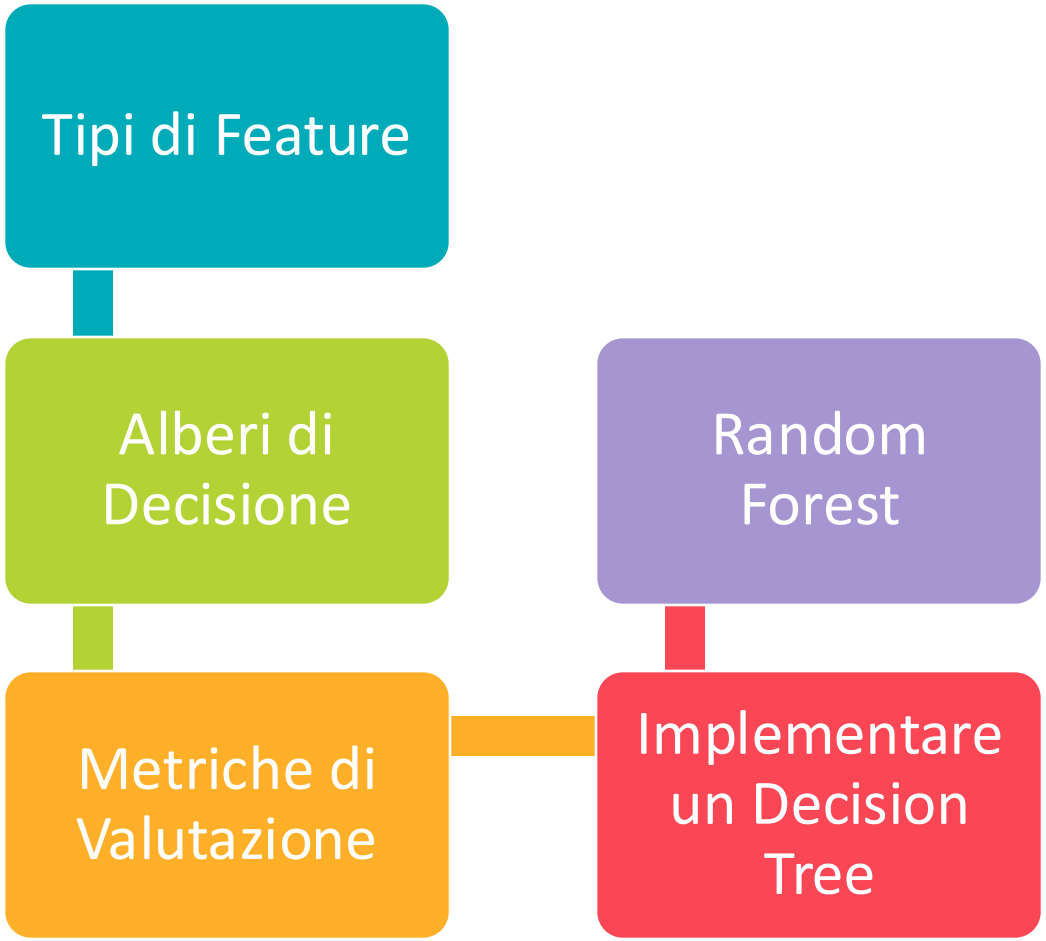
Machine Learning

Algoritmi Tree-Based

Prof. Giuseppe Polese Prof.ssa Loredana Caruccio

Outline


Tipi di Feature
Alberi di Decisione
Random Forest
Metriche di Valutazione
Implementare un Decision Tree

Tipi di Feature

Features (1)

- Ogni dataset può contenere varie tipologie di features:
  - **Categoriche**
    - Dette anche ***qualitative***
    - Rappresentano caratteristiche, gruppi distinti e un numero contato di opzioni
    - Le caratteristiche categoriche possono avere o meno un ordine logico

id	color
1	red
2	blue
3	green
4	blue

Features (2)

- Ogni dataset può contenere varie tipologie di features:
  - **Numeriche**
    - Dette anche ***quantitative***
    - Hanno un significato matematico
    - Sono ordinabili

Case	Attributes		
	Weight	Length	Height
1	0.8	0.3	7.2
2	0.8	1.1	7.2
3	0.8	1.1	10.2
4	1.2	0.3	10.2
5	1.2	2.3	10.2

Features (3)

- Le **categoriche** possono anche assumere **valori numerici**
  - Ad esempio,
    - 1 a 12 possono rappresentare i **mesi** dell'anno
    - 1 e 0 possono indicare **maschio** e **femmina**.

Tuttavia, questi valori non hanno implicazioni matematiche.

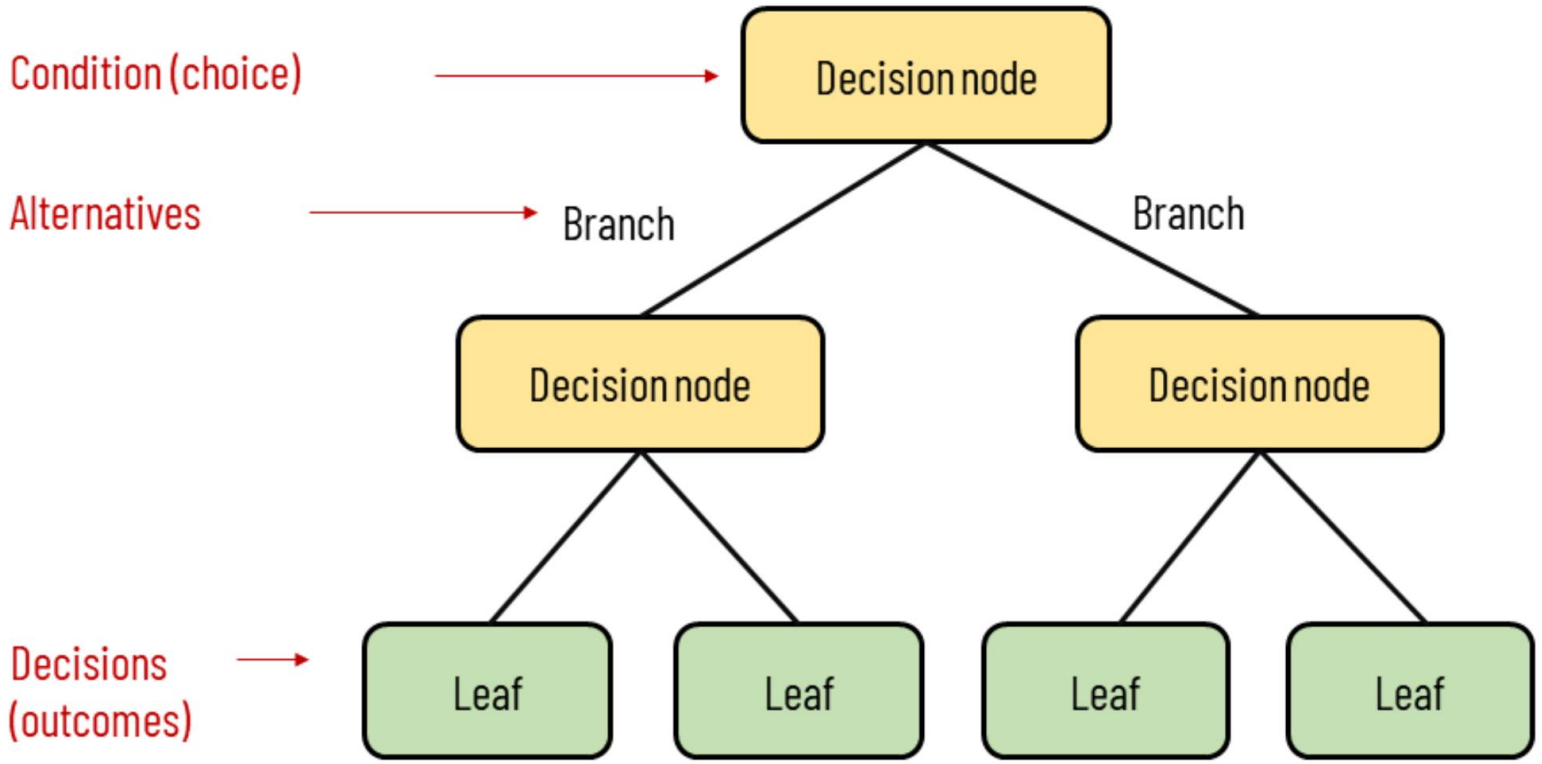
Date	Value	Year	Month
2022-03-28	5.48	2022	02
2021-06-02	6.80	2021	12
2021-05-12	3.20	2021	01
2020-06-30	4.76	2020	04
2022-05-31	3.23	2022	06
2020-01-18	5.62	2020	03

Alberi di Decisione

2

Alberi di Decisione

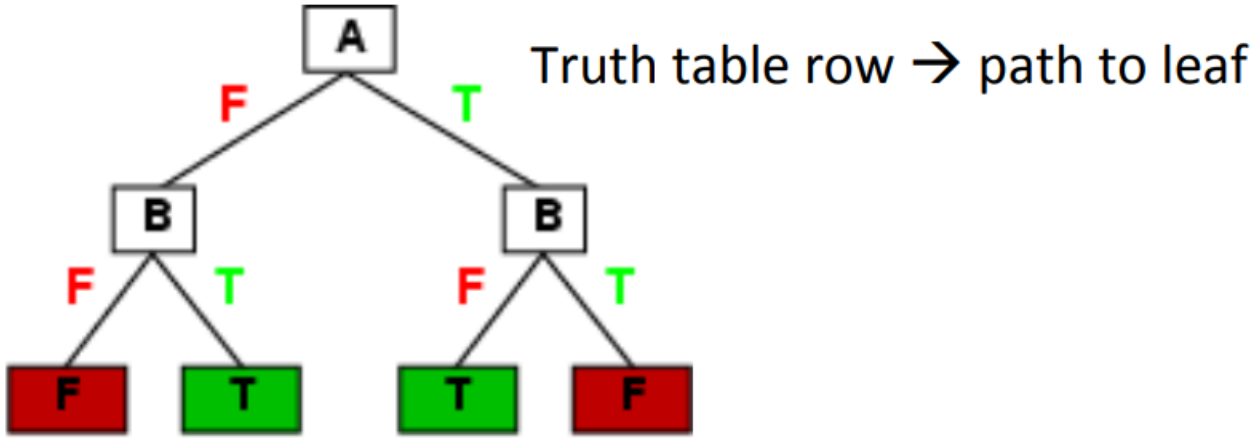
- Un **albero decisionale** è un grafo ad albero, cioè un diagramma sequenziale che illustra **tutte** le **possibili alternative decisionali** e i risultati corrispondenti.



