



# Intelligenza Artificiale e al Machine Learning

**Classificazione supervisionata**



# Classificazione: Definizione

- Data una collezione di record (*training set*)
  - ✓ Ogni record è composto da un insieme di *attributi*, di cui uno esprime la *classe* di appartenenza del record.
- Trova un *modello* per l'attributo di classe che esprima il valore dell'attributo in funzione dei valori degli altri attributi.
- Obiettivo: record non noti devono essere assegnati a una classe nel modo più accurato possibile
  - ✓ Viene utilizzato un *test set* per determinare l'accuratezza del modello. Normalmente, il data set fornito è *suddiviso* in training set e test set. Il primo è utilizzato per costruire il modello, il secondo per validarlo.
- I classificatori possono essere utilizzati sia a scopo descrittivo sia a scopo predittivo
- Sono più adatti ad attributi nominali (binari o discreti) poiché faticano a sfruttare le relazioni implicite presenti negli attributi ordinali, numerici o in presenza di gerarchie di concetti (es. scimmie e uomini sono primati)

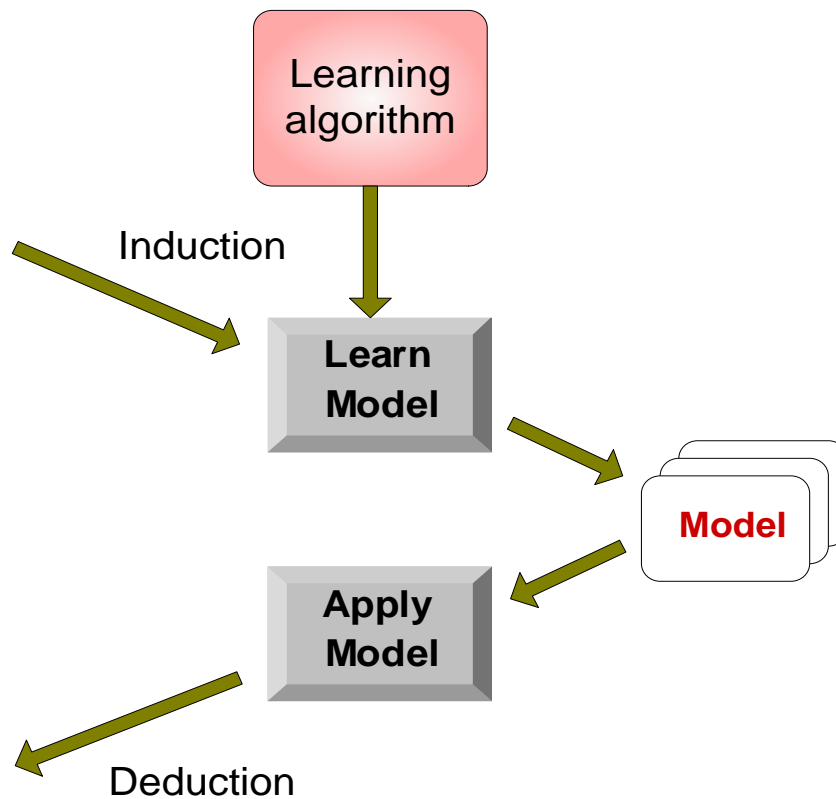
# Classificazione: un esempio

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

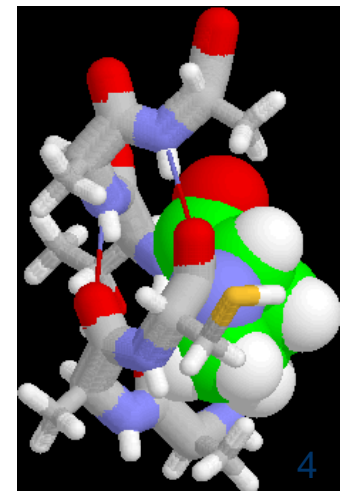
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



# Applicazioni

- Predire se una cellula tumorale è benigna o maligna in base alle sue caratteristiche
- Classificare se una transazione con carta di credito sia o meno fraudolenta
- Classificare le strutture proteiche secondarie in alpha-helix, beta-sheet, or random coil
- Classificare le news in base all'argomento: finanza, meteo, sport, intrattenimento, ecc.



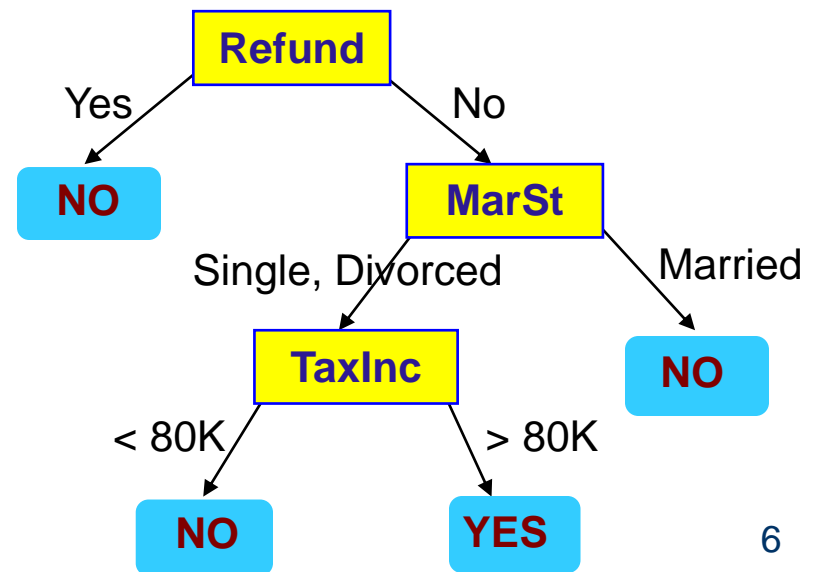


# Tecniche di classificazione

- **Alberi decisionali o Decision Tree**
- Regole di decisione
- Nearest-neighbor
- Reti Bayesiane
- Reti neurali
- Support Vector Machines

# I Decision Tree

- È una delle tecniche di classificazione maggiormente utilizzate che permette di rappresentare con un albero un insieme di regole di classificazione.
- Struttura gerarchica che consiste di un insieme di nodi, correlati da archi (rami) orientati ed "etichettati". Si hanno due tipi di nodi:
  - ✓ Le classi sono definite nei nodi foglia mentre i rimanenti nodi sono etichettati in base all'attributo che partiziona i record. Il criterio di partizionamento rappresenta l'etichetta degli archi
- Ciascun percorso radice-foglia rappresenta una regola di classificazione



