UNIVERSIDAD
COMPLUTENSE
MADRID

Desarrollo de Aplicaciones y Sistemas Inteligentes (DASI)

Clasificación con árboles de decisión

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial UCM

DASI

Árboles de decisión

Árboles de decisión

- Un **árbol de decisión** es un grafo jerárquico que muestra todas las alternativas y resultados posibles
  - A partir de la raíz del árbol, en cada **nodo** se plantea una **pregunta** (sobre el valor de un atributo o característica)
  - Cada **arco** desde el nodo representa una **elección**, posible respuesta (valores posibles en los ejemplos)
  - Cada nodo terminal (**hojas** del árbol) representa un **resultado**, la predicción (clase de ese ejemplo)
- Es un método de **aprendizaje supervisado**:
  - Utiliza un conjunto de ejemplos de entrenamiento etiquetados para construir el árbol de decisión
  - El árbol construido permite clasificar nuevos ejemplos:
    - Colocando el ejemplo nuevo en la raíz y respondiendo las preguntas
    - Con ello desciende por las ramas hasta llegar a un nodo hoja
- En principio el árbol de decisión es **interpretable**, por ejemplo:
  - Qué variables tienen mayor poder discriminante
  - Qué valores de variables están asociados con una clase u otra

Ejemplo: predicción de clicks

Predicción de click o no click para un anuncio de coches autónomos

User gender	Annual income	Have a car	Interested in tech	Click
M	200,000	True	True	1
F	5,000	False	False	0
F	100,000	True	True	1
M	10,000	True	False	0
M	80,000	False	False	0
…	…	…	…	…
…	…	…	…	…

M	120,000	True	True	?
F	70,000	False	True	?

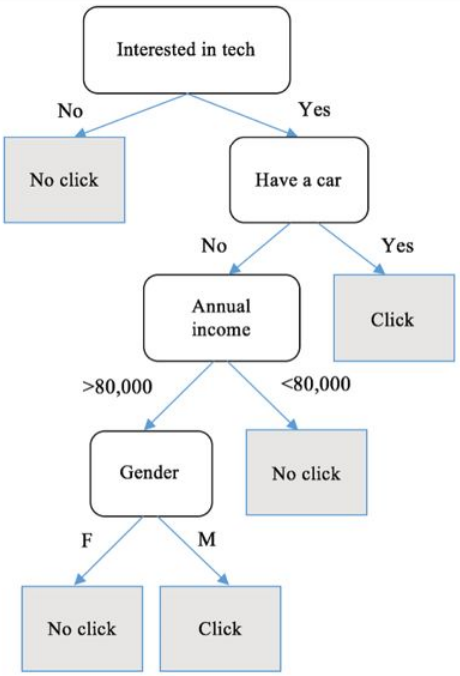


Figure 4.3: Predicting Click/No Click with a trained decision tree, del libro Yuxi (Hayden) Liu: *Python Machine Learning By Example, Third Edition*. Packt Publishing, 2020

Construcción de árboles de decisión

- Proceso inductivo, de la raíz a las hojas:
  - En cada nodo se selecciona la mejor característica (*feature*) y valor para hacer una partición
    - Calcula un criterio de optimalidad para todas las combinaciones de características y valores usando una función de medida
      ⇒ se trata de identificar el que mejor discrimine entre el conjunto de ejemplos
    - Selecciona la combinación más significativa de característica y valor para hacer la partición (la que mejor valor de optimalidad tiene)
    - Genera los nodos hijos en función de la característica seleccionada
  - Divide y vencerás aplicado recursivamente
    - Se dividen los ejemplos de cada nodo en función de la característica elegida
    - El proceso se aplica recursivamente en los nodos hijos
    - El proceso termina cuando:
      - Todos los elementos pertenecen a la misma clase
      - No hay más características que elegir
      - Se ha alcanzado el número mínimo de elementos para nuevos nodos
      - Se ha alcanzado la profundidad máxima del árbol (téngase en cuenta que un árbol demasiado profundo puede generar sobreajuste)
  - En cada nodo terminal (hoja) la clase dominante de sus elementos determina la predicción