Alberi di Decisione

- Partendo dalla radice dell'albero, ogni nodo interno rappresenta la base su cui viene presa una decisione.
- Ogni ramo di un nodo rappresenta il modo in cui una scelta può portare ai nodi successivi.
- Infine, ogni nodo terminale, la foglia, rappresenta il risultato prodotto.

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Alberi di Decisione

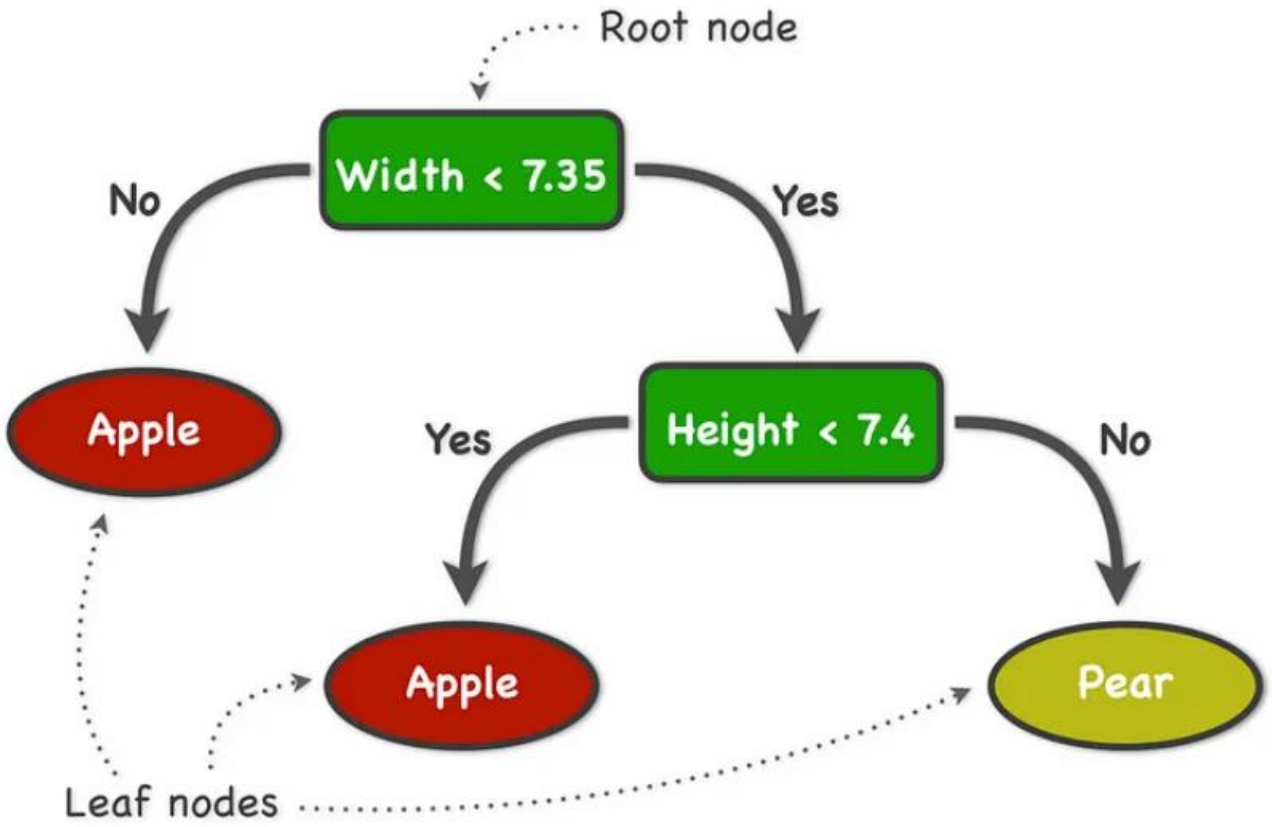
- Un classificatore **Decision Tree** opera sotto forma di albero decisionale.
  1. Esso mappa le osservazioni in assegnazioni di classe
     - Simboleggiate come nodi **foglia**
  2. Attraverso una serie di test
     - Rappresentati come **nodi interni**
  3. Sulla base dei valori delle caratteristiche e sulle condizioni corrispondenti
     - Rappresentate come **rami**

Alberi di Decisione

- In ogni nodo viene posta una domanda relativa ai valori e alle caratteristiche di una caratteristica;
- a seconda della risposta alla domanda, le osservazioni vengono suddivise in sottoinsiemi.
- Vengono condotti test sequenziali finché non si giunge a una conclusione sull'etichetta di destinazione delle osservazioni.
- I percorsi dalla radice alle fogli finali rappresentano il processo decisionale e le regole di classificazione.

Esempio: Alberi di Decisione

Width	Height	Fruit
7.1	7.3	Apple
7.9	7.5	Apple
7.4	7.0	Apple
8.2	7.3	Apple
7.6	6.9	Apple
7.8	8.0	Apple
7.0	7.5	Pear
7.1	7.9	Pear
6.8	8.0	Pear
6.6	7.7	Pear
7.3	8.2	Pear
7.2	7.9	Pear

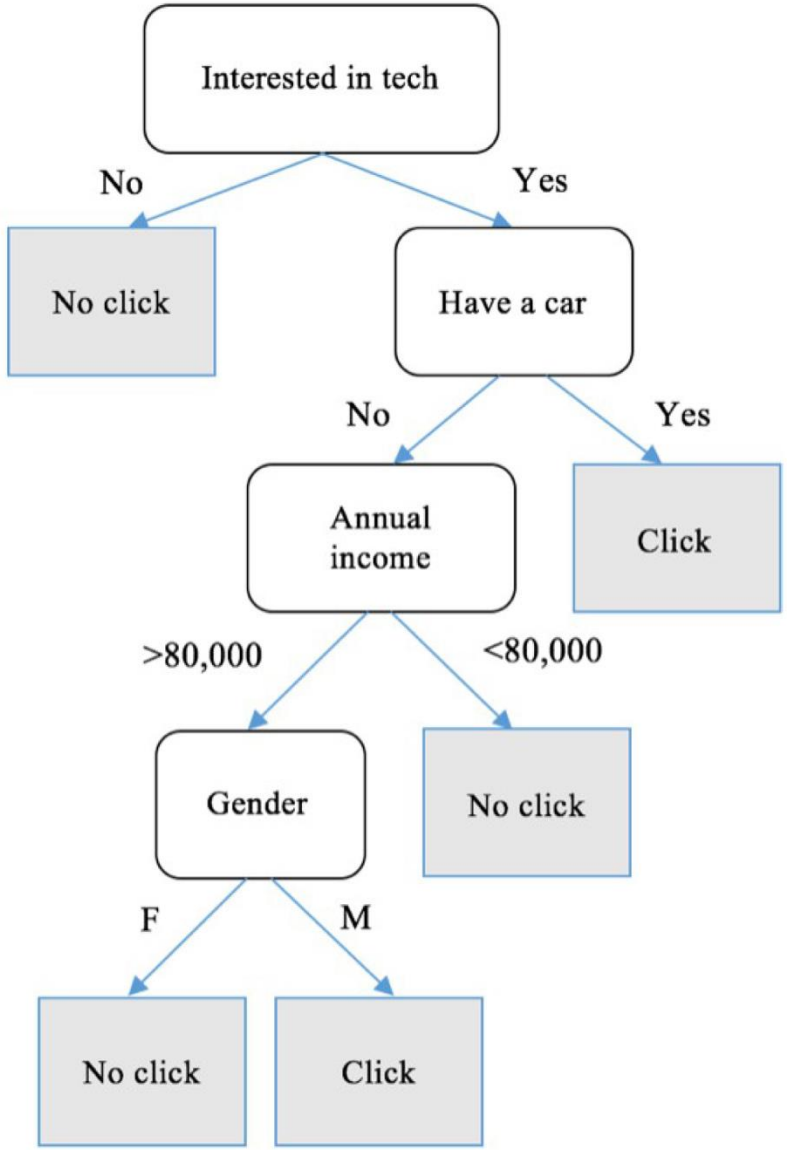


Esempio: Alberi di Decisione

- In uno scenario in cui si vuole prevedere il clic o il non clic su un annuncio pubblicitario di un'auto a guida autonoma

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	?