# Decision Tree: un esempio

Categorico

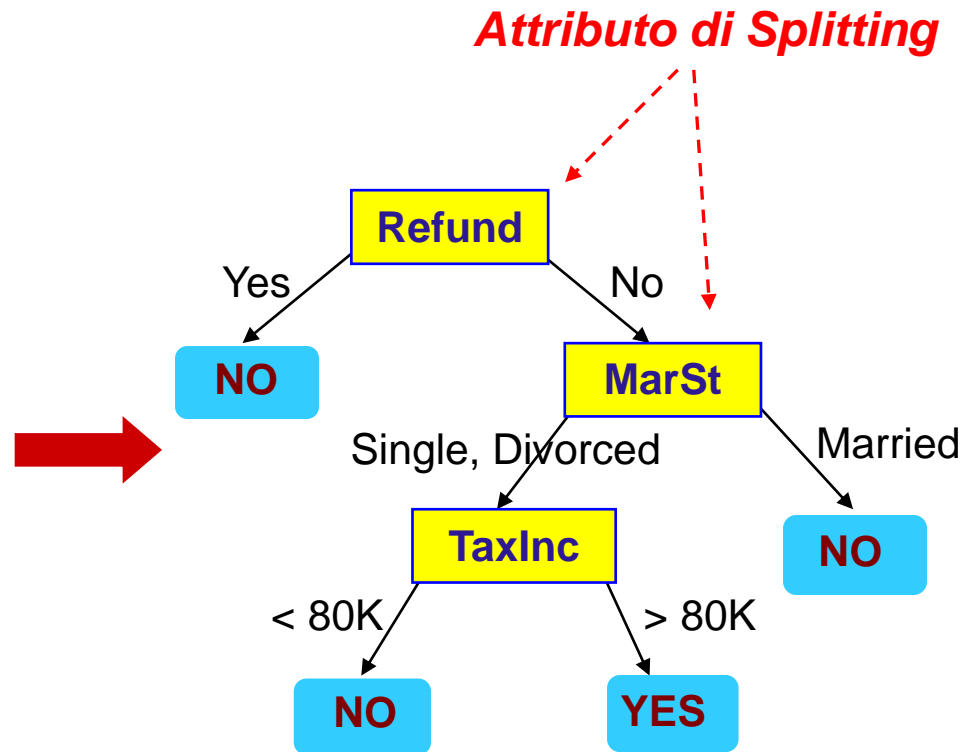
Categorico

Continuo

Class

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data

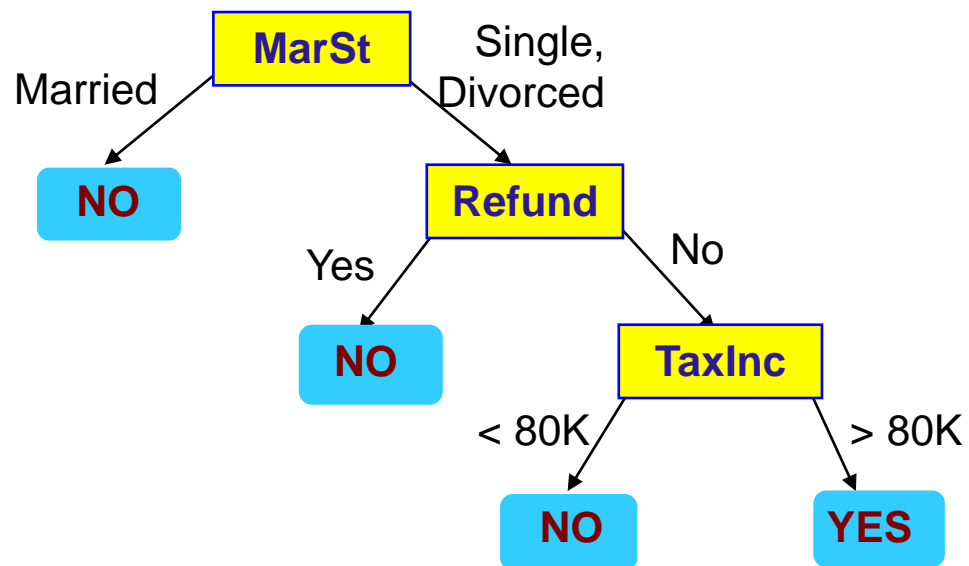


Modello: Decision Tree

# Decision Tree: un altro esempio

Categorico  
Categorico  
Continuo  
Classe

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



Ci possono essere più alberi di decisione per lo stesso data set



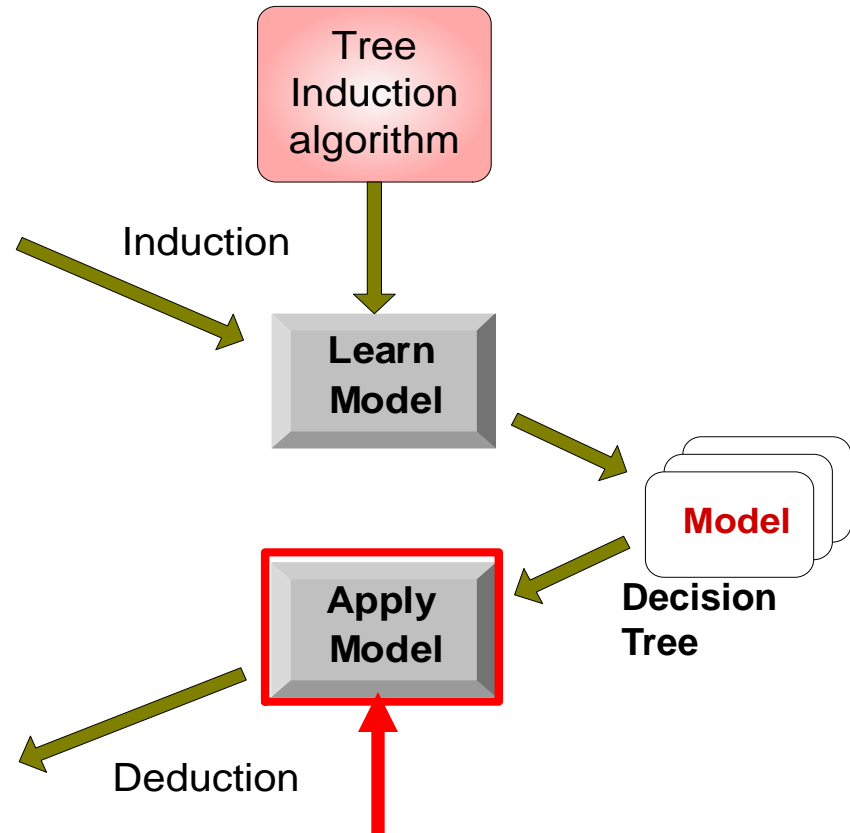
# Applicare il modello al data set

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

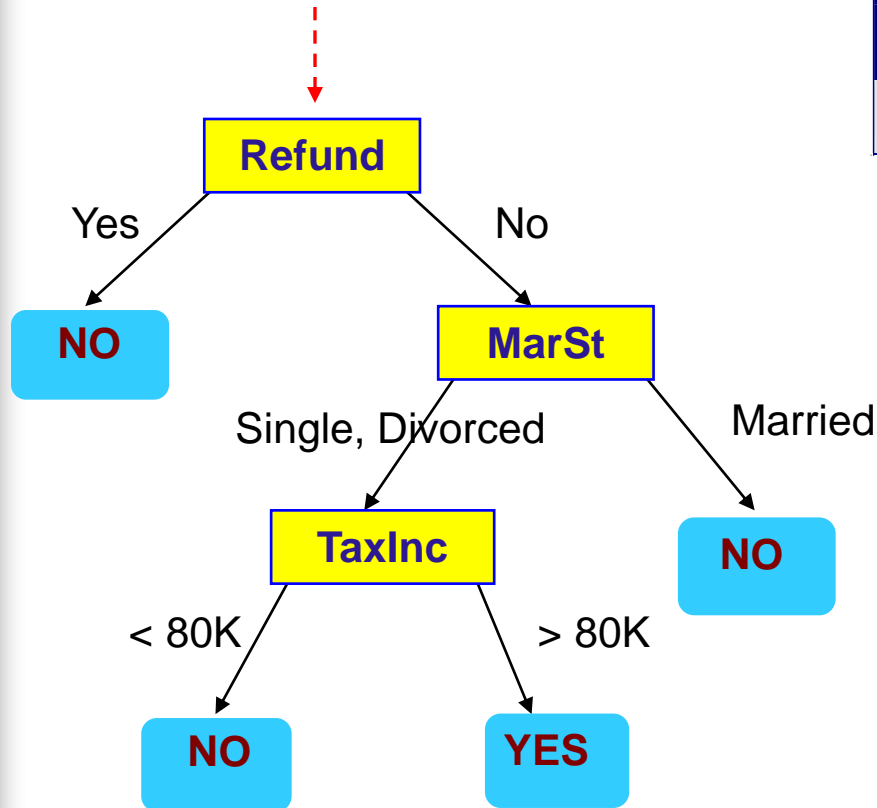
Test Set



# Applicare il modello al data set

## Test Data

Si parte dalla radice

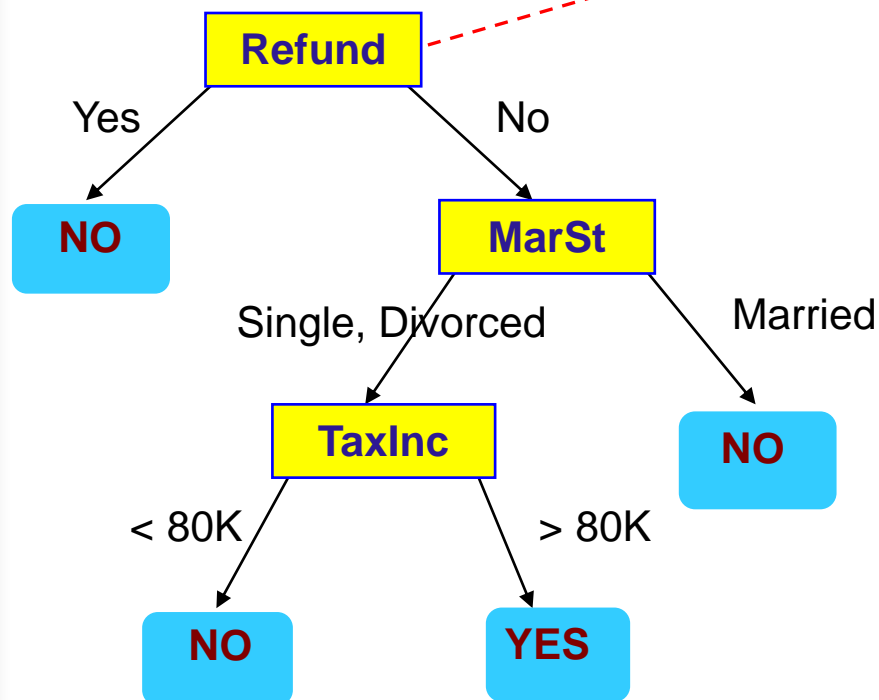


Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

# Applicare il modello al data set

Test Data

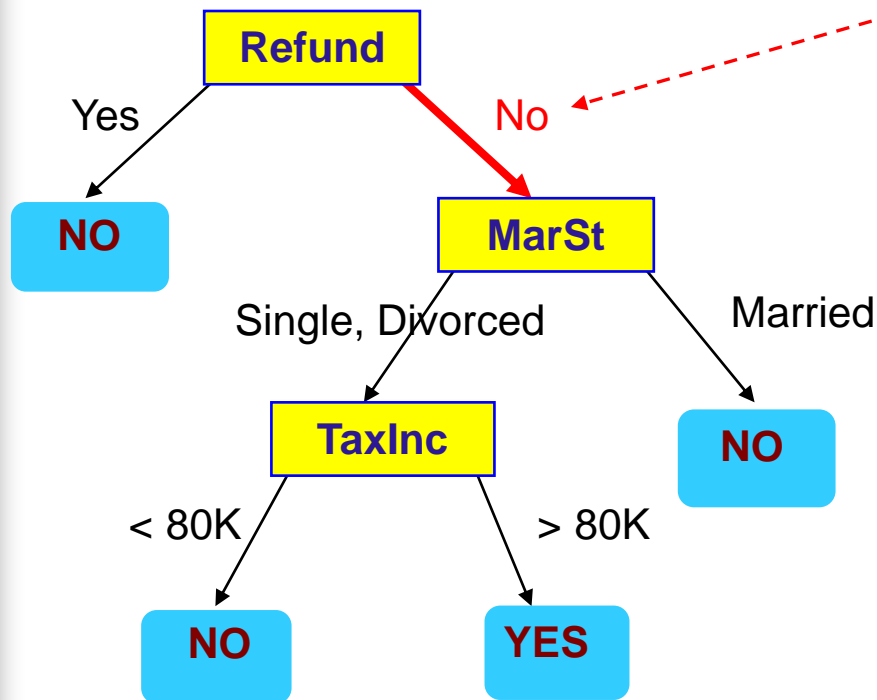
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



# Applicare il modello al data set

Test Data

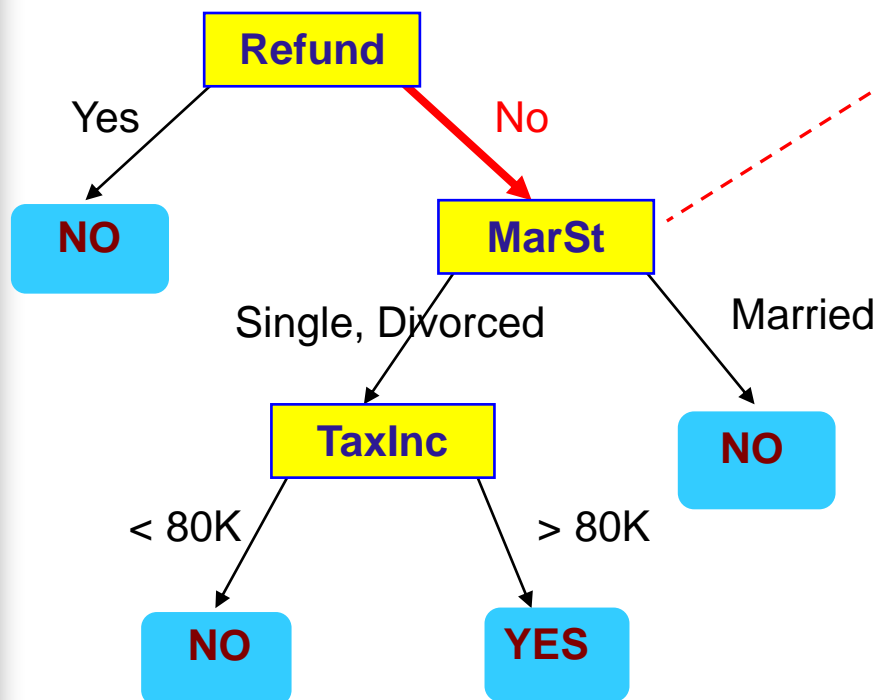
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



# Applicare il modello al data set

Test Data

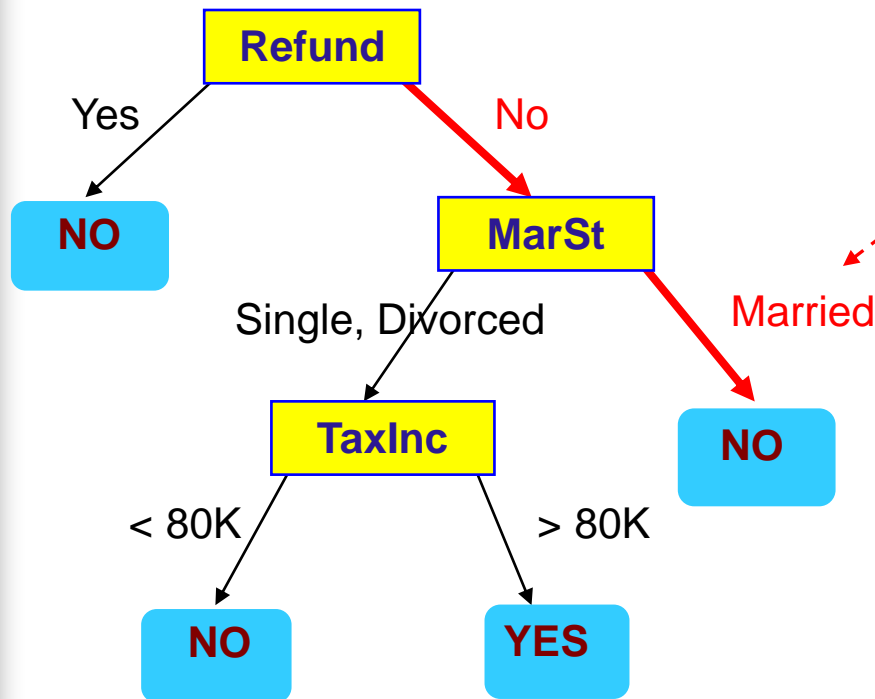
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



