Arboles de Decisión y Bosques Aleatorios

M. Sc. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Computación, bachillerato en Ingeniería en Computación,
PAttern Recongition and MAchine Learning Group (PARMA-Group)

20 de enero de 2022

En el presente material se estudiarán los arboles de decisión, para luego implementarlos en el contexto de clasificación.

1. Arboles de Clasificación y Regresión (CART)

Leo Breiman en (Classification and Regression Trees By Leo Breiman, Jerome Friedman, Charles J. Stone, R.A. Olshen), acuñó el acronimo CART refiriendose a los árboles de clasificación y regresion en ingles. Un CART se refiere usualmente a un árbol binario. En tal árbol, cada nodo está compuesto por una variable o característica x_d y un valor de umbral o división τ_d . Tal nodo tiene entonces dos hojas al referirse a un árbol binario, los cuales generan la ruta para tomar la decisión según el modelo, generando la salida $\widetilde{t}$ del modelo.

Tómese el ejemplo del conjunto de datos del *Titanic,* el cual se ilustra en la Tabla *1.* Cada registro se refiere a las caracteristicas o *features* medidos para el individuo (edad, clase, tarifa, y si sobrevivió o no al naufragio*).* Tales caracteristicas son un sub-conjunto de las caracteristicas contenidas en el dataset de Kaggle `https://www.kaggle.com/c/titanic/data`. En general un conjunto de datos se define como $D = \langle X, T \rangle$ donde $X = \{\overrightarrow{x}_1, \overrightarrow{x}_2, \ldots, \overrightarrow{x}_n\}$ es el conjunto de n observaciones y $T = \{t_1, t_2, \ldots, t_n\}$ el conjunto de etiquetas correspondiente.

# Registro	Clase	Edad	Tarifa	Sobrevivió
0	3	22	7.2	0
1	1	38	71.3	0
2	3	26	7.9	1
3	1	35	53.1	0

Cuadro 1: Datos de muestra del *dataset* del Titanic.

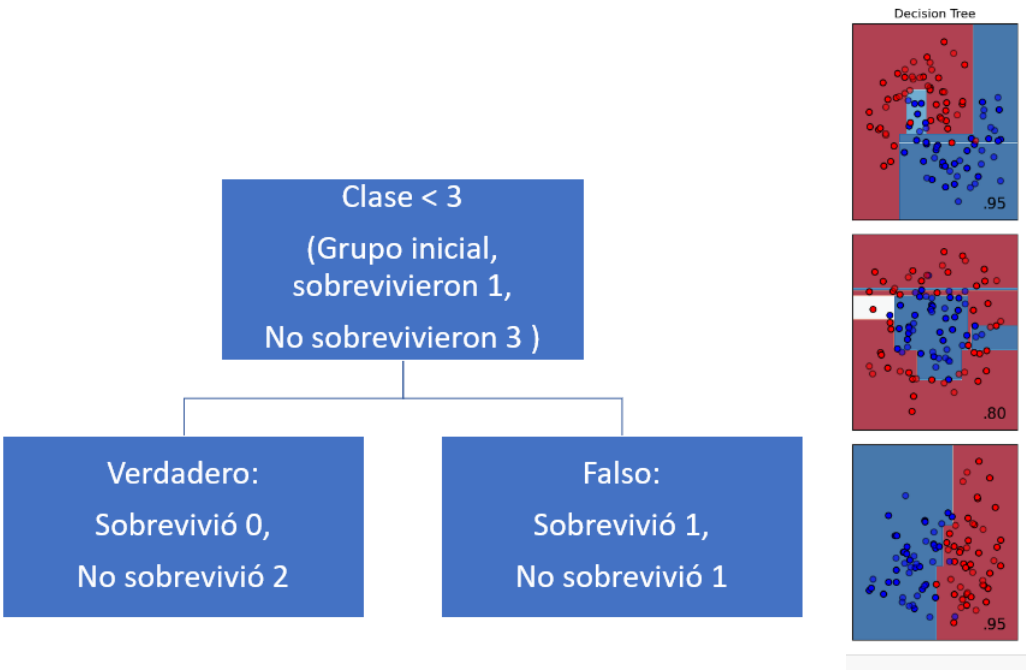

Figura 1: Arbol de decisión de ejemplo con la variable Clase y el valor de umbral.