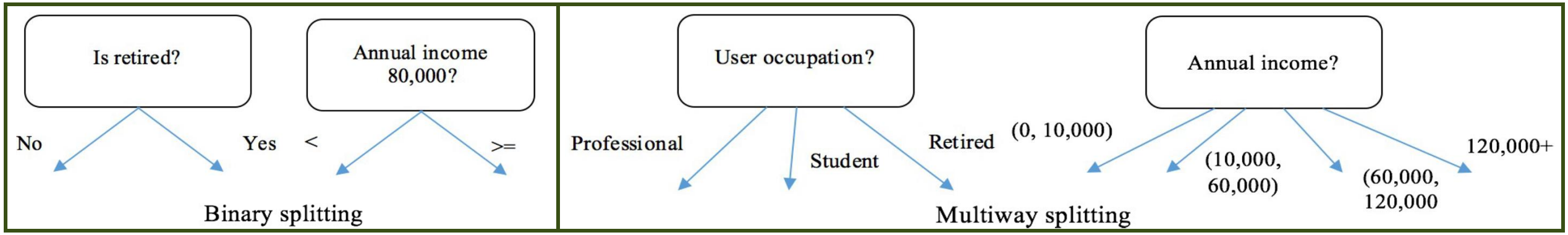


Costruzione di un Albero di Decisione (1)

- Un albero decisionale viene costruito suddividendo i campioni di addestramento in sottoinsiemi.
- Il processo è ripetuto in modo ricorsivo su ogni sottoinsieme.
- Per ogni partizione di un nodo, viene condotto un test di condizione basato sul valore di una caratteristica del sottoinsieme

Costruzione di un Albero di Decisione (3)

- Quando il sottoinsieme condivide la stessa etichetta di classe o quando nessuna ulteriore suddivisione può migliorare la purezza della classe di questo sottoinsieme, il partizionamento ricorsivo termina.
  - Suddivisione per caratteristica (Numerica o Categorica):
    - Con *n* valori diversi, esistono *n* modi diversi di suddivisione

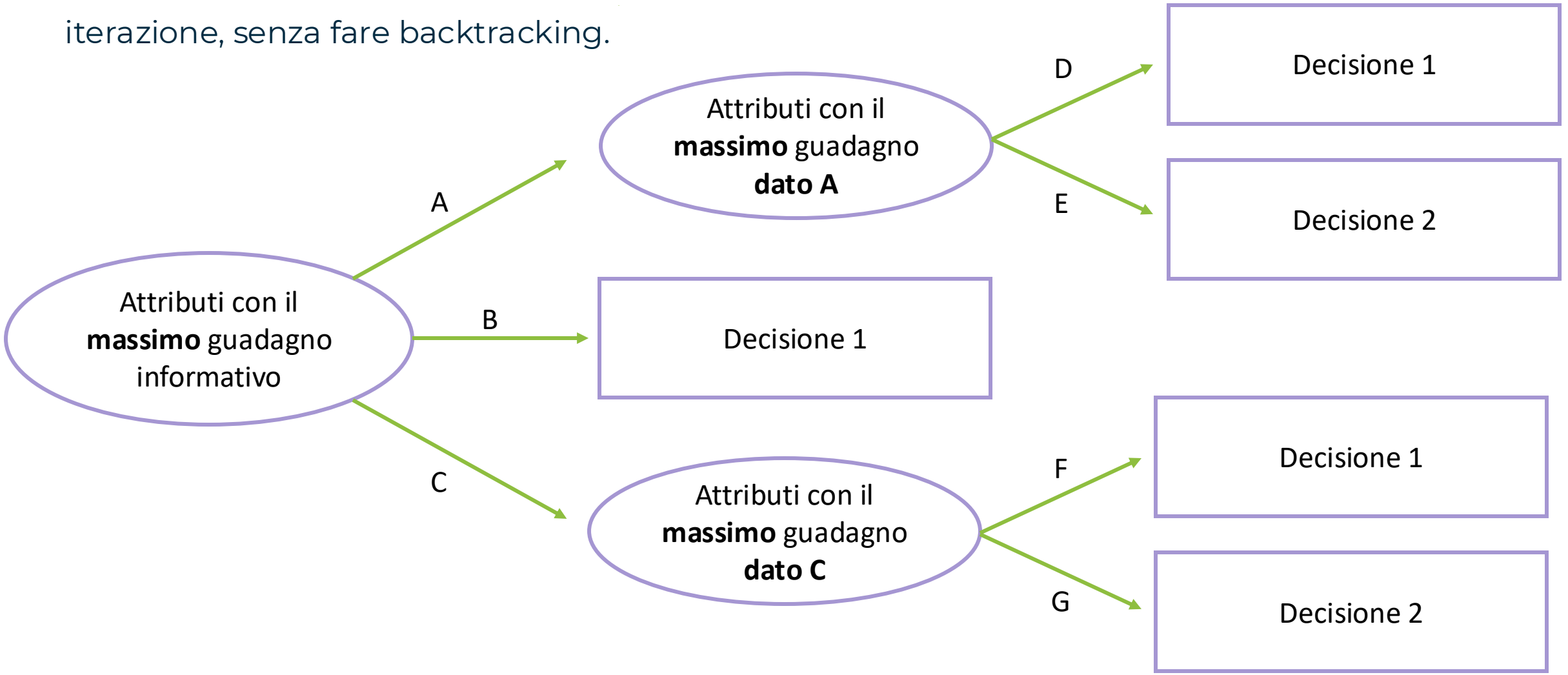


Algoritmi per Alberi di Decisione

- Sono stati sviluppati molti algoritmi per costruire in modo efficiente una decisione accurata
  - ***Iterative Dichotomiser 3 (ID3)***
  - ***C4.5***
  - ***Classification and Regression Tree (CART):***
  - ***Chi-squared Automatic Interaction Detector (CHAID)***
- L'idea di base di questi algoritmi è quella di far crescere l'albero in modo greedy effettuando una serie di ottimizzazioni locali quando si sceglie la caratteristica più significativa da utilizzare per partizionare i dati

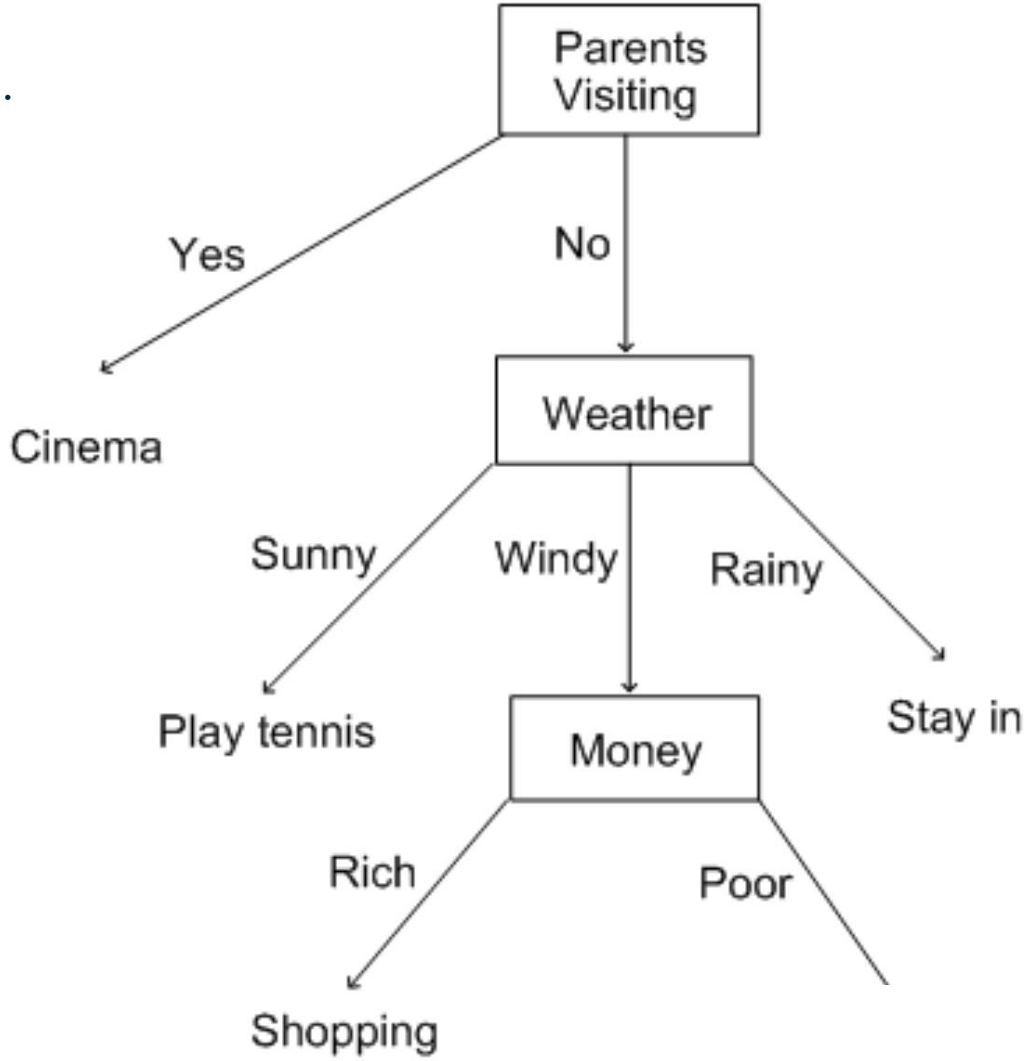
Iterative Dichotomiser 3 (ID3)

- Ricerca **greedy top-down**. Seleziona il miglior attributo su cui dividere il set di dati a ogni iterazione, senza fare backtracking.



C4.5

- Versione migliorata di ID3 che introduce il backtracking.
  - Attraversa l'albero costruito e sostituisce i rami con i nodi foglia se la purezza è migliorata.