# Applicare il modello al data set

Test Data

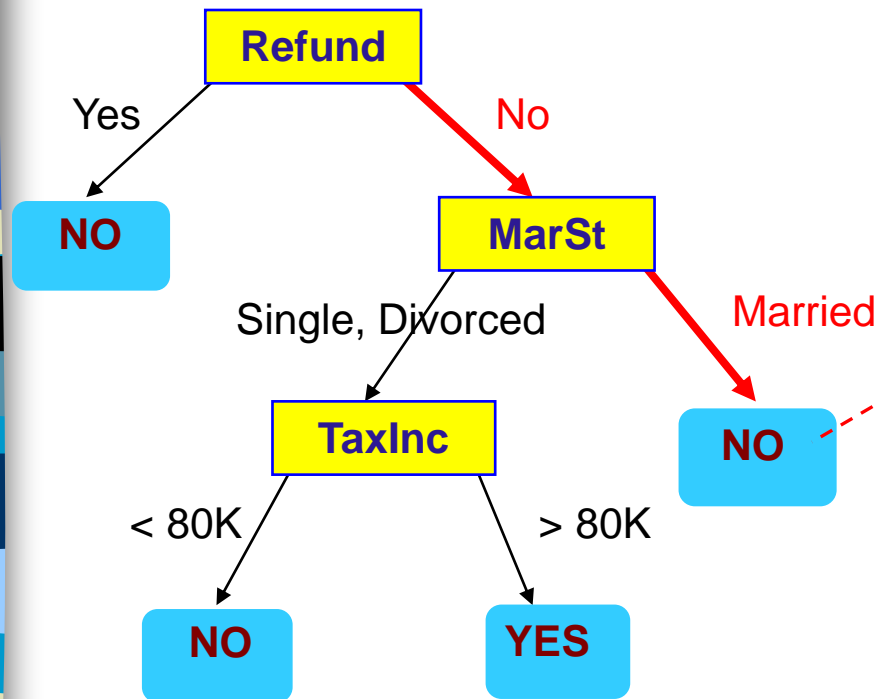
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



# Applicare il modello al data set

## Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Poni Cheat = "No"

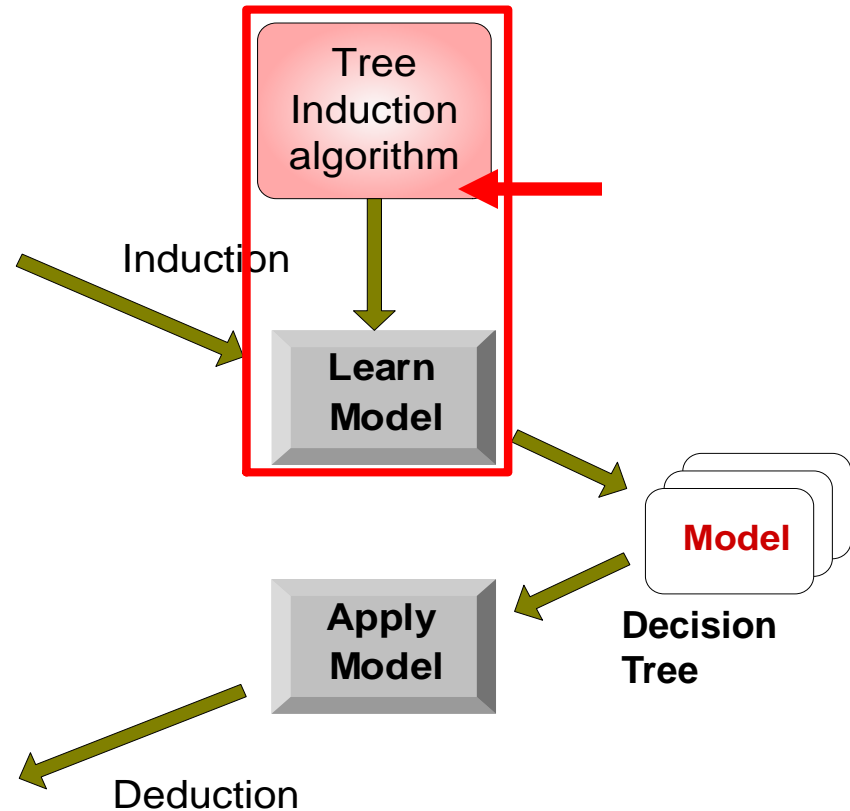
# Decision Tree: imparare il modello

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set







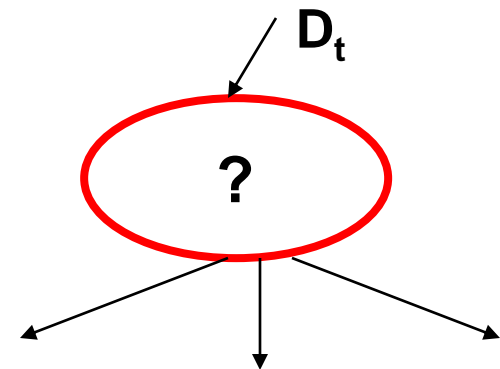
# Induzione con Decision Tree

- Il numero di decision tree cresce esponenzialmente con il numero di attributi
- Gli algoritmi utilizzano generalmente tecniche greedy che fanno localmente la scelta “migliore”
- Sono a disposizione molti algoritmi:
  - ✓ Hunt's Algorithm
  - ✓ CART
  - ✓ ID3, C4.5
  - ✓ SLIQ, SPRINT
- Devono essere affrontati diversi problemi
  - ✓ Scelta del criterio di split
  - ✓ Scelta del criterio di stop
  - ✓ Underfitting
  - ✓ Overfitting
  - ✓ Frammentazione dei dati
  - ✓ Criterio di ricerca
  - ✓ Espressività
  - ✓ Replicazione degli alberi

# Hunt's Algorithm

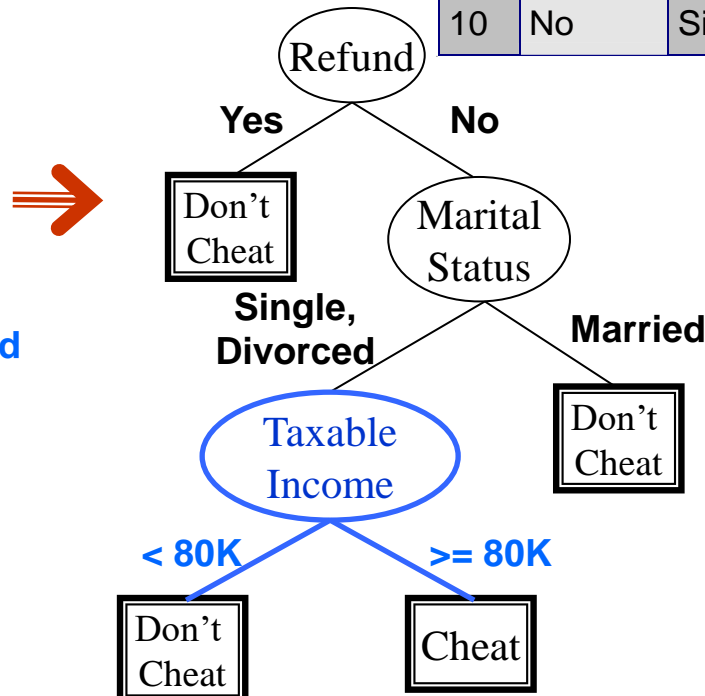
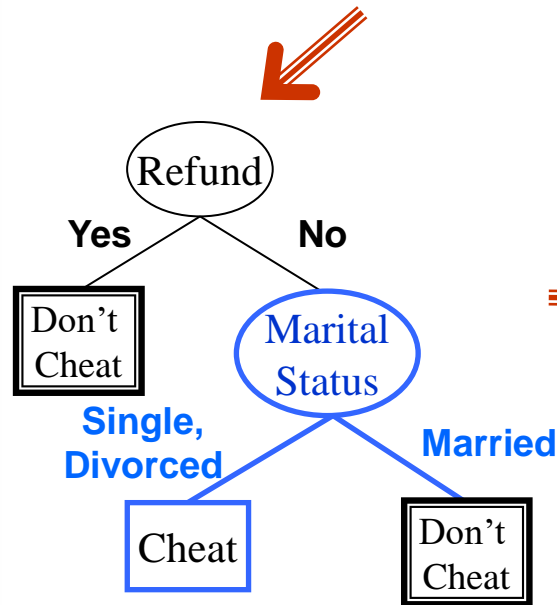
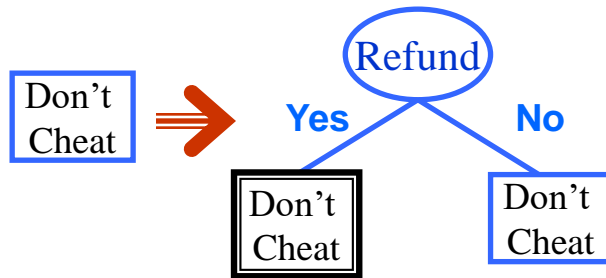
- Approccio ricorsivo che suddivide progressivamente un insieme di record  $D_t$  in insiemi di record via via più puri
- Sia  $D_t$  l'insieme dei record del training set corrispondenti al nodo  $t$  e  $y_t = \{y_1, \dots, y_k\}$  le possibili label di classe
- Procedura generale:
  - ✓ Se  $D_t$  contiene record appartenenti alla sola classe  $y_j$ , allora  $t$  è un nodo foglia con label  $y_j$
  - ✓ Se  $D_t$  è un insieme vuoto, allora  $t$  è un nodo foglia a cui è assegnata una classe del nodo padre
  - ✓ Se  $D_t$  contiene record appartenenti a più classi, si scelga un **attributo e un criterio di split** per partizionare i record in più sottoinsiemi.
  - ✓ Si riapplichino ricorsivamente la procedura generale ai sottoinsiemi

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



# Hunt's Algorithm

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



# Pseudo-codice di massima

```
// Let E be the training set and F the attributes
```

```
result=PostPrune(TreeGrowth(E,F));
```

```
TreeGrowth(E,F)
```

```
  if StoppingCond(E,F) = TRUE then
```

```
    leaf=CreateNode();
```

```
    leaf.label=Classify(E);
```

```
    return leaf;
```

```
  else
```

```
    root = CreateNode();
```

```
    root.test_cond = FindBestSplit(E,F);
```

```
    let V = {v | v is a possible outcome of root.test_cond}
```

```
  for each v ∈ V do
```

```
    Ev = {e | root.test_cond(e)=v and e ∈ E}
```

```
    child = TreeGrowth(Ev,F);
```

```
    add child as descendants of root and label edge
```

```
      (root→child) as v
```

```
  end for
```

```
end if
```

```
  return root;
```

```
end;
```

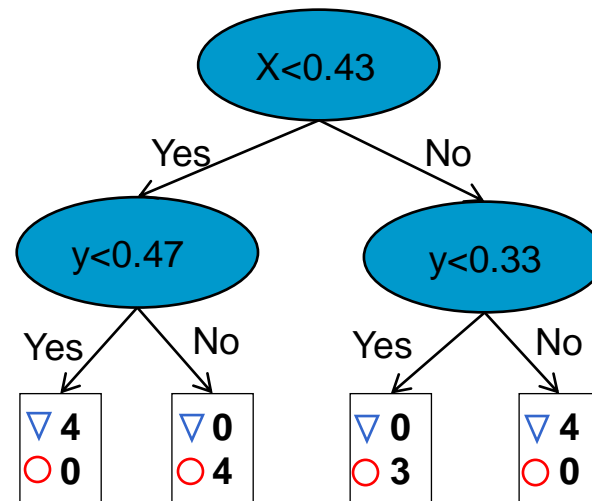
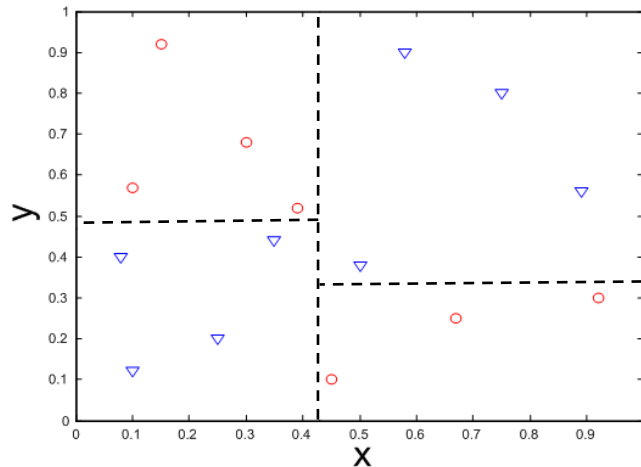
# Alcune considerazioni...