El objetivo del CART es partir los datos a partir de una o más variables y un umbral en cada una de ellas, para generar un árbol que permita dar con la decisión correcta, según los datos. Ejemplo de un CART se muestra en la Figura 1, con $d =$ clase y $\tau_d = 3$. Los nodos hoja muestran la tasa de aciertos generada por la partición, la cuál es del 100 % para el nodo de *verdadero* y de 50 % para el nodo de *falso*. La hoja define tambien la clase dominante o clase con mayor cantidad de observaciones en esa hoja, de modo que esa información pueda ser usada durante la etapa de inferencia, luego de ser construido el árbol (entrenamiento del modelo).

La estimación del CART $\widetilde{t}_i = f\left(\overrightarrow{x}_i\right)$ corresponde a la clase dominante del nodo hoja alcanzado luego de evaluar todas las decisiones dentro del árbol.

1.1. Funciones de pérdida o evaluación de particiones

¿Qué tan buena, según los datos de entrenamiento, es una partición generada por una caracteristica y umbral elegidos $x_{d,\tau}$? Para ello es necesario evaluarla de forma cuantitativa, a través de lo que se conoce en general en el aprendizaje automático como **función de error o pérdida.** En el contexto de los CART, es frecuente usar como base de la función de error, **la función de impu-**

reza Gini. Tal función, mide la proporción a_k para cada clase, en una partición especifica ρ:

$$E_{\text{gini},\rho}\left(\tau_d\right) = 1 - \sum_{k=1}^{K} a_k^2.$$

Así por ejemplo, para la partición inicial $\rho = 1$ en la raíz del arbol, la proporción para la primer clase (no sobrevió) es de $\frac{3}{4}$, para la segunda clase (sobrevió), es de $\frac{1}{4}$, por lo que entonces:

$$E_{\text{gini},\rho=1}\left(\tau_d\right) = 1 - \frac{3}{4}^2 - \frac{1}{4}^2 = 0{,}375.$$

Para el segundo nodo (de izquierda a derecha la primer hoja), se tiene entonces que:

$$E_{\text{gini},\rho=2}\left(\tau_d\right) = 1 - \frac{0}{2}^2 - \frac{2}{2}^2 = 0.$$

Observe que en cuanto la partición tiene una mayor proporción de una clase respecto a las demás, más *pura* es, por lo que $E_{\text{gini},\rho} \to 0$. Por eso se le llama **función de impureza de Gini, entre más alta, más impura es la partición**. El objetivo al construir un CART, es entonces lograr que todas las hojas tengan un $E_{\text{gini},\rho} \to 0$.

La evaluación total del error o la pérdida del CART, se hace entonces al sumar (usualmente de forma ponderada), las impurezas de todos los nodos hoja en un CART. Siguiendo el ejemplo, calculamos la impureza de Gini para la última hoja (de izquierda a derecha), con $\rho = 3$:

$$E_{\text{gini},\rho=3}\left(\tau_d\right) = 1 - \frac{1}{2}^2 - \frac{1}{2}^2 = 0{,}5.$$

1.2. Construcción del CART

Construir un CART que genere un error bajo o cero es el objetivo. El método más común es siguiendo la **heurística «voraz» o *greedy*, donde luego de desarrollar los primeros nodos hoja, se elige el nodo con menor impureza Gini**. Es claro que al utilizar este enfoque, el árbol final puede que no sea el óptimo (con el menor error posible), puesto que **tomar la decisión *localmente* de desarrollar un nodo según su impureza, puede que a la larga genere otros nodos de mayor impureza.**

Para desarrollar un nodo y elegir el umbral, es necesario probar todos los umbrales posibles dentro del conjunto de datos. En el ejemplo, si eligieramos el atributo *edad*, deberíamos entonces probar con todas las edades de los registros en el dataset, en este caso 22, 26, 35, 38, y quedarnos con la partición que menor Gini ponderado genere. Posteriormente, se elige la partición con menor Gini para continuar recursivamente con el proceso.

Para decidir cuál *feature* y umbral escoger, utilice un ponderado de los posibles coeficientes de gini para las posibles particiones del nodo izquierdo y

derecho. Si n es la cantidad de observaciones en la partición, con además n_i y n_d la cantidad de observaciones en la partición izquierda y derecha, respectivamente, el coeficiente de gini ponderado sugerido para decidir el feature y umbral óptimos es:

$$\overline{E}_{\text{gini}}(\tau_d, d) = \frac{n_i}{n} E_{\text{gini}}(D_i) + \frac{n_d}{n} E_{\text{gini}}(D_d)$$

donde D_i es el conjunto de datos de la partición izquierda y D_d el conjunto de datos de la partición derecha.