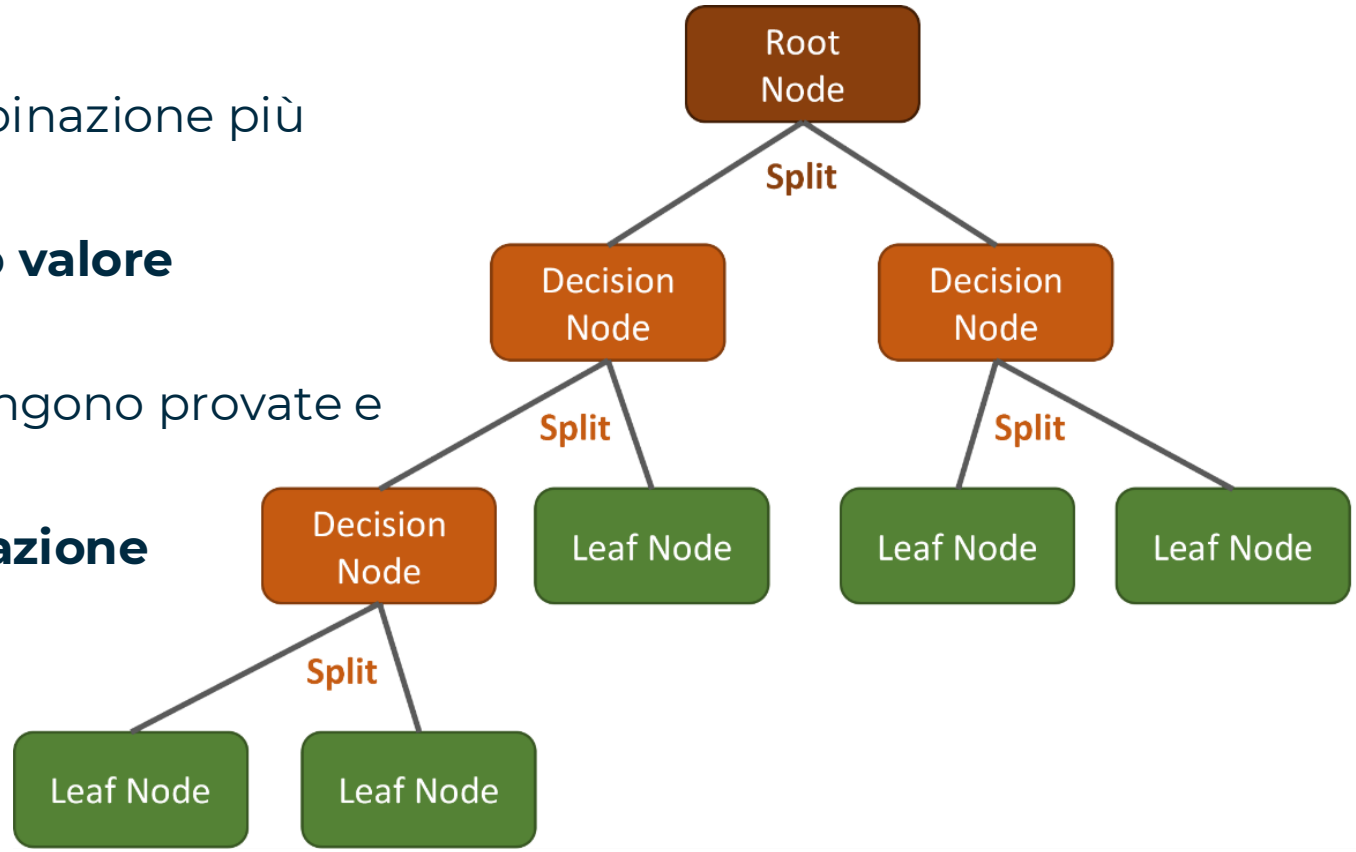
Chi-squared Automatic Interaction Detector (CHAID)

- Questo algoritmo è spesso utilizzato nel marketing diretto, per la selezionare gruppi di consumatori per prevedere come **le loro risposte ad alcune variabili influenzano altre variabili**, sebbene altre prime applicazioni siano state nei campi della ricerca medica e psichiatrica.

- Implica concetti statistici complessi, ma fondamentalmente determina il modo ottimale di unire le variabili predittive per spiegare al meglio il risultato.

Classification and Regression Tree (CART) (1)

- Costruzione di soli alberi binari.
  - Fa crescere l'albero attraverso regole **if-else**, applicando **split binari**
- L'algoritmo cerca in modo **greedy** la combinazione più significativa di una **caratteristica** e del suo **valore**
- Tutte le diverse **combinazioni possibili** vengono provate e testate utilizzando una **funzione di misurazione**



Classification and Regression Tree (CART) (2)

- L'algoritmo divide il set di dati come segue:
  - I campioni con la caratteristica di questo valore (per una caratteristica categorica) o un valore maggiore (per una caratteristica numerica) diventano il figlio di destra.
  - I campioni rimanenti diventano il figlio sinistro
- Questo processo di suddivisione si ripete e divide ricorsivamente i campioni in ingresso in due sottogruppi

Criteri di Arresto (1)

- Il processo di suddivisione si ferma a un sottogruppo in cui è soddisfatto uno dei due criteri:
- **Numero minimo di campioni**
  - Quando il numero di campioni non è superiore al numero minimo di campioni necessari per un'ulteriore suddivisione, il partizionamento si interrompe per evitare che l'albero si adatti eccessivamente all'insieme di allenamento.
- **Profondità massima dell'albero**
  - Un nodo smette di crescere quando la sua profondità, definita come il numero di partizioni che avvengono dall'alto verso il basso, partendo dal nodo radice e terminando in un nodo foglia, non è inferiore alla profondità massima dell'albero.
  - Gli alberi più profondi sono più specifici per il set di addestramento e possono portare a un **overfitting**.

Criteri di Arresto (2)

- Un nodo senza rami diventa una foglia e la classe dominante dei campioni in questo nodo rappresenterà la previsione.
- Una volta terminati tutti i processi di divisione, l'albero viene costruito e rappresentato con le etichette assegnate ai nodi terminali e i punti di divisione (caratteristica + valore) in tutti i nodi interni superiori.

Metriche di Valutazione

3

Metriche per misurare uno split

- Quando si seleziona la migliore combinazione di una caratteristica e di un valore come punto di divisione, si possono utilizzare due criteri per misurare la qualità della separazione:
  - **Impurità di Gini**
  - **Guadagno di informazione (Information Gain)**

Impurità di Gini (1)

- Misura il tasso di impurità della distribuzione delle classi dei punti dati, o il tasso di mescolanza delle classi.
- Per un insieme di dati con **K** classi, supponiamo che i dati della classe **k (1 ≤ k ≤ K)** occupino una frazione **fk (0 ≤ fk ≤ 1)** dell'intero insieme di dati
- **Gini Impurity** di questo insieme di dati si scrive:

$$Gini\ Impurity = 1 - \Sigma_{k=1}^{K} {f_k}^2$$

Impurità di Gini (2)

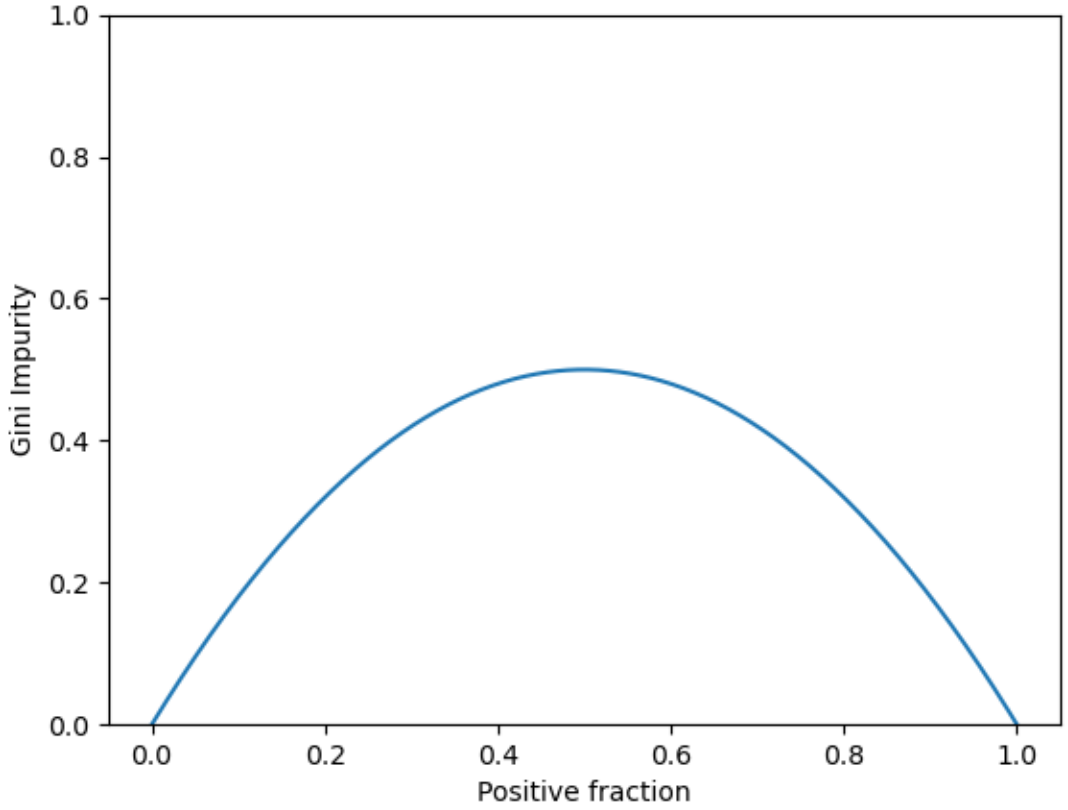
- Una Gini Impurity più bassa indica un set di dati più puro.
- Ad esempio, se il set di dati contiene una sola classe, ad esempio, la frazione di questa classe è 1 e quella delle altre è 0
  - la sua **Gini Impurity** diventa $1 - (1^2 + 0^2) = 0$.
- In un altro esempio, un set di dati registra un gran numero di lanci di monete, e le teste e le code occupano ciascuna la metà dei campioni.
  - L'impurità di **Gini** è $1 - (0{,}5^2 + 0{,}5^2) = 1 - (0{,}25 + 0{,}25) = 0{,}5$.

$$Gini\ Impurity = 1 - \Sigma_{k=1}^{K} {f_k}^2$$

Python code: Gini_1.py

Impurità di Gini (3)

- Come si può notare, nei casi binari:
  - Se la frazione positiva è del 50%, l'impurità sarà massima, pari a 0,5
  - Se la frazione positiva è del 100% o dello 0%, raggiungerà impurità 0



Python code: Gini_1.py

Impurità di Gini (4)

- Per valutare la qualità di una suddivisione, è sufficiente sommare la Gini Impurity di tutti i sottogruppi risultanti, combinando le proporzioni di ciascun sottogruppo come corrispondenti fattori di peso.
- Date le etichette di un set di dati, possiamo implementare la funzione di calcolo dell'impurità di Gini

Python code: Gini_2.py

Impurità di Gini (5)

- Per valutare la qualità di una suddivisione, è sufficiente sommare la Gini Impurity di tutti i sottogruppi risultanti, combinando le proporzioni di ciascun sottogruppo come corrispondenti fattori di peso.
- Anche in questo caso, quanto più piccola è la somma ponderata della Gini Impurity, tanto migliore è la suddivisione.