Algoritmos de construcción de árboles de decisión

- Iterative Dichotomiser 3 (**ID3**)
  - Hace búsqueda voraz de arriba a abajo seleccionando el mejor atributo para dividir el conjunto de datos en cada interacción, sin backtracking
- **C4.5**
  - Versión mejorada del ID3 que usa backtracking
  - Utiliza como criterio de selección de atributos la ratio de ganancia de información en vez de la tendencia del ID3 a a favorecer la elección de atributos con muchos valores posibles
- Classification and Regression Tree (**CART**)
  - Parecido al C4.5
  - Construye el árbol con particiones binarias
- Chi-squared Automatic Interaction Detector (**CHAID**)
  - Utiliza conceptos estadísticos complejos para determinar el modo óptimo de mezclar variables predictivas para explicar mejor el resultado
  - Común en aplicaciones de marketing

Criterios para medir la calidad de la separación

- **Impureza de Gini**
  - El objetivo de la división en cada nodo es obtener subnodos lo más homogéneos posibles
    - Homogéneos significa que todos los elementos son de la misma clase
  - La impureza de Gini mide la impureza de los nodos como

    $$\text{Impureza de Gini} = 1 - \text{Gini}$$

  - Siendo **Gini** un valor entre 0 y 1, como la suma de los cuadrados de las probabilidades de éxito de cada clase

    $$\text{Gini} = p_1^2 + p_2^2 + \ldots + p_n^2$$

    para n clases
  - Habiendo calculado la impureza de Gini para los subnodos, la impureza de Gini de la división será la impureza ponderada de ambos subnodos de esa división
    - El peso se decide por el número de observaciones de muestras en ambos nodos

  ⇒ Una impureza de Gini baja significa un conjunto de datos más puro

Impureza de Gini - Implementación

- Véase una implementación en

  https://colab.research.google.com/drive/1df97axR1PLpnZf6bZVDPB5NwQaaHli4m?usp=sharing

Impureza de Gini - Ejemplo

User gender	Interested in tech	Click	Group by gender
M	True	1	Group 1
F	False	0	Group 2
F	True	1	Group 2
M	False	0	Group 1
M	False	1	Group 1

#1 split based on gender

User gender	Interested in tech	Click	Group by interest
M	True	1	Group 1
F	False	0	Group 2
F	True	1	Group 1
M	False	0	Group 2
M	False	1	Group 2

#2 split based on interest in tech

$$\#1\ Gini\ Impurity = \frac{3}{5}[1 - (\frac{2}{3}^2 + \frac{1}{3}^2)] + \frac{2}{5}[1 - (\frac{1}{2}^2 + \frac{1}{2}^2)] = 0.467$$

$$\#2\ Gini\ Impurity = \frac{2}{5}[1 - (1^2 + 0^2)] + \frac{3}{5}[1 - (\frac{1}{3}^2 + \frac{2}{3}^2)] = 0.267$$

Sale mejor hacer la participación por interés en tecnología que por género

Criterios para medir la calidad de la separación

- **Ganancia de Información**
  - Mide la mejora de la pureza tras la partición
    ⇒ si con la partición se reduce la **entropía** (medida de la incertidumbre)
- La entropía mide la ausencia de homogeneidad de un conjunto de ejemplos con respecto a su clase
  - Entropía 0 significa homogeneidad total, esto es, todos los elementos son de la misma clase
- La ganancia de información es la diferencia entre la entropía del conjunto original y la de los subconjuntos obtenidos

Ganancia_Información = Entropía(antes) - Entropía(después)
= Entropía(padre) - Entropía(hijos)

Cálculo de la entropía

- La **entropía inicial** de un nodo N (antes de crear la partición) en el que la variable de salida *y* tiene s_i posibles clases ($i=1..p$) se calcula como:

$$E(N) = -\sum_{i=1}^{p} P(s_i) \log_2 P(s_i)$$

  $P(s_i)$ es la probabilidad de que un elemento sea de la clase s_i en el nodo N (se consideran solo las clases que se dan en el nodo N)

- La **entropía final** de un nodo N tras usar una característica A que tiene q valores (a_j):

$$E(N|A) = \sum_{j=1}^{q} P(a_j) \left(-\sum_{i=1}^{p} P(s_i|a_j) \log_2 P(s_i|a_j) \right)$$