- La ricerca di un albero di decisione ottimo è un problema NP-Completo, ma gli algoritmi euristici utilizzati sono molto efficienti
  - ✓ La maggior parte degli approcci eseguono una partizione ricorsiva top down basata su criteri greedy
- La classificazione utilizzando un albero decisionale è estremamente veloce e offre una facile interpretazione dei criteri
  - ✓ Il caso peggiore è  $O(w)$  dove  $w$  è la profondità dell'albero
- Gli alberi di decisione sono sufficientemente robusti rispetto alla presenza di attributi fortemente correlati
  - ✓ Uno dei due attributi non sarà considerato
  - ✓ E' anche possibile cercare di scartare uno degli attributi in fase di preprocessing mediante opportune tecniche di feature selection

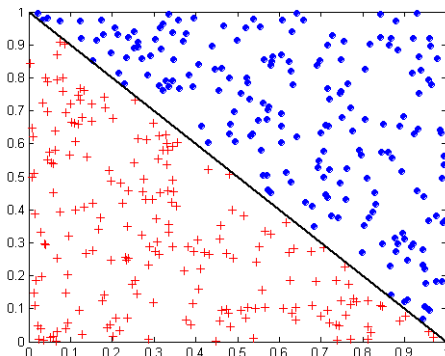
# Alcune considerazioni...

- L'espressività degli alberi decisionali è limitata alla possibilità di effettuare partizionamenti dello spazio di ricerca con condizioni che coinvolgono un solo attributo per volta

✓ Decision boundary paralleli agli assi



- Questa suddivisione non è ottenibile con alberi decisionali tradizionali



$$X - Y = 1$$



# Elementi caratterizzanti

- A parte la logica di base per definire completamente un algoritmo per la costruzione di alberi decisionali è necessario definire:
  - ✓ **La condizione di split**
  - ✓ Il criterio che definisce lo split migliore
  - ✓ Il criterio per interrompere lo splitting
  - ✓ Le modalità per valutare la bontà di un albero decisionale



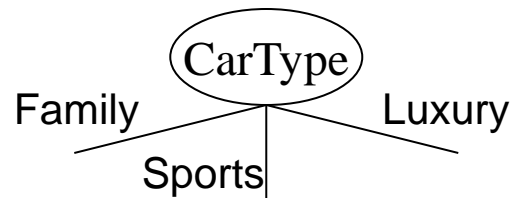
# Come definire la condizione di split

- Dipende dal tipo di attributo
  - ✓ Nominale
  - ✓ Ordinale
  - ✓ Continuo
- Dipende dal numero di split applicabili ai valori dell'attributo
  - ✓ A 2 vie
  - ✓ A più vie

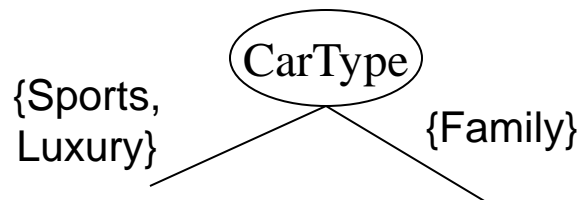


# Splitting con attributi nominali

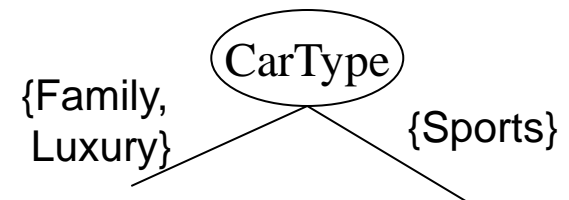
- **Split a più vie:** crea tante partizioni quanti sono i valori dell'attributo



- **Split a 2 vie:** crea due partizioni e richiede di suddividere i possibili valori dell'attributo in modo ottimale

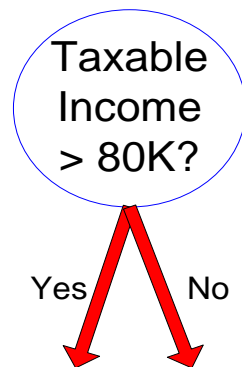


OR

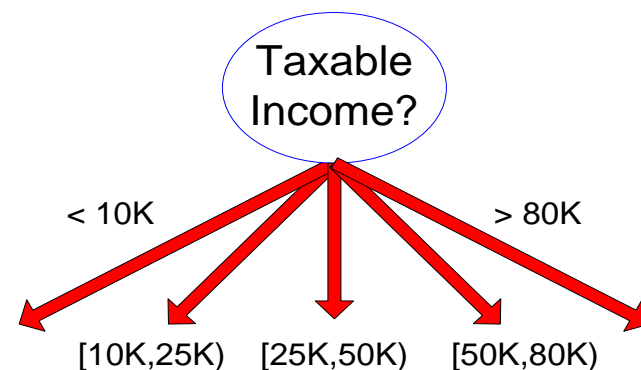


# Splitting con attributi continui

- **Split a più vie:** la condizione di split può essere espressa come un test di comparazione che ha per risultato più range di valori. L'algoritmo deve considerare tutti i possibili range di valori come possibili punti di split
- **Split a 2 vie:** la condizione di split può essere espressa come un test di comparazione con risultato binario. L'algoritmo deve considerare tutti i valori come possibili punti di split



(i) Binary split



(ii) Multi-way split

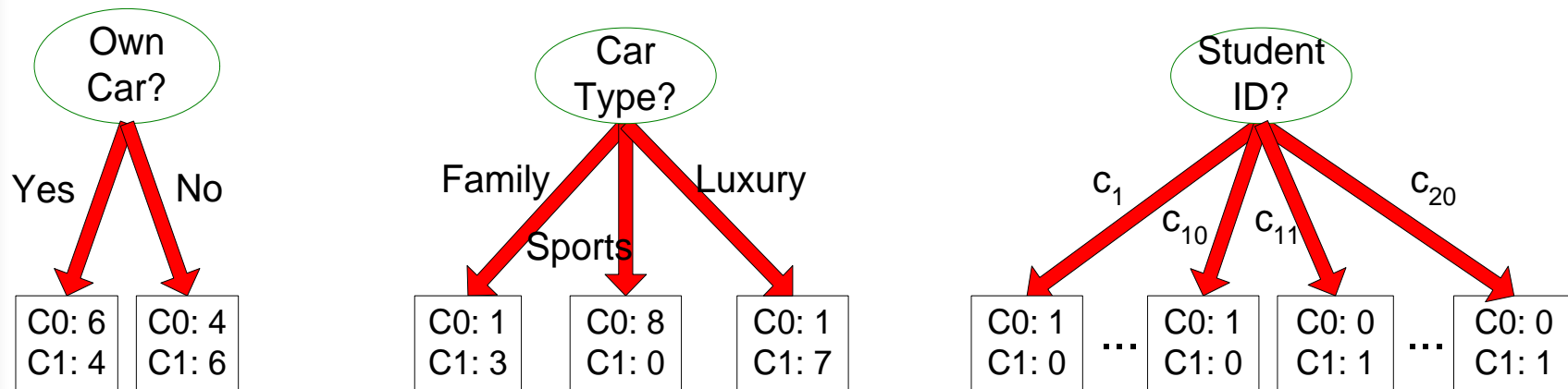


# Elementi caratterizzanti

- A parte la logica di base per definire completamente un algoritmo per la costruzione di alberi decisionali è necessario definire:
  - ✓ La condizione di split
  - ✓ **Il criterio che definisce lo split migliore**
  - ✓ Il criterio per interrompere lo splitting
  - ✓ Le modalità per valutare la bontà di un albero decisionale

# Come determinare lo split migliore

- Prima dello split una sola classe con 10 record in classe C0 e 10 record in classe C1



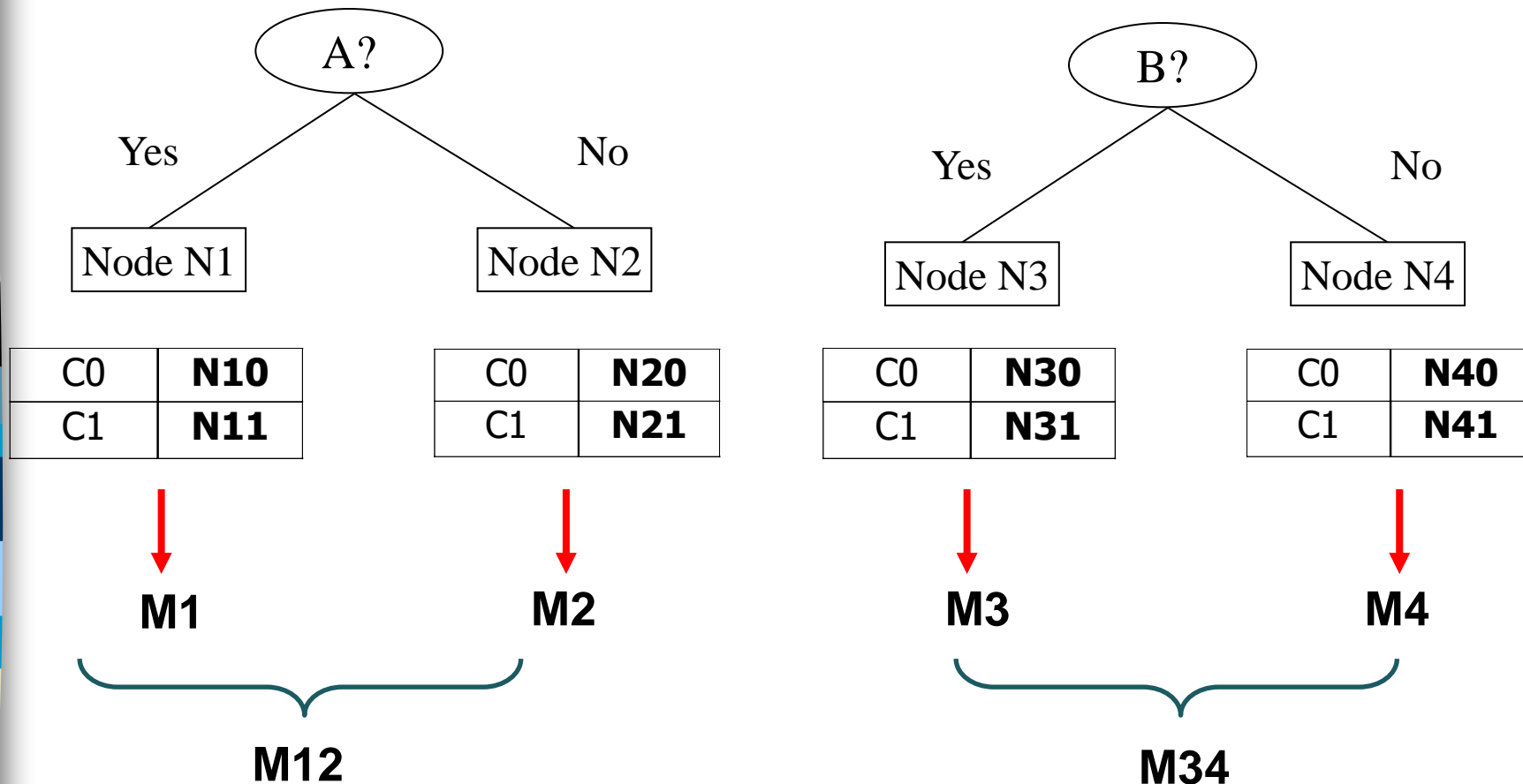
- Il criterio di split deve permettere di determinare classi più pure. Serve una misura di purezza
  - ✓ Gini index
  - ✓ Entropia
  - ✓ Misclassification error

# Come determinare lo split migliore

Prima dello Splitting:

C0	<b>N00</b>
C1	<b>N01</b>

→ **M0**



$$\text{Gain} = M0 - M12 \text{ vs } M0 - M34$$

# Misure di impurità

- Dato un nodo  $p$  con record appartenenti a  $k$  classi e un suo partizionamento in  $n$  nodi figli

- ✓  $m$  = numero di record nel padre  $p$
- ✓  $m_i$  = numero di record nel figlio  $i$

**ATTENZIONE a non confondere il numero delle classi ( $k$ )  
e quello dei nodi figli ( $n$ )**

- Gini index: usato in CART, SLIQ, SPRINT.