Python code: Gini_2.py

Impurità di Gini (6)

- Si veda il seguente esempio di pubblicità di auto a guida autonoma. In questo caso, abbiamo suddiviso i dati in base al sesso e all'interesse per la tecnologia dell'utente

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

Impurità di Gini (7)

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

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

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

La suddivisione dei dati in base all'interesse dell'utente per la tecnologia è una strategia migliore rispetto al genere.

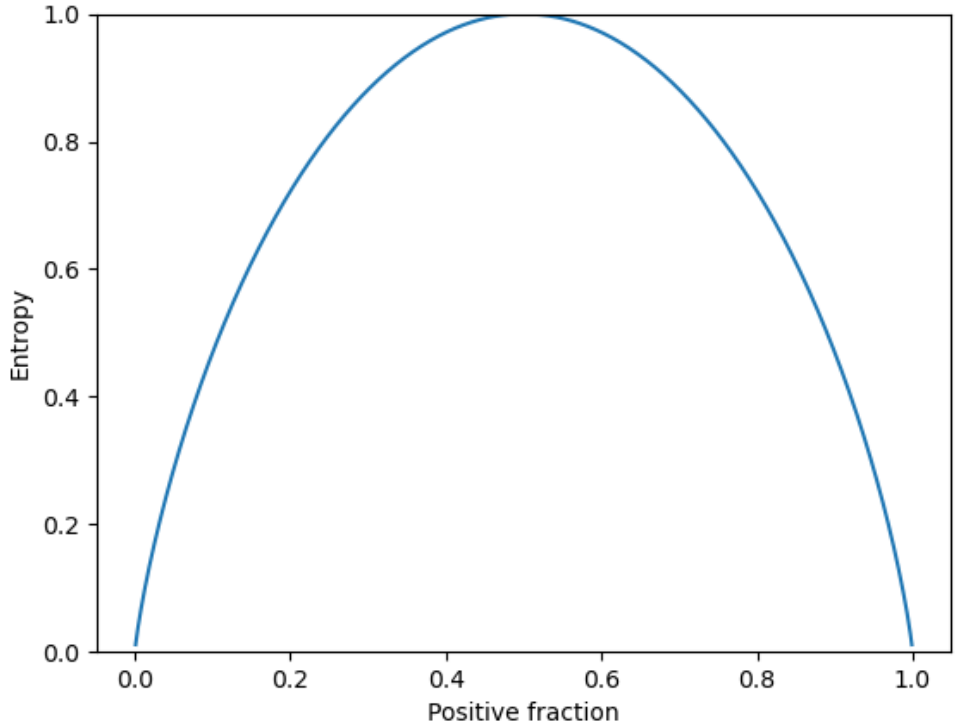
Information Gain

- Misura il miglioramento della purezza dopo la suddivisione o, in altre parole, la riduzione dell'incertezza dovuta alla suddivisione.
- Un information gain più elevato implica una migliore suddivisione.
- L'information gain di uno split si ottiene confrontando **l'entropia** prima e dopo lo split.
  - L'entropia misura l'incertezza di un insieme di dati.
    - Dato un insieme di dati con K classi, e f_k ($0 \leq f_k \leq 1$) la frazione di dati della classe k ($1 \leq k \leq K$), l'entropia dell'insieme di dati è definita come segue:

$$Entropy = -\Sigma_{k=1}^{K} f_k * log_2 f_k$$

Entropia (1)

- Un'entropia più bassa implica un insieme di dati più puro e con meno ambiguità.
- In un caso perfetto, in cui il set di dati contiene una sola classe, l'entropia è $-(1 * log_2 1 + 0) = 0$.
- Nell'esempio del lancio di una moneta, l'entropia diventa $-(0{,}5 * log_2 0{,}5 + 0{,}5 * log_2 0{,}5) = 1$
- Come si può notare, nei casi binari
  - Se la frazione positiva è pari al 50%, l'entropia sarà massima, pari a 1
  - Se la frazione positiva è pari al 100% o allo 0%, raggiungerà un'entropia pari a 0.



Python code: Entropia_1.py

Entropia (2)

- Possiamo esaminare come **l'information gain** misuri la riduzione dell'incertezza dopo lo split, definita come la differenza di entropia prima dello split (genitore) e dopo lo split (figli):

$$InformationGain = Entropy(before) - Entropy(after) = Entropy(parent) - Entropy(children)$$

- L'entropia dopo uno split è calcolata come somma ponderata dell'entropia di ciascun figlio, che è simile all'impurità di Gini ponderata.

Python code: Entropia_2.py

Entropia (3)

- Durante il processo di costruzione di un nodo di un albero, il nostro obiettivo è cercare il punto di divisione in cui si ottiene il massimo guadagno di informazioni.
- Poiché l'entropia del nodo genitore è invariata, è sufficiente misurare l'entropia dei figli risultanti da una divisione.

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

Entropia (4)

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

$$\#1\ entropy = \frac{3}{5}(-(\frac{2}{3} * log_2 \frac{2}{3} + \frac{1}{3} * log_2 \frac{1}{3})] + \frac{2}{5}(-(\frac{1}{2} * log_2 \frac{1}{2} + \frac{1}{2} * log_2 \frac{1}{2})] = 0.951$$

$$\#2\ entropy = \frac{2}{5}(-(1 * log_2 1 + 0)] + \frac{3}{5}(-(\frac{1}{3} * log_2 \frac{1}{3} + \frac{2}{3} * log_2 \frac{2}{3})] = 0.551$$

Calcolo Information Gain

$$\#1\ entropy = \frac{3}{5}(-(\frac{2}{3} * log_2\frac{2}{3} + \frac{1}{3} * log_2\frac{1}{3})] + \frac{2}{5}(-(\frac{1}{2} * log_2\frac{1}{2} + \frac{1}{2} * log_2\frac{1}{2})] = 0.951$$

$$\#2\ entropy = \frac{2}{5}(-(1 * log_2 1 + 0)] + \frac{3}{5}(-(\frac{1}{3} * log_2\frac{1}{3} + \frac{2}{3} * log_2\frac{2}{3})] = 0.551$$

$$Entropy\ before = -(\frac{3}{5} * \log_2\frac{3}{5} + \frac{2}{5} * \log_2\frac{2}{5}) = 0.971$$

$$\#1\ InformationGain = 0.971 - 0.951 = 0.020$$

$$\#2\ InformationGain = 0.971 - 0.551 = 0.420$$

- La seconda suddivisione è da preferire, il che rappresenta la stessa conclusione del criterio dell'impurità di Gini.

Python code: 5. Calcolo_Metriche.py

Implementare un Decision Tree

3

Implementazione di un albero decisionale

- Per cominciare, decidiamo il primo punto di divisione, la radice, provando tutti i valori possibili per ciascuna delle due caratteristiche.

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

Python code: 6. Implementazione_DC_1.py

Implementazione di un albero decisionale

- In questo caso, se si suddivide in base all'interesse dell'utente per la **tecnologia**, si hanno il 1°, il 5° e il 6° campione per un gruppo e i restanti campioni per un altro gruppo.
- Allora le classi del primo gruppo sono **[1, 1, 0]** e le classi del secondo gruppo sono **[0, 0, 0, 1]**

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

Python code: 6. Implementazione_DC_1.py

Implementazione di un albero decisionale

- In questo caso, se si suddivide in base all'interesse dell'utente per la **moda**, si hanno il secondo e il terzo campione per un gruppo e i restanti campioni per un altro gruppo.
- Allora le classi del primo gruppo sono **[0, 0]** e le classi del secondo gruppo sono **[1, 0, 1, 0, 1]**:

