Árboles de decisión (Aprendizaje supervisado: tenemos datos de la variable target)

Principales 2 caracteristicas:

- Sirven como ejemplo para entender la estructura y funcionamiento de otros algoritmos.

- Sirven de baseline trivial. Sencillo y fácil como punto de referencia para comparar contra otras técnicas más complejas.

Dato: Random Forest es un conjunto de estos tipos de árboles y es una de las mejores técnicas.

¿Qué busca? Aproximacion de funciones: buscamos una funcion que aproxime el **criterio** (como salir a caminar o no)

La salida del algoritmo de aprendizaje es la hipótesis que aproxima a la función F. La función ideal es una que tiene muchos atributos y nos devuelve la predicción.

Tenemos un algoritmo para entrenar un modelo, y ese modelo nos va a ayudar para predecir.

Para construir un árbol, tomamos un atributo y lo ubicamos en un nodo del árbol. Cada nodo evalúa un atributo discreto.

Cada nodo tiene el “mejor” atributo para dicho nodo.

Arrancamos con una tabla con instancias atributos y clases

atributos — clase

instancias

Cielo	Temperatura	Humedad	Viento	¿Camina?
Sol	Calor	Alta	Débil	No
Sol	Calor	Alta	Fuerte	No
Nublado	Calor	Alta	Débil	Sí
Lluvia	Templado	Alta	Débil	Sí
Lluvia	Frío	Normal	Débil	Sí
Lluvia	Frío	Normal	Fuerte	No
Nublado	Frío	Normal	Fuerte	Sí
Sol	Templado	Alta	Débil	No
Sol	Frío	Normal	Débil	Sí
Lluvia	Templado	Normal	Débil	Sí
Sol	Templado	Normal	Fuerte	Sí
Nublado	Templado	Alta	Fuerte	Sí
Nublado	Calor	Normal	Débil	Sí
Lluvia	Templado	Alta	Fuerte	No

Buscamos una funcion F(x1,x2,x3..)→ Y
Tenemos un espacio de hipotesis (un arbol por ejemplo)
Datos de entrenamiento, que son todos los pares X, Y

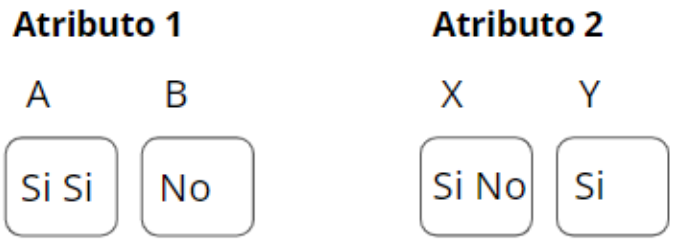
¿Cómo se arma el arbol?

5 pasos

- Elegir mejor atributo A
- Asignarlo como decision del nodo actual
- Para cada valor de A creamos una rama (hijo)
- Repartir las instancias en los nuevos nodos
- Si ya están bien separadas todas, no iterar más. sino iterar sobre los que no estan bien separados

Concepto homogeneidad o pureza de la variable objetivo

Queremos determinar para cada nuevo nodo, que atributo utilizar para repartir nuestras instancias. La idea es que la creación de los subnodos incrementa la homogeneidad o pureza respecto a la variable objetivo.



Si elijo el atributo 1 la muestra es más homogénea.

¿Cómo elegimos el mejor atributo para el split ? 2 opciones

Impureza de Gini: **GiniGain**

Quiero reducir la impureza. Gini gain es la reducción de la impureza respecto a la variable objetivo. Gini gain = impureza que tenia antes - impureza que tengo ahora
En Gini, las clases son los valores posibles (para el ejemplo puede ser las clases son “si” y “no”). El gini se calcula por hoja.

- La impureza de una muestra S sería la siguiente

$$Gini(S) = 1 - \sum_{c \in Clases} \left(\frac{|S_c|}{|S|}\right)^2$$

Siendo S_c el conjunto de instancias que pertenecen a la clase c

- Reducción de impureza de S con respecto al atributo A (Gini Gain)

$$GiniGain(S, A) = Gini(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} Gini(S_v)$$

Donde Valores(A) son los valores posibles de A.
Sv = {s ϵ S | A(s) = v}

- Elegimos el atributo que más reduce la impureza

Ganancia de información: **InfoGain**

Utiliza el concepto de **entropía**. Es como una medida de incertidumbre. Cuanta más certeza, más homogénea. La entropía es cero cuando la probabilidad de distribución es 0. Queremos reducir la entropía.