$$GINI(i) = 1 - \sum_{j=1}^k [p(j|i)]^2$$

- Entropia usato in ID3 e C4.5

$$Entropy(i) = - \sum_{j=1}^k p(j|i) \log p(j|i)$$

- Errore di classificazione

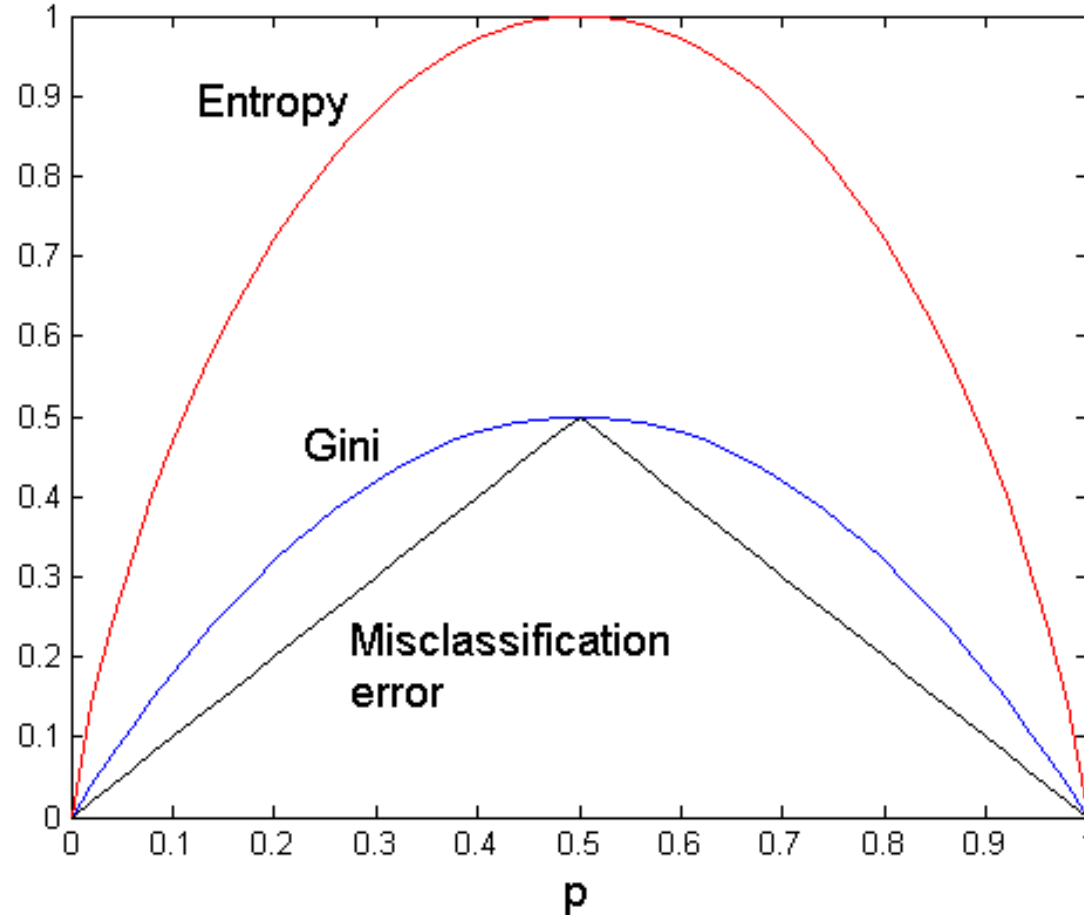
$$Error(i) = 1 - \max_{j \in K} p(j|i)$$

- Impurità complessiva dello split è data dalla seguente formula dove  $meas()$  è una delle misure introdotte

$$Impurity_{split} = \sum_{i=1}^n \frac{m_i}{m} meas(i)$$

# Una comparazione dei criteri di splitting

- Valore dei diversi indici, per un partizionamento in due classi



# Split basato sul GAIN

- Utilizzando misure di impurità delle classi come Gini e Entropy richiede di scegliere il valore di split che massimizza il “guadagno” in termini di riduzione dell’impurità delle classi dopo lo split.
- Per esempio, considerando l’entropia, il guadagno del partizionamento di un nodo  $p$  in  $n$  nodi figli è:

$$GAIN_{split} = Entropy(p) - \left( \sum_{i=1}^n \frac{m_i}{m} Entropy(i) \right)$$

- Selezionare il valore di split che massimizza il GAIN tende a determinare criteri di split che generano un numero molto elevato di classi molto pure e con pochi record.
  - ✓ Partizionare gli studenti in base alla loro matricola garantisce che tutte le classi (formate da un solo studente) siano totalmente pure!!



# Split basato sulle INFO

- Per evitare il problema della polverizzazione delle classi è preferibile massimizzare il Gain Ratio:
  - ✓  $n$  = numero di nodi figli
  - ✓  $m$  = numero di record nel padre  $p$
  - ✓  $m_i$  = numero di record nel figlio  $i$

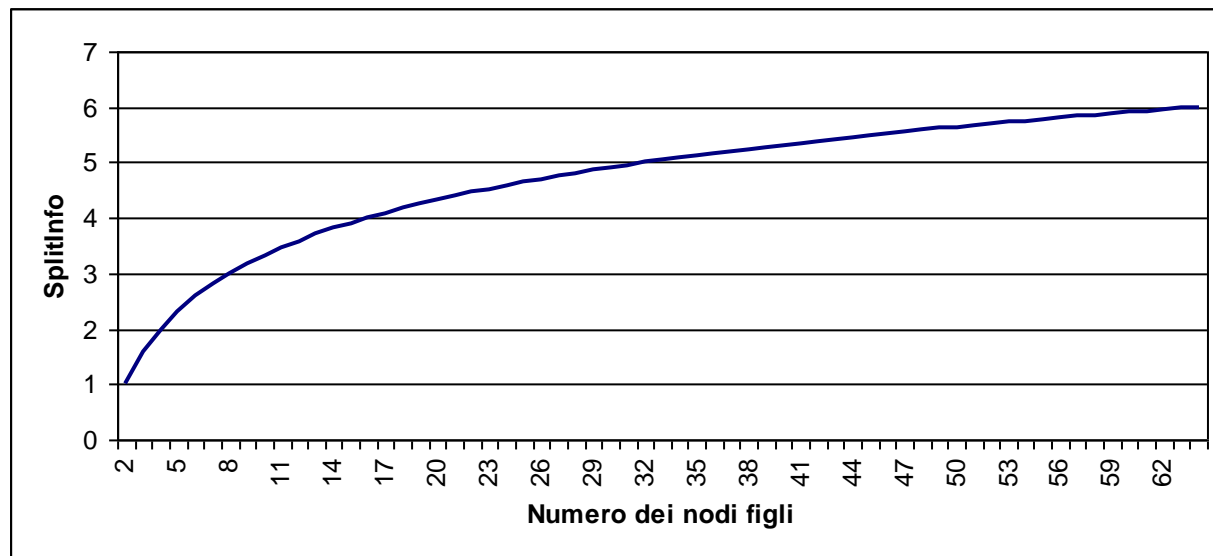
$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO}$$

$$SplitINFO = - \sum_{i=1}^n \frac{m_i}{m} \log \frac{m_i}{m}$$

- ✓ Maggiore il numero dei figli, maggiore il valore di SplitInfo con una conseguente riduzione del GainRatio
- ✓ Per esempio, assumendo che ogni nodo figlio contenga lo stesso numero di record, SplitInfo =  $\log n$ .
- ✓ C4.5 utilizza il criterio basato su SplitINFO

# Split basato sulle INFO

- Per evitare il problema della polverizzazione delle classi è preferibile massimizzare il Gain Ratio:
  - ✓  $n = \text{da } 2 \text{ a } 64$
  - ✓  $m = 100$
  - ✓  $m_i = m/n$





# Elementi caratterizzanti

- A parte la logica di base per definire completamente un algoritmo per la costruzione di alberi decisionali è necessario definire:
  - ✓ La condizione di split
  - ✓ Il criterio che definisce lo split migliore
  - ✓ **Il criterio per interrompere lo splitting**
  - ✓ Le modalità per valutare la bontà di un albero decisionale



# Criteri di stop per l'induzione di alberi decisionali

- Interrompere lo split di un nodo quando tutti i suoi record appartengono alla stessa classe
- Interrompere lo split di un nodo quando tutti i suoi record hanno valori simili su tutti gli attributi
  - ✓ La classificazione sarebbe poco significativa e dipendente da piccole fluttuazioni dei valori
- Interrompere lo split quando il numero dei record nel nodo è inferiore a una certa soglia (*data fragmentation*)
  - ✓ Il criterio selezionato non sarebbe statisticamente rilevante

# Metriche per la valutazione del modello

- La Confusion Matrix valuta la capacità di un classificatore sulla base dei seguenti indicatori
  - ✓ TP (true positive): record correttamente classificati come classe Yes
  - ✓ FN (false negative): record **incorrettamente** classificati come classe No
  - ✓ FP (false positive): record **incorrettamente** classificati come classe Yes
  - ✓ TN (true negative) record correttamente classificati come classe No

	Classe prevista		
Classe effettiva		Class=Yes	Class=No
	Class=Yes	TP	FN
	Class=No	FP	TN

- Se la classificazione utilizza  $n$  classi, la matrice di confusione sarà di dimensione  $n \times n$

# Accuratezza

	Classe prevista		
Classe effettiva		Class=Yes	Class=No
	Class=Yes	TP	FN
	Class=No	FP	TN

- L'accuratezza è la metrica maggiormente utilizzata per sintetizzare l'informazione di una confusion matrix

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Equivalentemente potrebbe essere utilizzata la frequenza dell'errore

$$\text{Error rate} = \frac{FP + FN}{TP + TN + FP + FN}$$



# Limiti dell'accuratezza

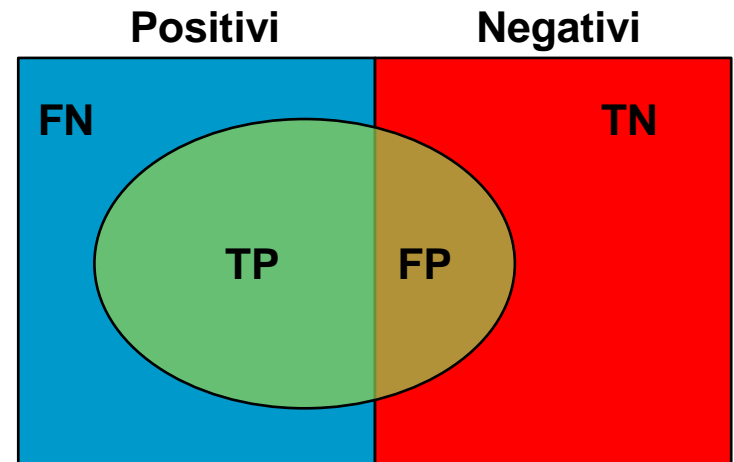
- L'accuratezza non è una metrica adeguata nel caso in cui le classi contengano un numero fortemente diverso di record
  - ✓ Consideriamo un problema di classificazione binario in cui
    - # record della classe 0 = 9990
    - # record della classe 1 = 10
  - ✓ Un modello che predica sempre l'appartenenza alla classe 0 avrà un'accuratezza di  $9990/10000 = 99.9 \%$
- Nel caso di problemi di classificazione binaria la classe la classe “rara” è anche chiamata *classe positiva*, mentre la classe che include la maggioranza dei record è chiamata *classe negativa*

# Precision e Recall

- Precision e Recall sono due metriche utilizzate nelle applicazioni in cui la corretta classificazione dei record della classe positiva riveste una maggiore importanza
  - ✓ **Precision** misura la frazione di record risultati effettivamente positivi tra tutti quelli che erano stati classificati come tali
    - ✓ Valori elevati indicano che pochi record della classe negativa sono stati erroneamente classificati come positivi.
  - ✓ **Recall** misura la frazione di record positivi correttamente classificati
    - ✓ Valori elevati indicano che pochi record della classe positiva sono stati erroneamente classificati come negativi.