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

Python code: 6. Implementazione_DC_1.py

Implementazione di un albero decisionale

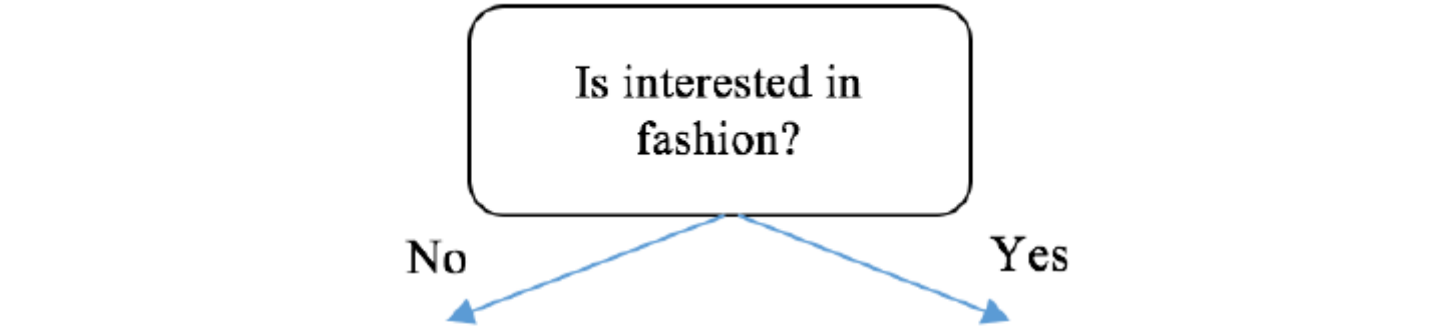
- Il criterio di split da utilizzare per la radice riguarderà la feature interesse dell'utente con il valore di **moda**, poiché questa combinazione raggiunge l'impurità di Gini più bassa e l'information gain più elevato.

```
Gini split Tech: 0.4048

Gini split Fashion: 0.3333

Gini split Sport: 0.4857

Gini split Professional: 0.4048

Gini split Student: 0.4048

Gini split Retired: 0.4286
```

Python code: 6. Implementazione_DC_1.py

Costruzioni di un albero decisionale

- Se ci accontentiamo di un albero profondo un solo livello, possiamo fermarci qui assegnando la giusta l'etichetta 0 del ramo sinistro e l'etichetta 1 del ramo sinistro come classe maggiormente rappresentata.



Tech	Professional	1
Sports	Student	0
Tech	Student	1
Tech	Retired	0
Sports	Professional	1

(Leaf: 1)

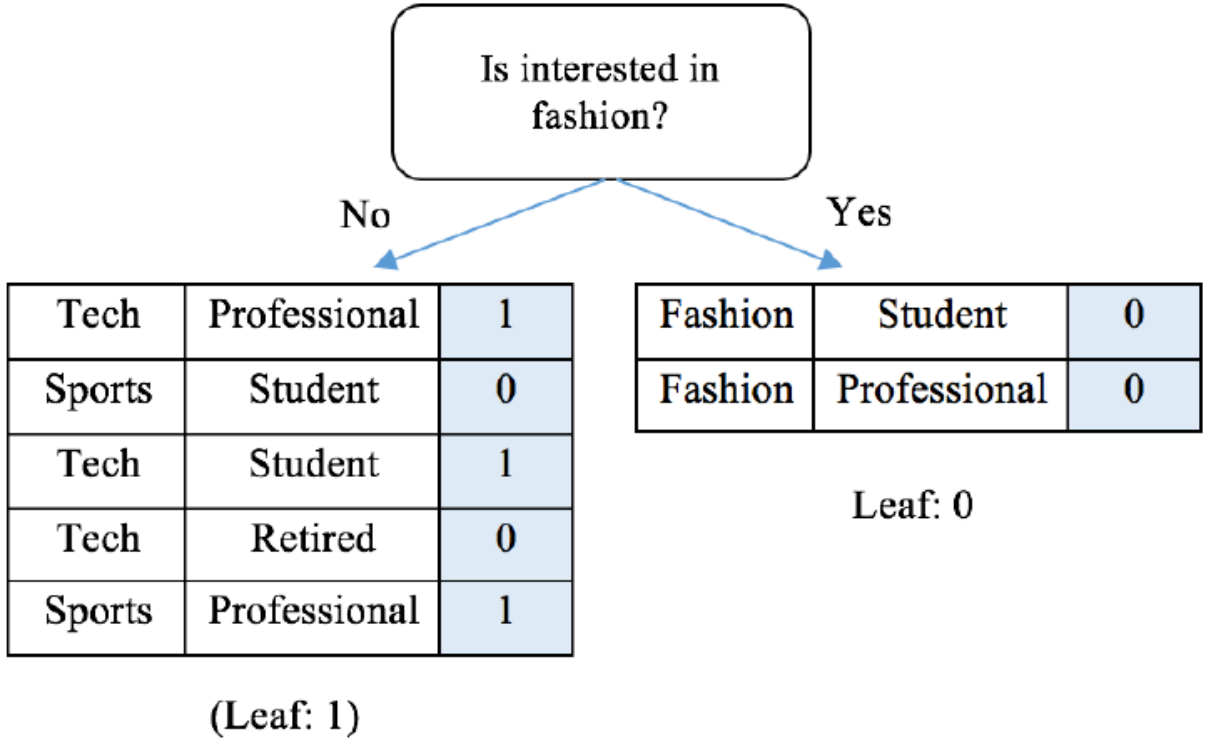
Fashion	Student	0
Fashion	Professional	0

Leaf: 0

Implementazione di un albero decisionale

- In alternativa, si può andare oltre, costruendo

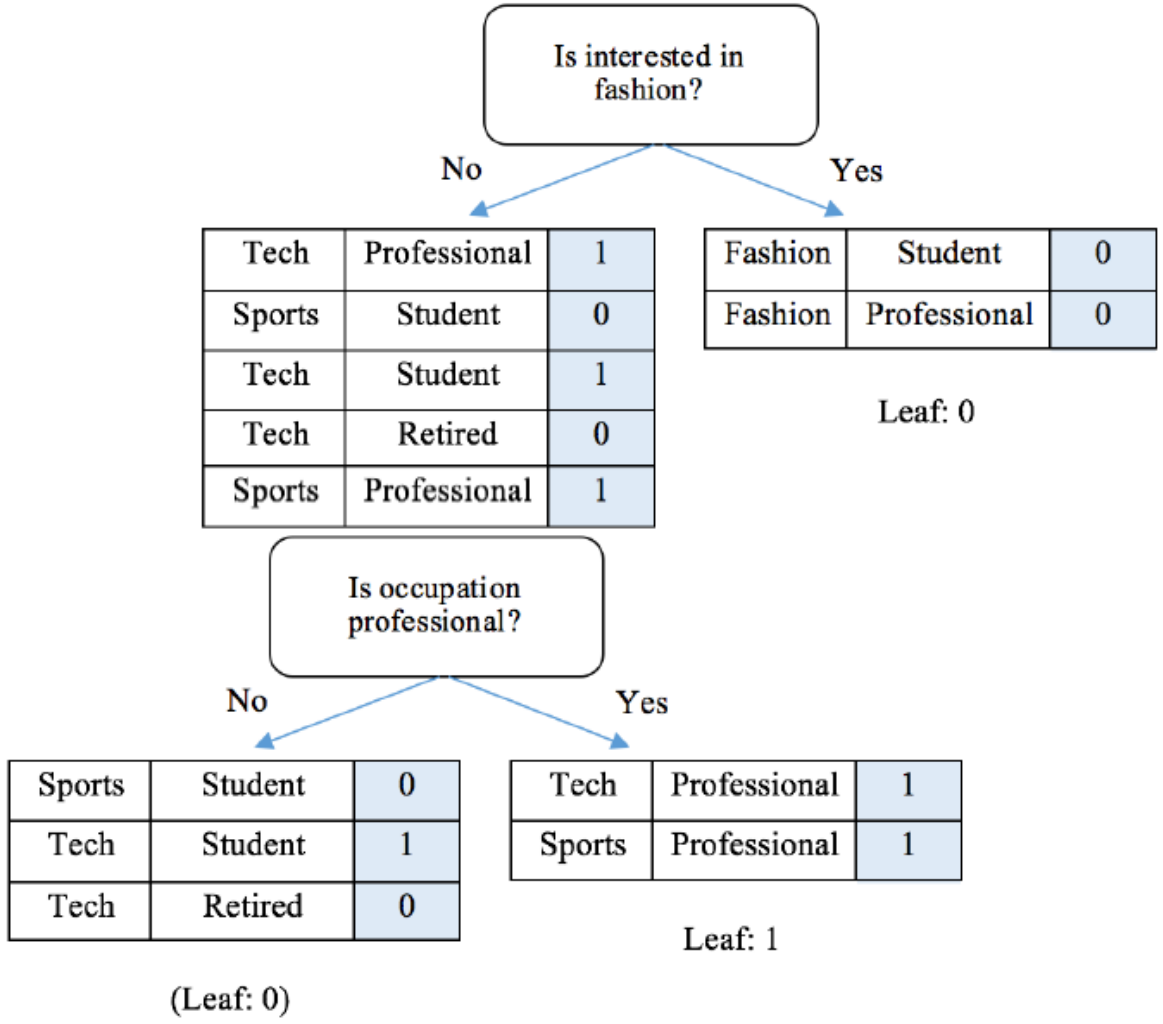
  il secondo livello a partire dal ramo sinistro



(il ramo destro non può essere ulteriormente diviso)