  $P(s_i|a_j)$ es la probabilidad de que un elemento del nodo hijo con $A=a_j$ tenga la clase s_i

Cálculo de la ganancia de información

- Para cada característica A, B, C, … se calculará la disminución de la entropía al aplicarse:

```
Ganancia(N, A) = E(N) - E(N|A)
Ganancia(N, B) = E(N) - E(N|B)
Ganancia(N, C) = E(N) - E(N|C)
...
```

- Se seleccionará la característica con la que se consiga mejor ganancia de información

  ⇒ En general, **tanto la impureza de Gini como la ganancia de información miden la impureza ponderada tras una partición**, por lo cual se suele utilizar una combinación de ambos o uno

- Código en: https://colab.research.google.com/drive/1df97axR1PLpnZf6bZVDPB5NwQaaHIi4m?usp=sharing

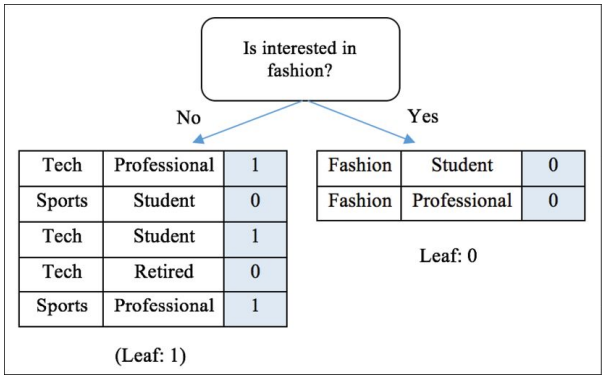
Ejemplo de aplicación del algoritmo CART

- Ejemplo del cap. 4 del libro: Yuxi (Hayden) Liu: *Python Machine Learning By Example, Third Edition*. Packt Publishing, 2020
- Código: https://colab.research.google.com/drive/1eub64vOOTTIjVPV12FJ32G2qX9wxgzQW?usp=sharing

User interest	User occupation	Click
Tech	Professional	1
Fashion	Student	0
Fashion	Professional	0
Sports	Student	0
Tech	Student	1
Tech	Retired	0
Sports	Professional	1

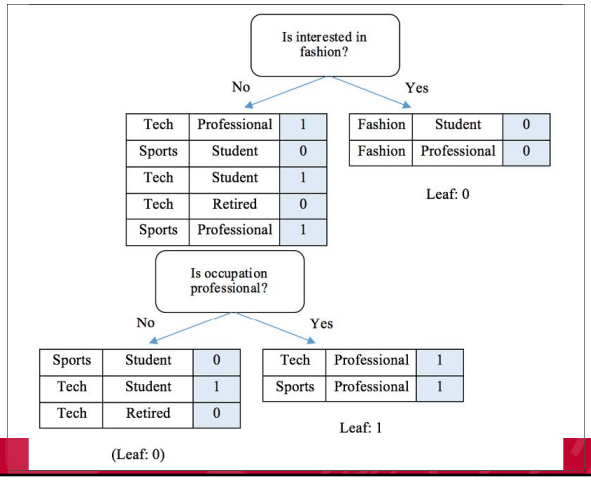
Ejemplo de aplicación del algoritmo CART

```
Gini(interest, tech) = weighted_impurity([[1, 1, 0], [0, 0, 0, 1]]) = 0.405
Gini(interest, Fashion) = weighted_impurity([[0, 0], [1, 0, 1, 0, 1]]) = 0.343
Gini(interest, Sports) = weighted_impurity([[0, 1], [1, 0, 0, 1, 0]]) = 0.486
Gini(occupation, professional) = weighted_impurity([[0, 0, 1, 0], [1, 0, 1]]) = 0.405
Gini(occupation, student) = weighted_impurity([[0, 0, 1, 0], [1, 0, 1]]) = 0.405
Gini(occupation, retired) = weighted_impurity([[1, 0, 0, 0, 1, 1], [1]]) = 0.429
```