$$\text{Precision, } p = \frac{TP}{TP + FP}$$

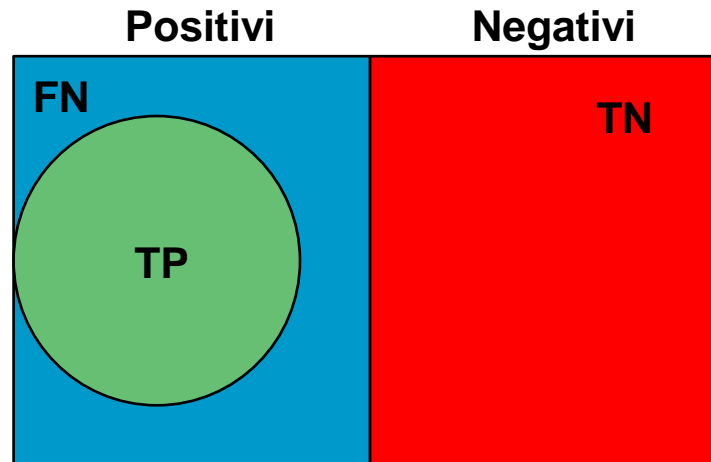
$$\text{Recall, } r = \frac{TP}{TP + FN}$$



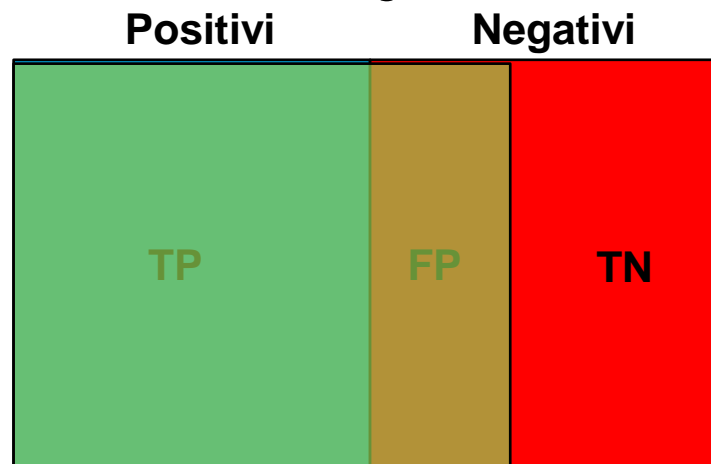


# Precision e Recall

- precision = 1 se tutti i record positivi sono stati effettivamente individuati



- recall = 1 se non ci sono falsi negativi



- Se entrambi valgono 1 le classi predette coincidono con quelle reali

# Matrice dei costi

- La matrice dei costi codifica la penalità in cui si incorre nel classificare un record in una classe diversa
  - ✓ Una penalità negativa indica il “premio” che si ottiene per una corretta classificazione

$$C(M) = TP \times C(\text{Yes}|\text{Yes}) + FP \times C(\text{Yes}|\text{No}) + FN \times C(\text{No}|\text{Yes}) + TN \times C(\text{No}|\text{No})$$

	Classe prevista j		
	C(i j)	Class=Yes	Class=No
Classe effettiva i	Class=Yes	C(Yes Yes)	C(Yes No)
	Class=No	C(No Yes)	C(No No)

- Un modello costruito struttando, come funzione di purezza, una matrice di costo tenderà a fornire un modello a costo minimo rispetto ai pesi specificati

# Calcolo del costo

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i j)	+	-
	+	-1	100
	-	1	0

Model $M_1$	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	150	40
	-	60	250

Accuracy = 80%

Cost = 3910

Model $M_2$	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	250	45
	-	5	200

Accuracy = 90%

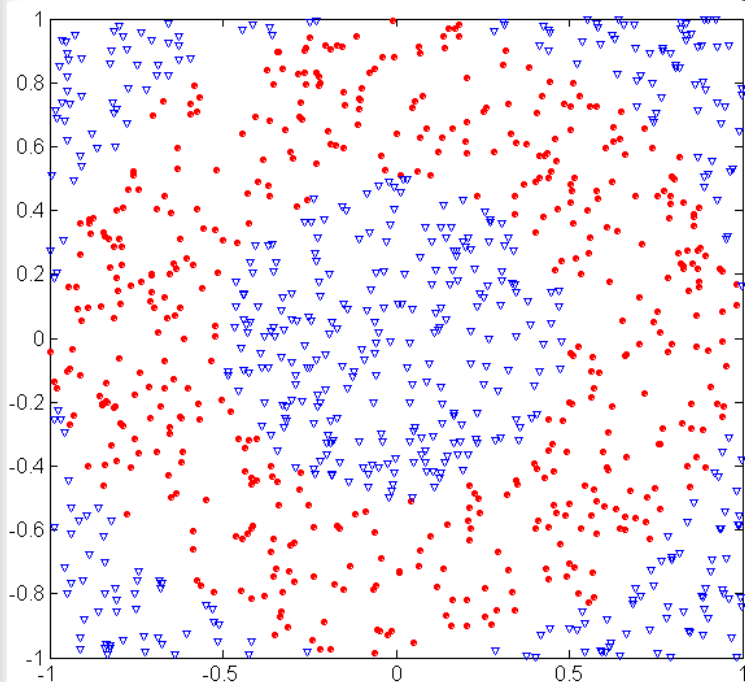
Cost = 4255

# Errori di classificazione

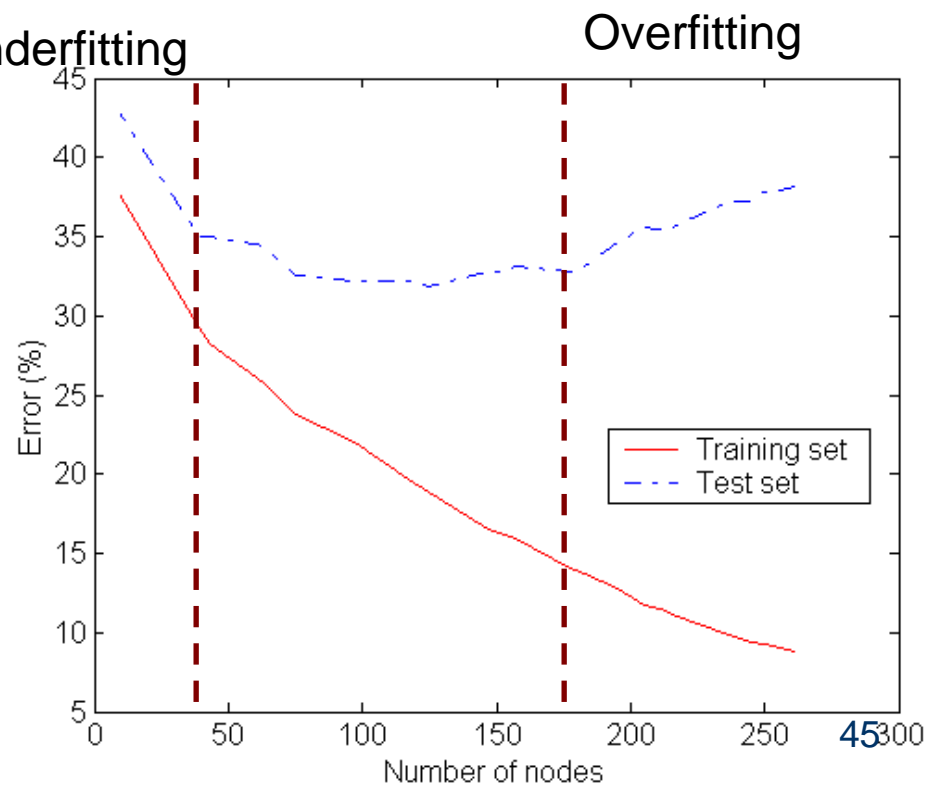
- **Training error:** sono gli errori che si commettono sul training set
- **Generalization error:** sono gli errori che si commettono sul test set (record su cui **non** è stato addestrato il sistema).
- **Underfitting:** il modello è troppo semplice e non consente una buona classificazione nè del training set, nè del test set
- **Overfitting:** il modello è troppo complesso, consente un'ottima classificazione del training set, ma una pessima classificazione del test set
  - ✓ Il modello non riesce a generalizzare poiché è basato su peculiarità specifiche del training set che non si ritrovano nel test set (es. rumore presente nel training set)

# Underfitting e Overfitting

- ✓ 500 cerchi e 500 triangoli
- ✓ Punti circolari:  $0.5 \leq \sqrt{x_1^2 + x_2^2} \leq 1$
- ✓ Punti triangolari:  $\sqrt{x_1^2 + x_2^2} > 0.5$  o  $\sqrt{x_1^2 + x_2^2} < 1$