Python code: 7. Implementazione_DC_2.py

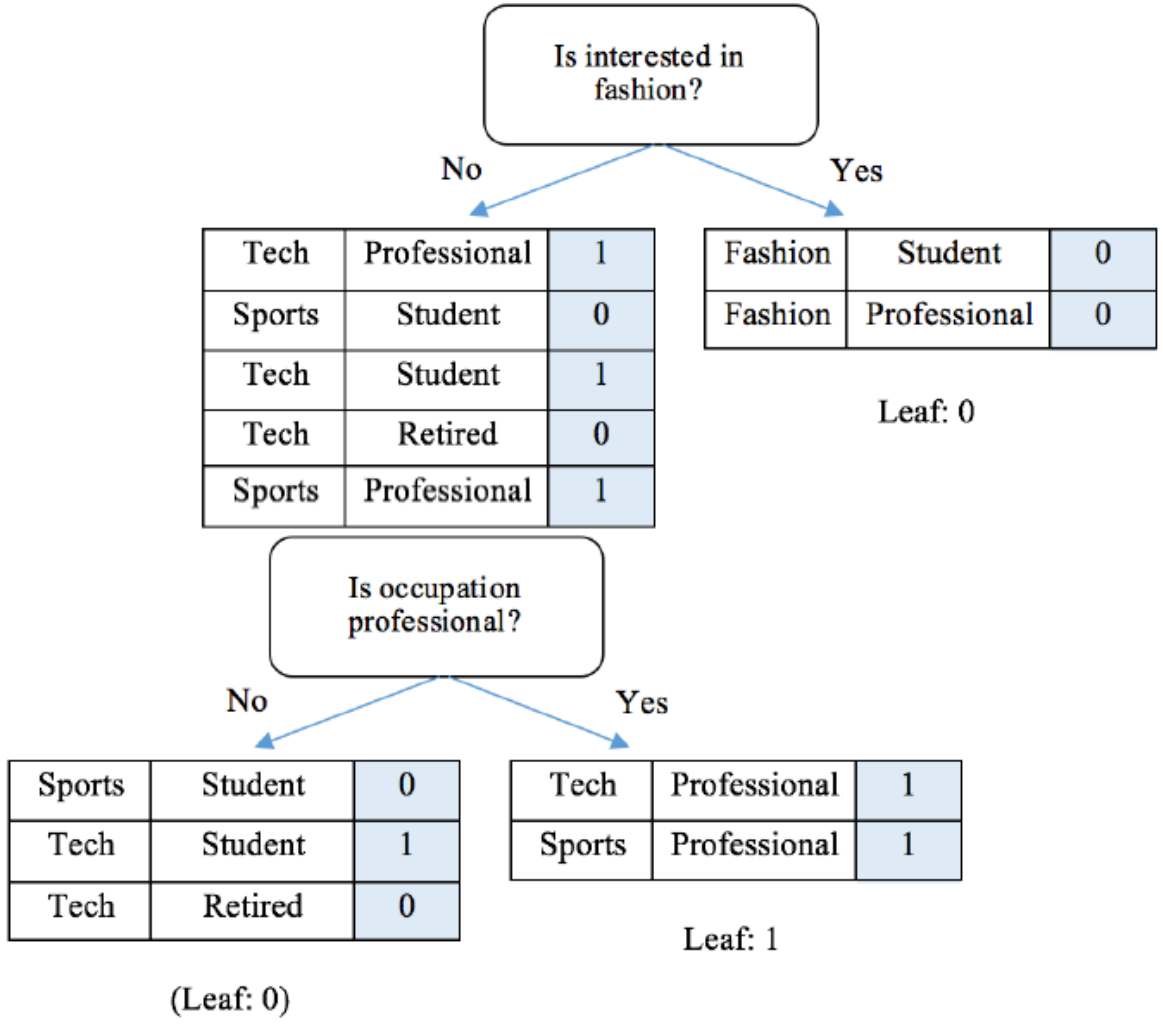
Implementazione di un albero decisionale




```
Gini split (Interest,Tech): 0.5000

Gini split (Interest,Sport): 0.4667

Gini split (Occupation,Professional): 0.2667

Gini split (Occupation,Student): 0.4667

Gini split (Occupation,Retired): 0.3000
```

Python code: 7. Implementazione_DC_2.py

Implementazione di un albero decisionale



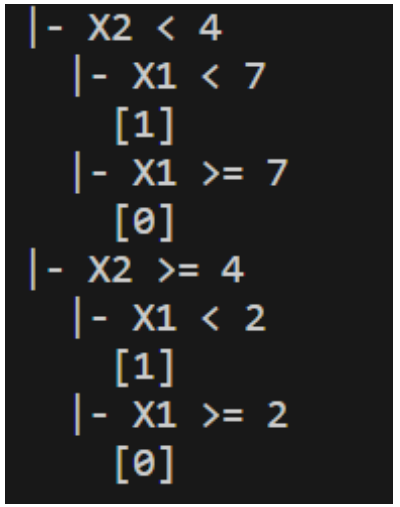

```
|- X1 is not fashion
  |- X2 is not professional
    [0]
  |- X2 is professional
    [1]
|- X1 is fashion
  [0]
```

Python code: 8. Implementazione_DC_3.py

Implementazione di un albero decisionale

- Esempio con feature numeriche

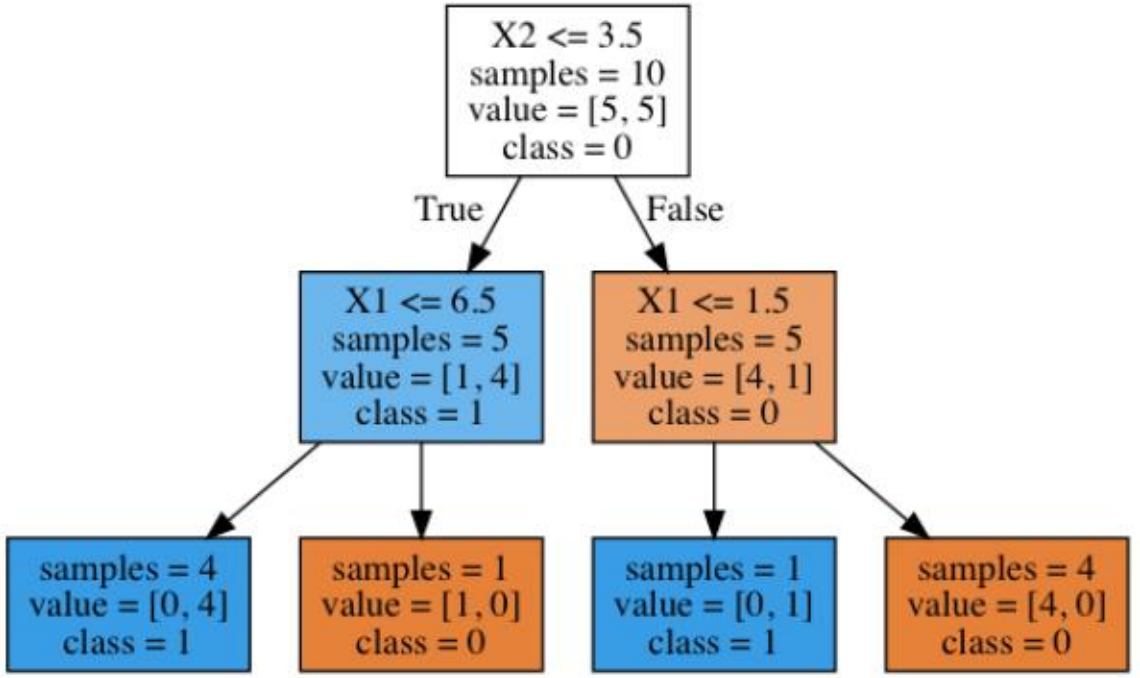
```
X_train_n = [[6, 7], [2, 4], [7, 2], [3, 6], [4, 7],
[5, 2], [1, 6], [2, 0], [6, 3], [4, 1]]

y_train_n = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
```



Python code: 9. Implementazione_DC_4.py

Albero Decisionale con scikitlearn

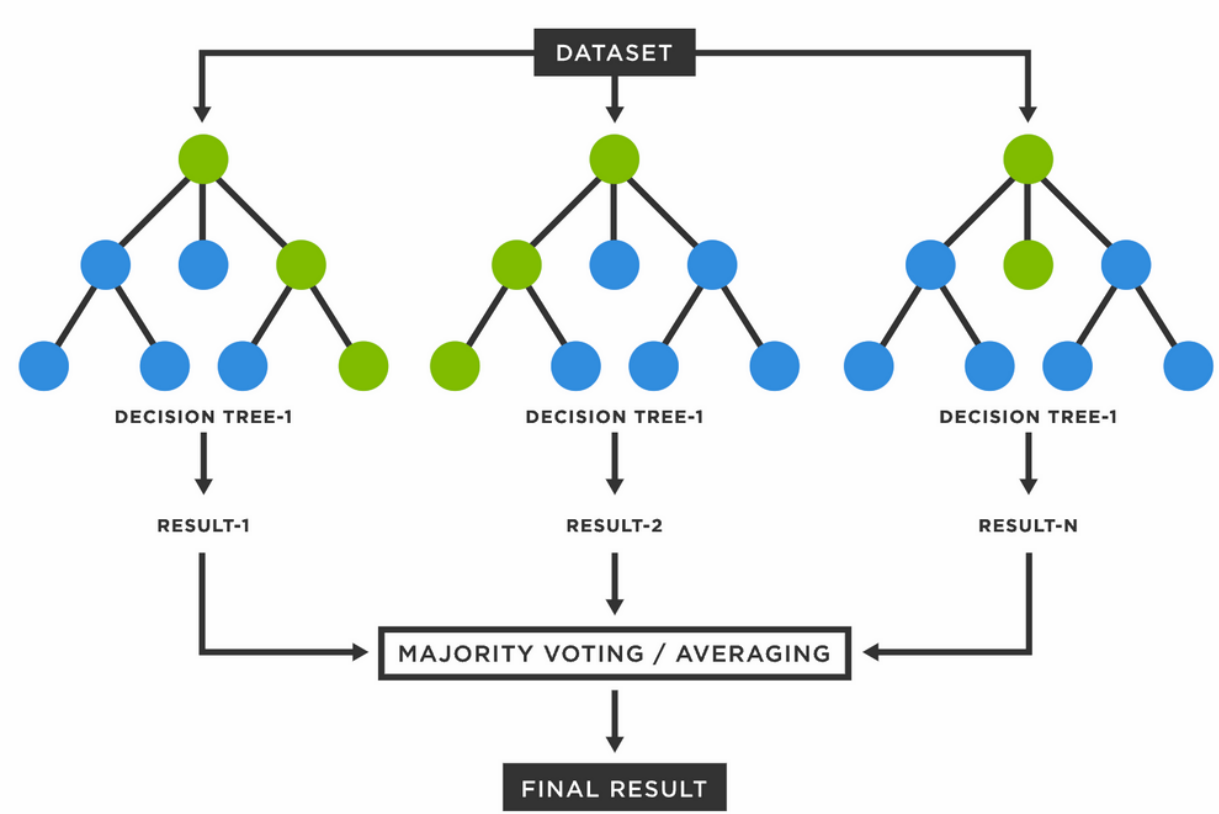
```
digraph Tree {
	node [shape=box, style="filled", color="black", fontname="helvetica"] ;
	edge [fontname="helvetica"] ;
	0 [label="X2 <= 3.5\nsamples = 10\nvalue = [5, 5]\nclass = 0", fillcolor="#ffffff"] ;
	1 [label="X1 <= 6.5\nsamples = 5\nvalue = [1, 4]\nclass = 1", fillcolor="#6ab6ec"] ;
	0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
	2 [label="samples = 4\nvalue = [0, 4]\nclass = 1", fillcolor="#399de5"] ;
	1 -> 2 ;
	3 [label="samples = 1\nvalue = [1, 0]\nclass = 0", fillcolor="#e58139"] ;
	1 -> 3 ;
	4 [label="X1 <= 1.5\nsamples = 5\nvalue = [4, 1]\nclass = 0", fillcolor="#eca06a"] ;
	0 -> 4 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
	5 [label="samples = 1\nvalue = [0, 1]\nclass = 1", fillcolor="#399de5"] ;
	4 -> 5 ;
	6 [label="samples = 4\nvalue = [4, 0]\nclass = 0", fillcolor="#e58139"] ;
	4 -> 6 ;
}
```