- **H(s) mide la entropía.**

- Entropía de una muestra S con respecto al feature

$$H(S) = \sum_{c \in Clases} -p_c \log_2 p_c$$

p_c: proporción de instancias S pertenecientes a C

- Reducción de entropía de la muestra S respecto al atributo A

$$InfoGain(S, A) = H(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} H(S_v)$$

- Elegimos el atributo que más reduce la entropía

Para reducir la entropía usamos infoGain: Prueba la entropía anterior menos la entropía de haber elegido cada atributo. Voy a tener que probar con todos los atributos.

¿Qué diferencias hay entre ellas 2?

- Numericamente son similares pero InfoGini es más fino (usa log).
- Gini gain es más fácil en grandes conjuntos
- Info gain da más valores posibles pero gini es más fácil para cuando tengo muchos valores.

A cada nodo se le aplica alguna de estas tecnicas, y se mantienen para todo el árbol (para que quede consistente)

Si el atributo a predecir es numérico: Se hace tipo un corte y se pregunta si la variable pertenece a un rango. Con estas técnicas, se va cortando el espacio (donde están las muestras).

¿Cuándo hay overfitting?

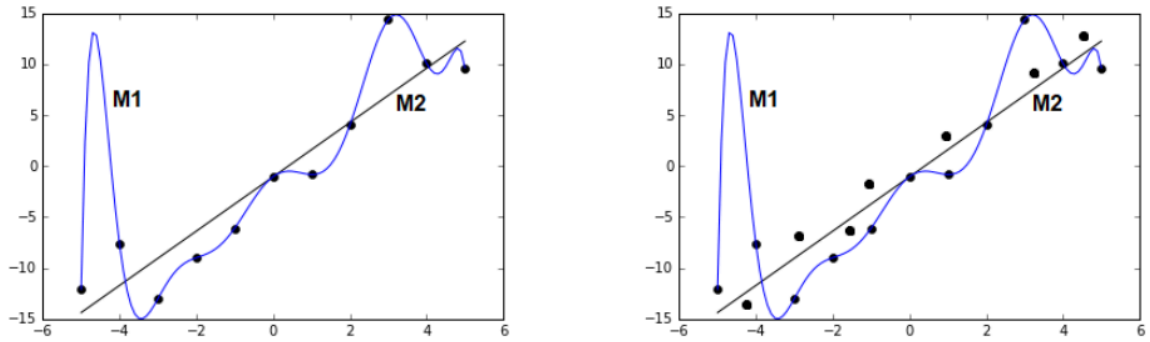
Overfitting es cuando el arbol es muy profundo (demasiadas particiones). nosotros buscamos que el error en la generealizacion sea menor que en el entrenamiento.

Cuando tenemos un nodo por cada evaluación tenemos overfitting.

Medición de overfitting: un modelo M1 sobreajusta si existe un modelo M2 tal que:

- $error_D(M_1) < error_D(M_2)$
- $error_X(M_1) > error_D(M_2)$

D es de **entrenamiento.**
X es un **dato nuevo.**



M1 es mejor en D pero M2 generaliza mejor.

Soluciones al sobreajuste:

- Criterio de parada de profundidad (depende de la cantidad de instancias, es un hiper parámetro que hay que calcular)

- Pruning (poda): dejamos que el algoritmo corra y que genere muchas ramas, y después vemos si sobre los datos separados una vez que cortamos, logramos mejorarlo o no.

- Regla post-pruning: Construir árbol, convertirlo en reglas y optimizarlas según accuracy

Ventajas:

- Interpretabilidad: es clara porque son tomas de decisiones que tomaría una persona, para evitar sesgo.
- Similares al criterio de decisión humano
- Sirve para identificar importancia de variables a partir de cientos de variables
- Varios tipos de inputs

Desventajas:

- Menos preciso que otros modelos.
- Alta varianza. No está bueno que sea así porque es muy volátil.

Si el árbol tiene un solo nodo, va a quedar muy sesgado (problema contrario al sobreajuste) porque solo va a ver una variable. Queda con alto sesgo y baja varianza.