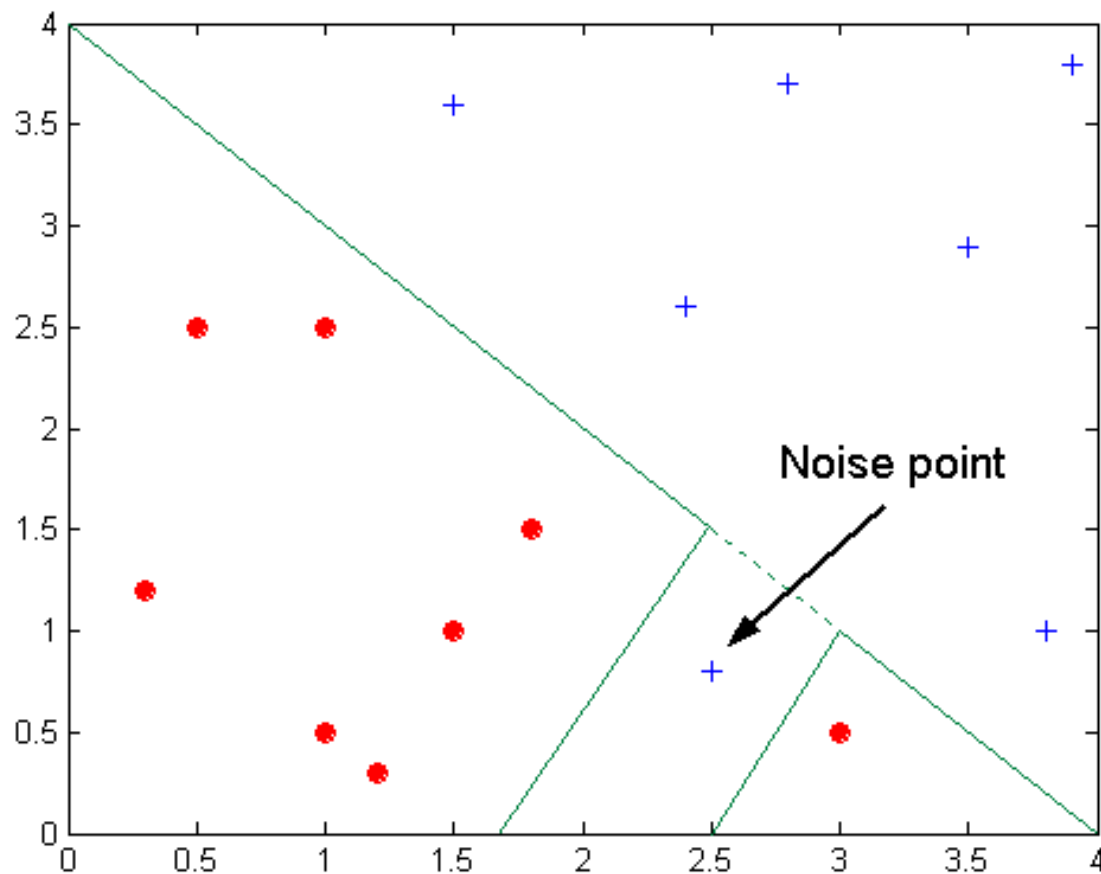


Underfitting



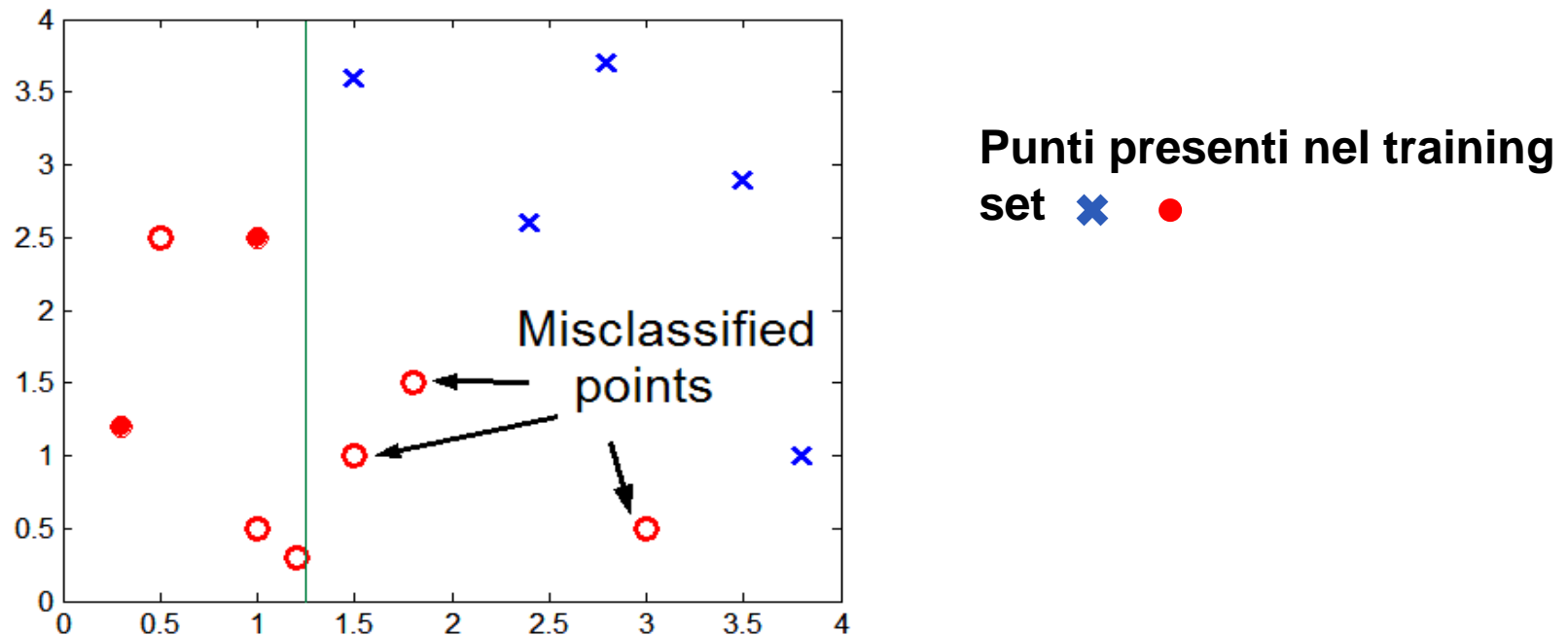
Overfitting

# Overfitting dovuto al rumore



- I confini delle aree sono distorte a causa del rumore

# Overfitting dovuto alla ridotta dimensione del training set



- La mancanza dei punti nella parte bassa del diagramma rende difficile individuare una corretta classificazione per quella porzione di regione



# Come gestire l'Overfitting: prepruing (Early stopping rule)

- Interrompere lo splitting prima che si arrivi a un albero di massima profondità
- Un nodo non può essere splittato ulteriormente se:
  - ✓ Il nodo non contiene istanze
  - ✓ Tutte le istanze appartengono alla medesima classe
  - ✓ Tutti gli attributi hanno gli stessi valori
- Condizioni più restrittive potenzialmente adottabili sono:
  - ✓ Interrompi lo splitting se il numero di istanze nel nodo è inferiore a una quantità fissata
  - ✓ Interrompi lo splitting se la distribuzione delle istanze tra le classi è indipendente dai valori degli attributi
  - ✓ Interrompi lo splitting se non si migliora la misura di purezza (es. Gini o information gain).





# Come gestire l'Overfitting: post-pruning (reduced error pruning)

- Esegui tutti gli split possibili
- Esamina i nodi del decision tree ottenuto con una logica bottom-up
- Collassa un sottoalbero in un nodo foglia se questo permette di ridurre l'errore di generalizzazione (ossia sul validation set)
  - ✓ Scegli di collassare il sottoalbero che determina la massima riduzione di errore (N.B. scelta greedy)
- Le istanze nella nuova foglia possono essere etichettate
  - ✓ In base all'etichetta che compare più frequentemente nel sottoalbero
  - ✓ In base all'etichetta che compare più frequentemente nelle istanze del training set che appartengono al sottoalbero
- Il post-pruning è più efficace ma implica un maggior costo computazionale
  - ✓ Si basa sull'evidenza del risultato di un albero completo



# Elementi caratterizzanti

- A parte la logica di base per definire completamente un algoritmo per la costruzione di alberi decisionali è necessario definire:
  - ✓ La condizione di split
  - ✓ Il criterio che definisce lo split migliore
  - ✓ Il criterio per interrompere lo splitting
  - ✓ **Le modalità per valutare la bontà di un albero decisionale**

# Costruzione del test set

## ■ Holdout

- ✓ Utilizzare  $2/3$  dei record per il training e  $1/3$  per la validazione
- ✓ Svantaggi:
  - Opera con un training set ridotto
  - Il risultato dipende dalla composizione del training set e del test set

## ■ Random subsampling

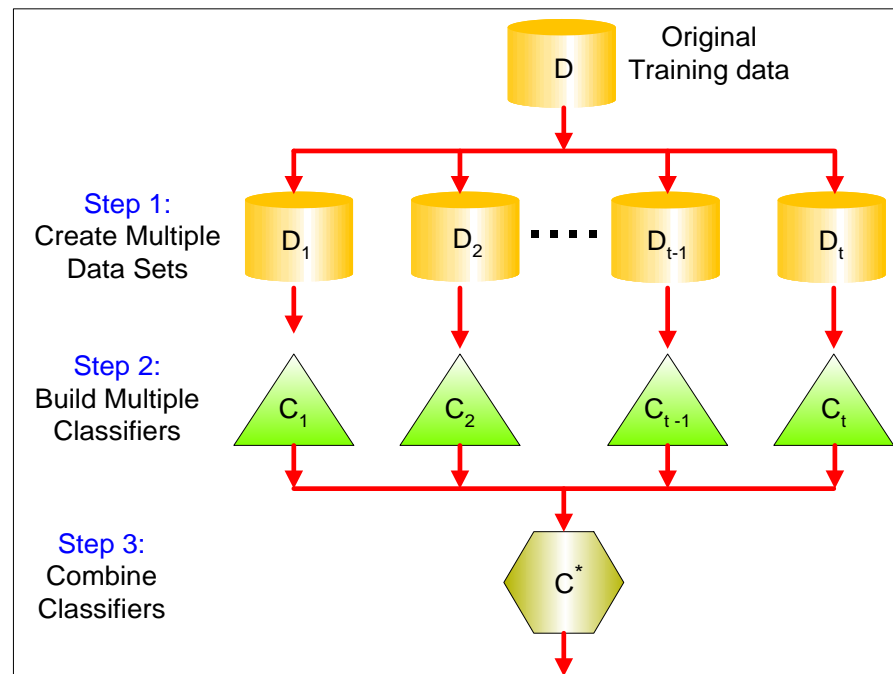
- ✓ Consiste in una esecuzione ripetuta del metodo holdout in cui il dataset di training è scelto casualmente

## ■ Cross validation

- ✓ Partiziona i record in  $k$  sotto-insiemi distinti
- ✓ Esegui il training su  $k-1$  partizioni ed il test sulla rimanente
- ✓ Ripeti il test  $k$  volte e calcola l'accuracy media
- ✓ **ATTENZIONE:** la cross validation crea  $k$  classificatori diversi e quindi la validazione indica quanto il tipo di classificatore e i suoi parametri sono adatti per lo specifico problema
  - I  $k$  alberi decisionali costruiti potrebbero avere attributi e condizioni di split diverse a seconda delle caratteristiche del  $k$ -esimo training set

# Combinazione di classificatori

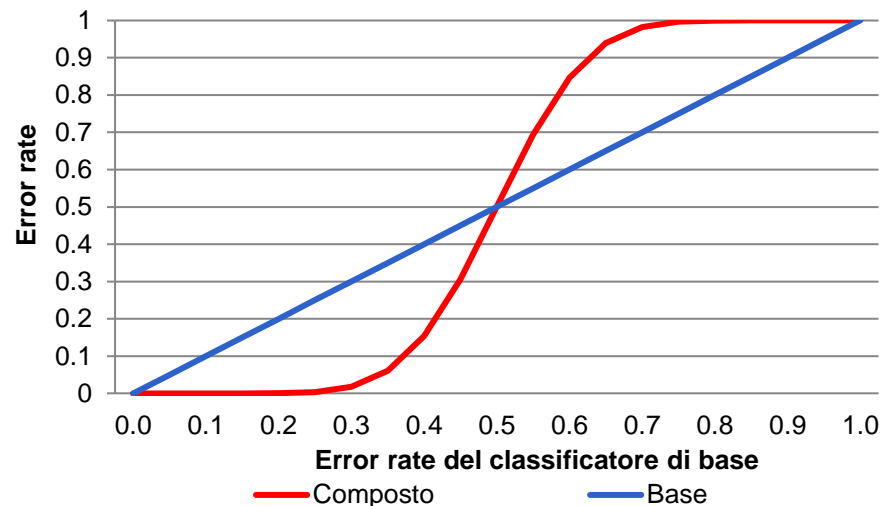
- Idea: costruire più classificatori di base e predire la classe di appartenenza di un record aggregando le classificazioni ottenute
  - ✓ Il risultato del classificatore composto è definito per mezzo di una funzione che, per esempio, assegna il record alla classe che è stata “*votata*” dal maggior numero di classificatori



# Principio di funzionamento

- Supponiamo di avere 25 classificatori semplici
  - ✓ Ogni classificatore ha un error-rate pari a  $\varepsilon = 0.35$
  - ✓ Si assuma che i classificatori siano indipendenti
    - Non c'è correlazione tra gli error-rate dei classificatori
- La probabilità che il classificatore composto dia un risultato errato è:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1 - \varepsilon)^{25-i} = 0.06$$



- Condizioni necessarie perchè il classificatore composto dia risultati migliori dei classificatori semplici sono:
  - ✓ Che i classificatori siano indipendenti
  - ✓ Che l'error-rate del singolo classificatore sia inferiore a 0.5

# Come costruire classificatori composti

- **Cambiando il training set:** si costruiscono più training set a partire dal data set dato
  - ✓ Bagging e Boosting
- **Cambiando gli attributi utilizzati:** i singoli classificatori sono basati su un sottoinsieme degli attributi
  - ✓ Utile quando gli attributi sono fortemente ridondanti
    - Random Forest
- **Cambiando le classi considerate:**
  - ✓ Si partizionano le classi in due gruppi A0 e A1 e si trasforma il problema dato in un problema binario. Le classi che appartengono ad A0 sono classificate come 0 le rimanenti come 1.
  - ✓ I diversi classificatori sono costruiti suddividendo le classi in sottoinsiemi diversi
  - ✓ La classificazione del classificatore composto si ottiene incrementando di 1 il punteggio delle classi che appartengono al sottoinsieme scelto.
  - ✓ Il record è infine assegnato alla classe che ottiene il punteggio maggiore
    - Error-Correcting Output Coding
- **Cambiando i parametri dell'algoritmo di learning:**
  - ✓ Topologia e pesi di una rete neurale
  - ✓ Alberi decisionali con politiche di scelta random degli attributi da utilizzare



# Bagging vs Boosting vs ...

- **Bagging:** mira a creare un insieme di classificatori aventi la stessa importanza. All'atto della classificazione, ciascun modello voterà circa l'esito della predizione e l'output complessivo sarà la classe che avrà ricevuto il maggior numero di voti.
- **Boosting:** a differenza del bagging, ciascun classificatore influisce sulla votazione finale con un certo peso. Tale peso sarà calcolato in base all'errore di accuratezza che ciascun modello commetterà in fase di learning.
- **Stacking:** mentre nel bagging l'output era il risultato di una votazione, nello stacking viene introdotto un ulteriore classificatore (detto meta-classificatore) che utilizza le predizioni di altri sotto-modelli per effettuare un ulteriore learning.