Python code: 10. DC_scikitlearn_.py

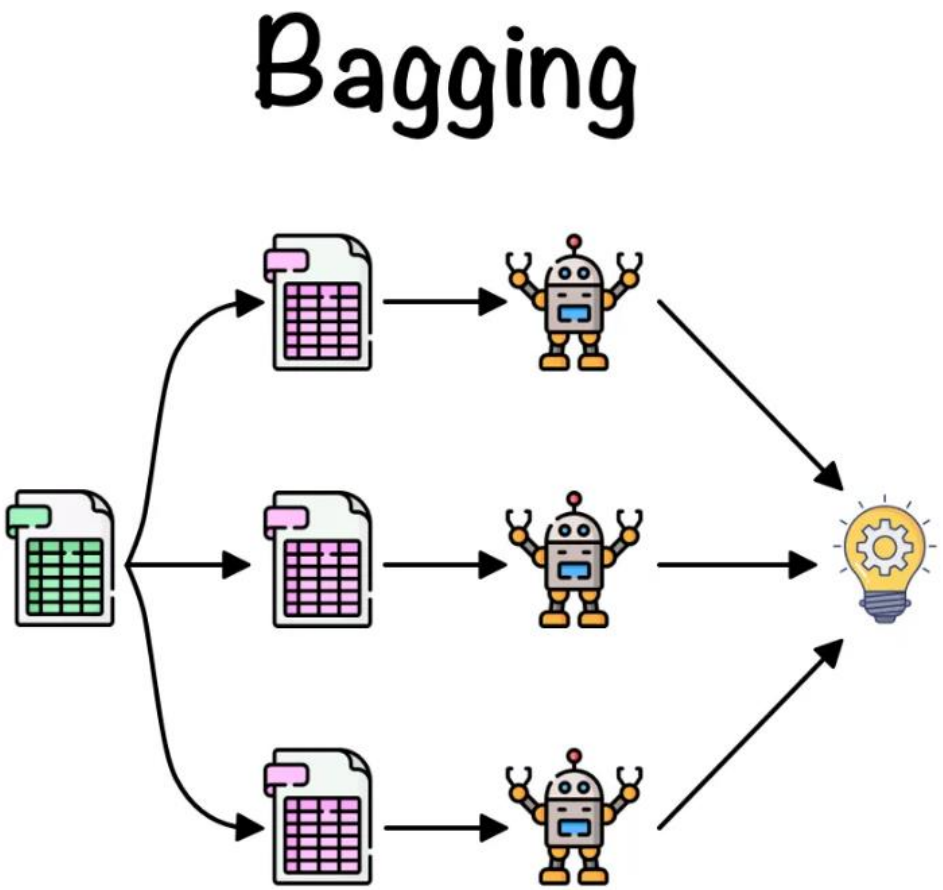
Ensembling decision trees – random forest

- L'ensamble di Decision Trees si definisce come l'utilizzo di più alberi di decisione, creando una foresta di alberi (**Random Forest**), che prendano decisioni su set di dati diversi.
- Il bagging riduce l'elevata varianza di un Decision Tree e ne migliora le prestazioni.



Ensembling decision trees – random forest

- La tecnica di **ensemble** del **bagging** (aggregazione bootstrap) può superare efficacemente l'overfitting.
- Questa tecnica si basa su una **votazione a maggioranza** dei modelli che prendono parte al processo, per prendere la **decisione finale.**



Ensembling decision trees – random forest

- L'aggregazione di più alberi correlati non farà molta differenza.
- Per costringere ogni albero a non essere correlato, il Random Forest considera solo un sottoinsieme casuale delle caratteristiche quando cerca il miglior punto di divisione in ogni nodo.
- I singoli alberi sono ora addestrati sulla base di diversi insiemi sequenziali di caratteristiche, il che garantisce una maggiore diversità e migliori prestazioni

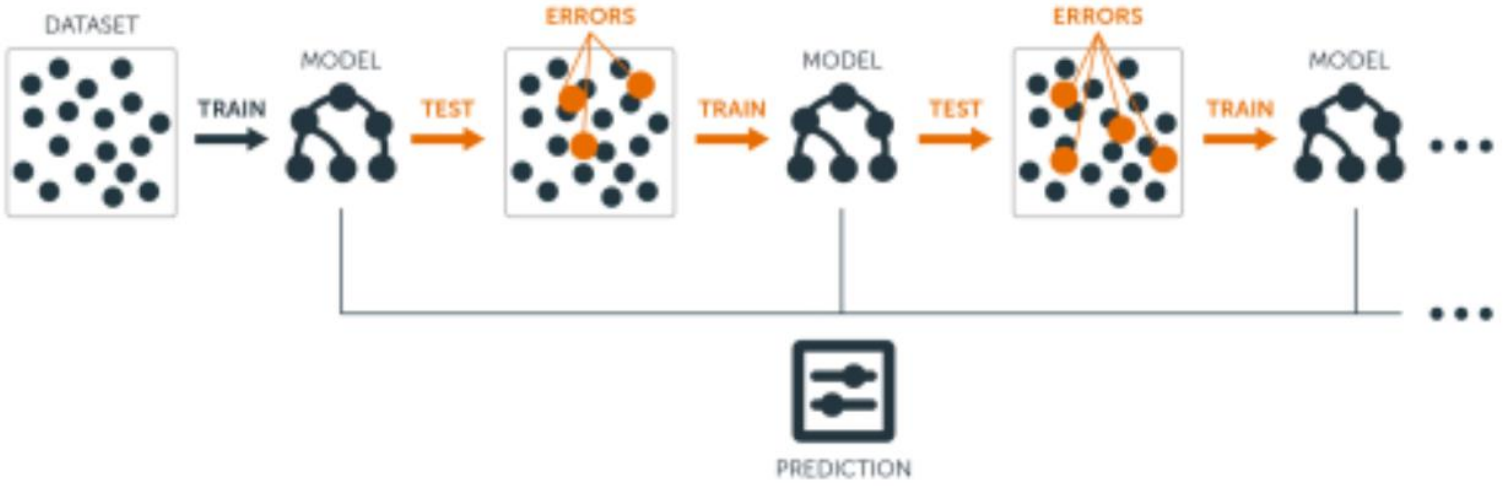
Python code: 12. home-data-for-ml-course

Parametri – random forest

- **max_depth**: È il singolo albero più profondo. Tende a sovra-adattarsi se è troppo profondo o a sotto-adattarsi se è troppo superficiale.
- **min_samples_split**: Questo iperparametro rappresenta il numero minimo di campioni necessari per un'ulteriore suddivisione in un nodo. Un valore troppo piccolo tende a causare un overfitting, mentre un valore troppo grande rischia di introdurre un underfitting. 10, 30 e 50 potrebbero essere buone opzioni per iniziare.
- **max_features**: Questo parametro rappresenta il numero di caratteristiche da considerare per ogni ricerca del miglior punto di divisione. In genere, per un insieme di dati m-dimensionali, $\sqrt{m}$ (arrotondato) è un valore consigliato per max_features. Questo può essere specificato come max_features="sqrt" in scikit-learn. Altre opzioni includono log2, 20% e 50% delle caratteristiche originali.
- **n_estimator**: Questo parametro rappresenta il numero di alberi considerati per la votazione a maggioranza. In generale, maggiore è il numero di alberi, migliore è il risultato della votazione a maggioranza. ma più lungo è il tempo di calcolo. Di solito è impostato su 100, 200, 500 e così via.

Ensembling decision trees – gradient boosted trees

- Il boosting, un'altra tecnica di ensemble, adotta un approccio iterativo invece di combinare più learners in parallelo. Negli alberi boostati, i singoli alberi non vengono più addestrati separatamente.
- In particolare, nei gradient boosted trees (GBT) (chiamati anche gradient boosting machines), i singoli alberi vengono addestrati in successione, dove un albero mira a correggere gli errori commessi dall'albero precedente.



Python code: 13. Gradient Boosted Trees