Se procede entonces, para una partición de datos D, a probar todos los posibles *features* y sus umbrales para dar con el mínimo coeficiente de gini, y realizar este procedimiento de nuevo para los nuevos nodos hijos, por lo que el procedimiento se puede definir recursivamente. En general es aconsejable, además de la necesidad de guardar los umbrales y sus *features* ya incluidos dentro del arbol, también guardar la lista de features probados hasta ahora en el árbol, de modo que no se re-usen *features* anteriormente escogidos en el árbol.

Una vez completado la construcción del CART (equivalente al entrenamiento del modelo, en este caso no paramétrico), la evaluación de la pertenencia de clase para una nueva observación se hace evaluando los umbrales a través del árbol, hasta llegar a una hoja. Una vez que llegamos a esa hoja, se define como etiqueta predicha $\widetilde{t}$, la clase con mayor cantidad de observaciones en tal nodo, según el conjunto de datos de entrenamiento.

En general, un árbol de decisión incluye los siguientes hiper-parámetros a escoger por el usuario:

- **Máxima profundidad ρ del CART**. Durante la construcción del árbol, la profundidad puede variar según el conjunto de datos. Si solamente se utiliza un feature de forma única, la profundidad máxima va a corresponder a la cantidad de *features* que definen la observación. De lo contrario, la profundidad puede ser arbitraria.
- **Mínimo coeficiente de gini**: Otra forma de evitar la creación de más nodos, es establecer un coeficiente de gini mínimo, por lo que si una nueva partición consigue un gini menor a tal umbral eligido, el agregado de más nodos hijos se detiene.
- C**antidad mínima de observaciones del nodo**: Una forma optativa de evitar el sobre-ajuste a los datos, es evitar particiones que sobre-segmenten los datos, es decir, generen particiones muy pequeñas.

Tales hiper-parámetros pueden ayudar a evitar el sobre-ajuste a los datos. Otra alternativa para lidiar con este problema, es utilizar técnicas de *podado* o *pruning* del árbol.

1.3. Ventajas y desventajas de los árboles de decisión

Ventajas:

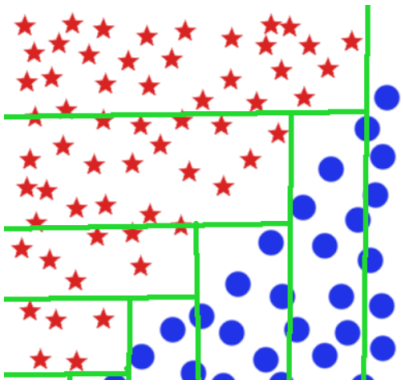

Figura 2: Datos linealmente separables con relación aditiva.

1. Los árboles de decisión son fáciles de entender y *explican* el porqué de su estimación.

2. Su evaluación y entrenamiento es relativamente rápido, respecto a otros métodos como las redes neuronales.

3. Los atributos o *features* a utilizar pueden ser tanto numéricos como categóricos. En el caso de los atributos categóricos, la construcción del árbol prueba distintas particiones de los valores posibles para la variable categórica.

4. Los árboles de decisión pueden ser fácilmente extendidos para realizar una regresión. En el caso de una regresión, la estimación del árbol de decisión $\widetilde{t}_i = f\left(\overrightarrow{x}_i\right)$ corresponde al valor promedio de las etiquetas dentro del nodo hoja alcanzado luego de evaluar todas las decisiones dentro del árbol. Para escoger el umbral óptimo y realizar la partición, se minimiza la función de la suma de los cuadrados del error, para todas las observaciones en cada particion:

$$E_{\text{SSE},\rho}\left(\tau_d\right) = \sum_{i=1}^{n} \left(\widetilde{t}_i - t_i\right)^2$$

   donde n se refiere a la cantidad de observaciones en la partición (ya sea izquierda o derecha).