Ejemplo de aplicación del algoritmo CART

```
Gini(interest, tech) = weighted_impurity([[0, 1], [1, 1, 0]]) = 0.467
Gini(interest, Sports) = weighted_impurity([[1, 1, 0], [0, 1]]) = 0.467
Gini(occupation, professional) = weighted_impurity([[0, 1, 0], [1, 1]]) = 0.267
Gini(occupation, student) = weighted_impurity([[1, 0, 1], [0, 1]]) = 0.467
Gini(occupation, retired) = weighted_impurity([[1, 0, 1, 1], [0]]) = 0.300
```



Ejemplo de aplicación del algoritmo CART

Código en:
https://colab.research.google.com/drive/1eub64vOOTTljVPV12FJ32G2qX9wxgzQW?usp=sharing

Aplicación de predicción de clicks

- Código:
  https://colab.research.google.com/drive/154ZpCNdk_iwdNxC1rsRZd5YUvIJ8uYNI?usp=sharing
- Conjunto de datos:
  Click-Through Rate Prediction:
  https://www.kaggle.com/c/avazu-ctr-prediction (1 GB aprox)

DASI

Random forest

Random forest

- La técnica de ***bagging*** es una de las que vimos para evitar el sobreajuste
  - Se extraen aleatoriamente diferentes conjuntos de muestras de entrenamiento con reemplazo de los datos de entrenamiento originales
  - Cada conjunto resultante se utiliza para ajustar un modelo de clasificación individual
  - Los resultados de estos modelos entrenados por separado se combinan mediante un voto mayoritario para tomar la decisión final.

Random forest

- El *bagging* reduce la elevada varianza que sufre un modelo de un solo árbol de decisión
  - Sin embargo, cuando una o más características son indicadores importantes, los árboles individuales se construyen en gran medida basándose en estas características y, como resultado, se vuelven altamente correlacionados ⇒ La agregación de varios árboles correlacionados no supondrá una gran diferencia
  - Para forzar que cada árbol no esté correlacionado, el bosque aleatorio sólo tiene en cuenta un subconjunto aleatorio de características cuando busca el mejor punto de división en cada nodo
    ⇒ Los árboles individuales se entrenan ahora basándose en **diferentes conjuntos secuenciales de características**, lo que garantiza una mayor diversidad y un mejor rendimiento
- El *random forest* es una variante del modelo de *bagging* de árboles con un *bagging* adicional sobre las características

Random forest

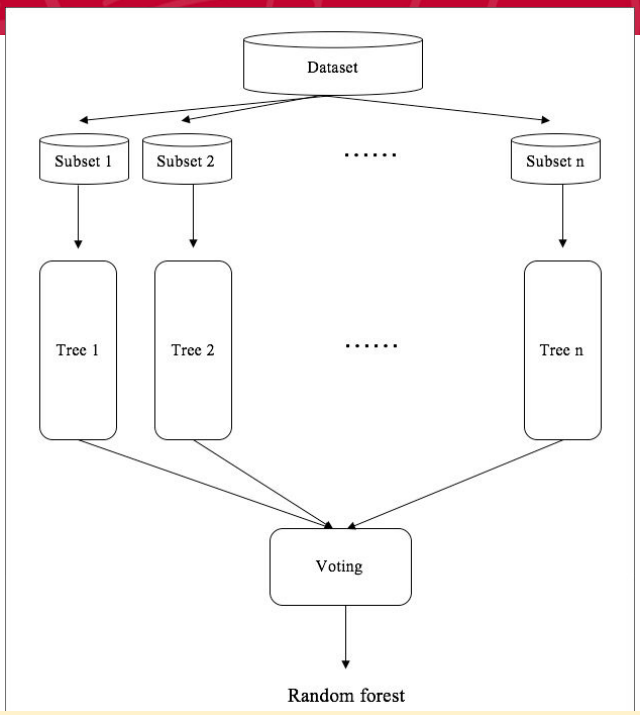


Figure 4.14: The random forest workflow, del libro Yuxi (Hayden) Liu: *Python Machine Learning By Example, Third Edition*. Packt Publishing, 2020

DASI

Bibliografía

Bibliografía

- Sridhar Alla & Suman Kalyan Adari: *Beginning MLOps with MLFlow. Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure*. Apress, 2021.
- Yuxi (Hayden) Liu: *Python Machine Learning By Example, Third Edition*. Packt Publishing, 2020.
- Documentación en línea de las distintas herramientas