Gini, Gini_split, la construcción de un árbol completamente categórico, la construcción de un árbol completamente numérico y uno mixto y finalmente, la predicción y cálculo de la matriz de confusión. Como parte del código de prueba se requerirá el siguiente archivo de datos que contiene la información del resultado final de los árboles.