Desventajas:

1. Los árboles de decisión no son apropiados para datos linealmente separables con relación aditiva, como se ilustra en la Figura 2

2. Alto riesgo de sobre-ajuste a los datos, con pocos métodos para evitarlo.

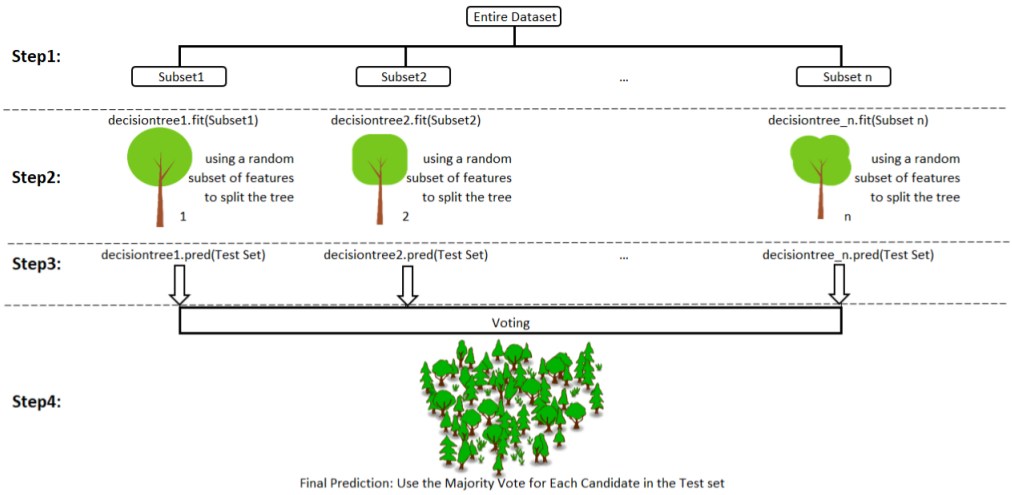

Figura 3: Ejemplo de un bosque aleatorio con n árboles de decisión. Cada modelo se entrena con una partición distinta de datos $D_1, \ldots, D_n$, método conocido como *bagging*. Tomado de `https://bit.ly/3vL2GoX`.

1.4. Aprendizaje por «consejos» o *ensemble learning* con bosques aleatorios

Los CART pueden tender a sobreajustarse a los datos, al probar como umbrales únicamente los valores existentes dentro del conjunto de datos de entrenamiento D_T. Para aliviar este problema, se desarrollaron los bosques aleatorios o *random forests*. Un bosque aleatorio está compuesto por 2 o más árboles de decisión, cada uno entrenado o construído con una partición distinta de datos. Luego de construidos, la inferencia final se calcula con el resultado de una *votación* entre los P árboles de decisión, como se ilustra en la Figura 3. Librerías de Python como scikit learn implementan los bosques aleatorios por ejemplo, en el paquete *sklearn.ensemble.RandomForestClassifier*.

Una variante del *ensemble learning* es el aprendizaje por *boosting* o *impulso*. El concepto del boosting se refiere al aprendizaje en *serie*, usando los errores del modelo anterior, para ponderar el error de forma consecutiva, en los demás modelos. Además, usualmente cuando se implementa boosting, el error e_i dentro de la serie de errores $e_1, \ldots, e_i, \ldots, e_n$, se utiliza para ponderar la decisión de cada arbol.

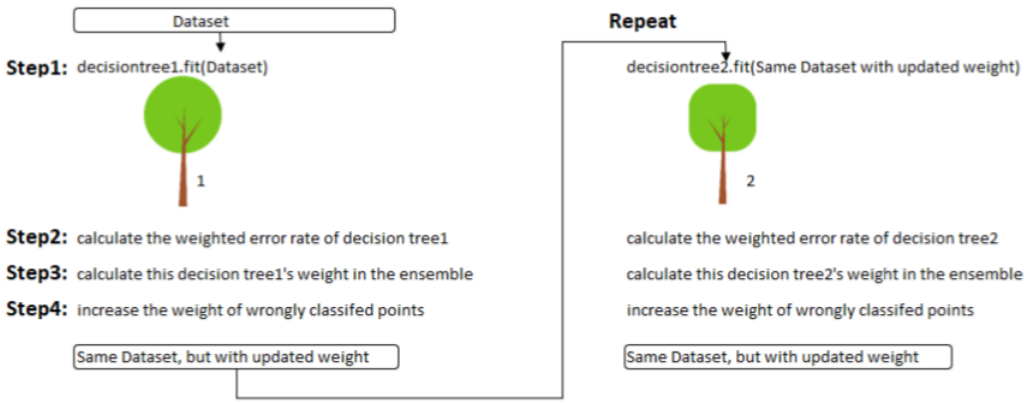

Figura 4: Boosting usando árboles de decisión. Tomado de `https://bit.ly/3vL2GoX`.