# Bagging

- Permette di costruire classificatori composti che associano un evento alla classe più votata dai classificatori base
- Ogni classificatore è costruito mediante **bootstrap** di un medesimo training set

```
// k = numero di cicli di bootstrap N = card. del training set  
//  $\delta()$ =1 se l'argomento della funzione è TRUE, 0 altrimenti
```

```
for i=1 to k do
```

```
  Crea un training set  $D_i$  di dimensione N
```

```
  Allena un classificatore  $C_i$  sul training set  $D_i$ 
```

```
end for
```

$$C^*(\mathbf{x}) = \arg \max_y \sum_i \delta(C_i(\mathbf{x}) = y)$$



# Bagging: un esempio

- Classificatore di base: albero decisionale binario a un livello
  - ✓ Può solo fare scelte del tipo  $x \leq s \rightarrow -1$   $x > s \rightarrow 1$  dove  $s$  è lo split point

- Il data set

$x$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
$y$	1	1	1	-1	-1	-1	-1	1	1	1

- L'accuratezza del classificatore base non può superare 70%
  - ✓  $x \leq 0.3 \rightarrow 1$   $x > 0.3 \rightarrow -1$
  - ✓  $x \leq 0.7 \rightarrow -1$   $x > 0.7 \rightarrow 1$

# Bagging: le sessioni

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 0.35 \implies y = 1$

$x > 0.35 \implies y = -1$

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

$x \leq 0.65 \implies y = 1$

$x > 0.65 \implies y = 1$

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.35 \implies y = 1$

$x > 0.35 \implies y = -1$

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.3 \implies y = 1$

$x > 0.3 \implies y = -1$

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

$x \leq 0.35 \implies y = 1$

$x > 0.35 \implies y = -1$

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \implies y = -1$

$x > 0.75 \implies y = 1$

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

$x \leq 0.75 \implies y = -1$

$x > 0.75 \implies y = 1$

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \implies y = -1$

$x > 0.75 \implies y = 1$

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \implies y = -1$

$x > 0.75 \implies y = 1$

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

$x \leq 0.05 \implies y = -1$

$x > 0.05 \implies y = 1$

# Bagging: i risultati

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

- Il bagging permette di ottenere il comportamento di un albero decisionale a due livelli



*Disegnare l'albero decisionale a due livelli  
corrispondente al risultato del bagging*

# Random Forest

- Appartiene alla famiglia di metodi di Bagging:
  - ✓ Estrazione con re-imbussolamento di un sottoinsieme  $D_i$  di pattern dal training set
  - ✓ Addestramento del classificatore  $C_i$  sul sottoinsieme  $D_i$
  - ✓ Fusione dei classificatori (e.g., majority vote rule, somma)
- In Random Forest i singoli classificatori sono alberi decisionali
- Per rendere maggiormente indipendenti i classificatori:
  - ✓ per ogni nodo la scelta della feature migliore su cui partizionare non è fatta sull'intero insieme delle  $d$  feature (dimensionalità dei pattern), ma su un sottoinsieme random di  $d'$  feature. Valore tipico  $d' = \sqrt{d}$
  - ✓ In assenza di questo accorgimento (noto anche come feature bagging) molti tree sceglierebbero con elevata probabilità le stesse variabili (quelle più discriminanti).
- Pertanto Random Forest opera simultaneamente due tipi di bagging: uno sui pattern del training set e uno sulle feature

# Classificatori K-NN



Prof. Matteo Golfarelli

Alma Mater Studiorum - Università di Bologna

# Classificatori Instance-Based

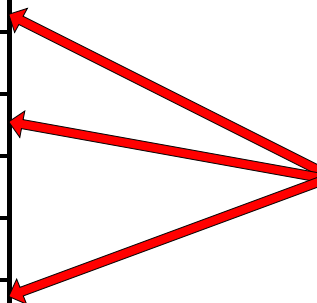
- Non costruiscono modelli ma classificano i nuovi record sulla base della loro somiglianza rispetto agli esempi nel training set
- Sono per questo detti **lazy** -pigri- **learners** in contrapposizione agli **eager** –diligenti, impazienti- **learners** (rule based, alberi decisionali, reti neurali, ecc.)
  - ✓ Rote-learner: classifica un record solo se coincide con uno del training set
  - ✓ Nearest-Neighbor: classifica il record in base ai più simili del training set

Set of Stored Cases

Atr1	.....	AtrN	Class
			A
			B
			B
			C
			A
			C
			B

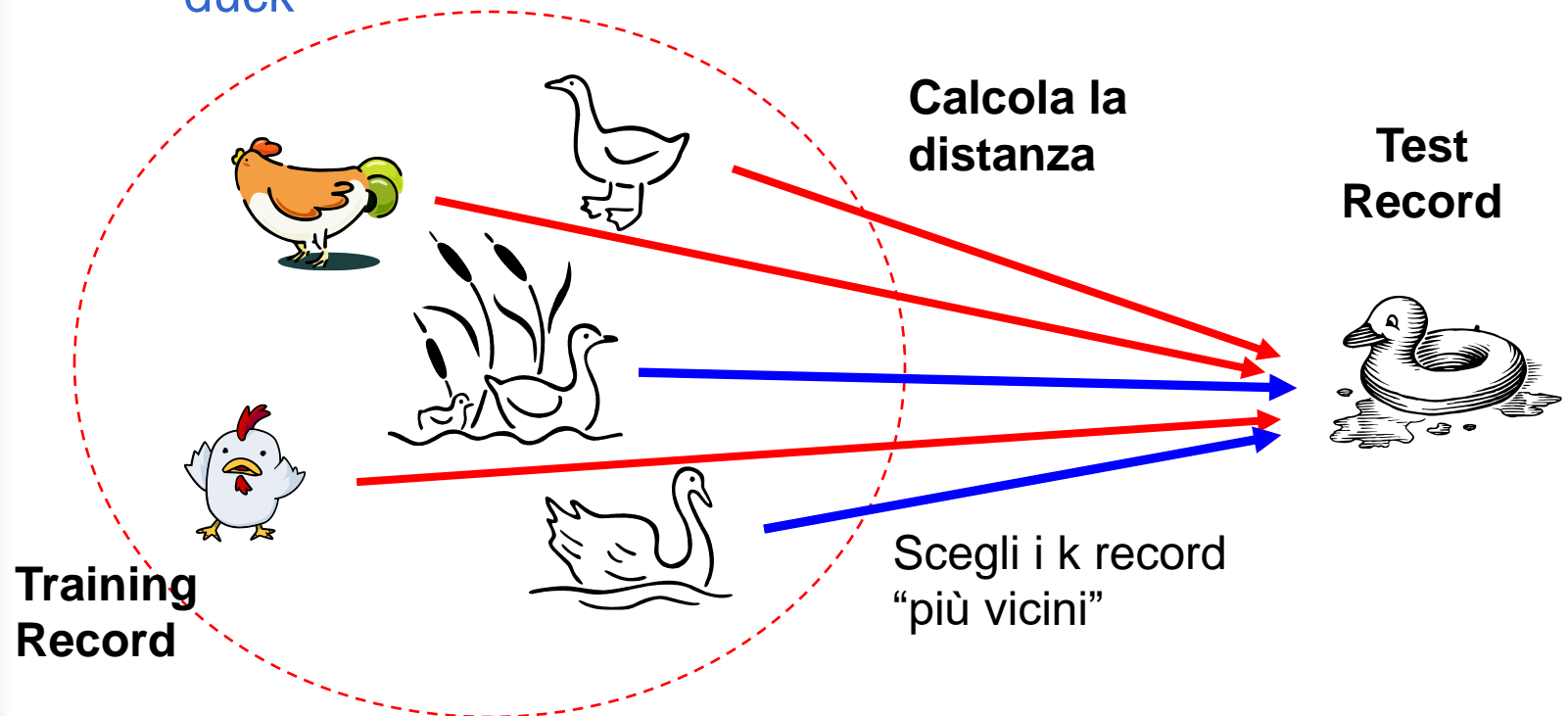
Unseen Case

Atr1	.....	AtrN



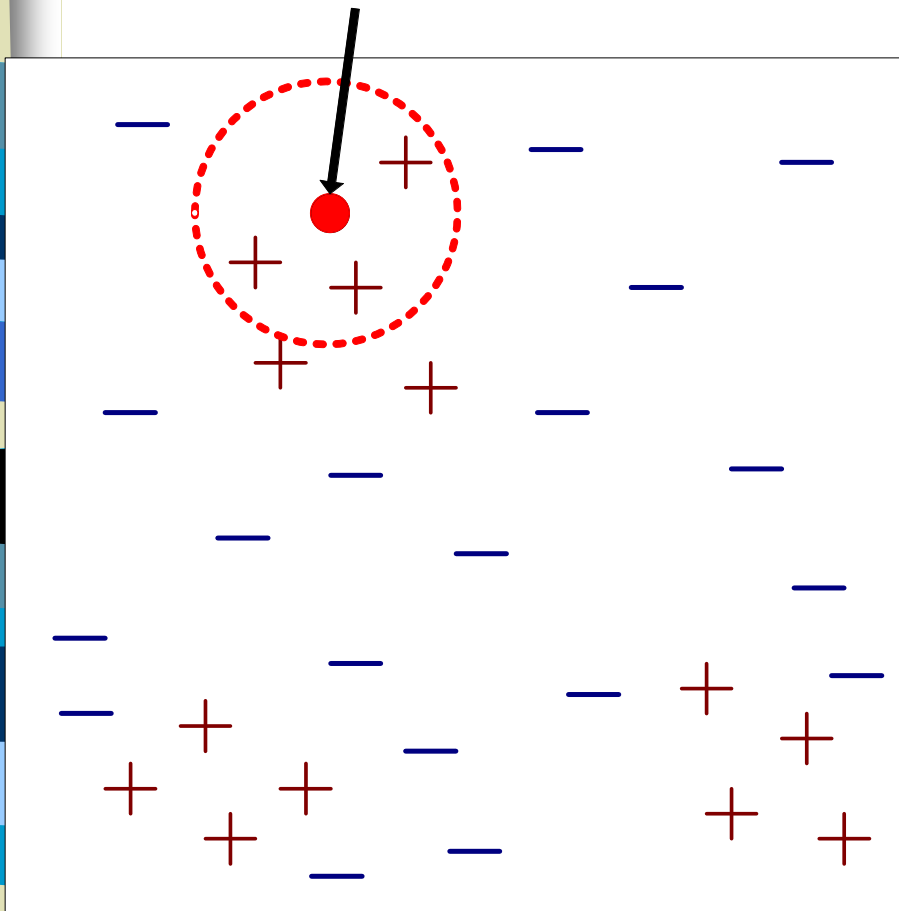
# Classificatori Nearest Neighbor

- Utilizzano i  $k$  punti “più vicini” (nearest neighbors) per effettuare la classificazione
- Idea di base:
  - ✓ If it walks like a duck, quacks like a duck, then it's probably a duck



# Classificatori Nearest Neighbor

Record da classificare



Richiedono:

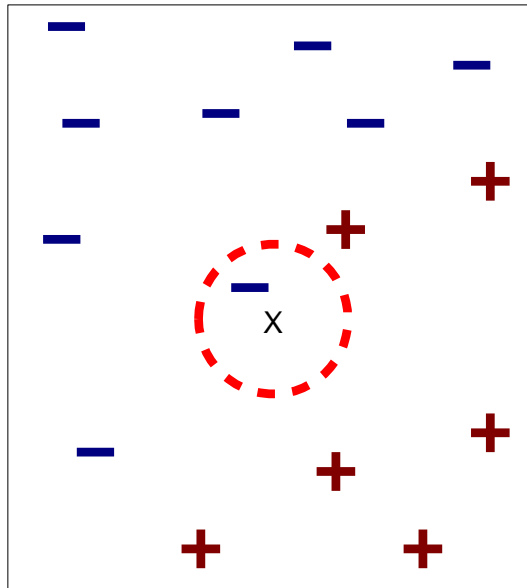
- Un training set
- Una metrica per calcolare la distanza tra i record
- Il valore di  $k$ , ossia il numero di vicini da utilizzare

Il processo di classificazione:

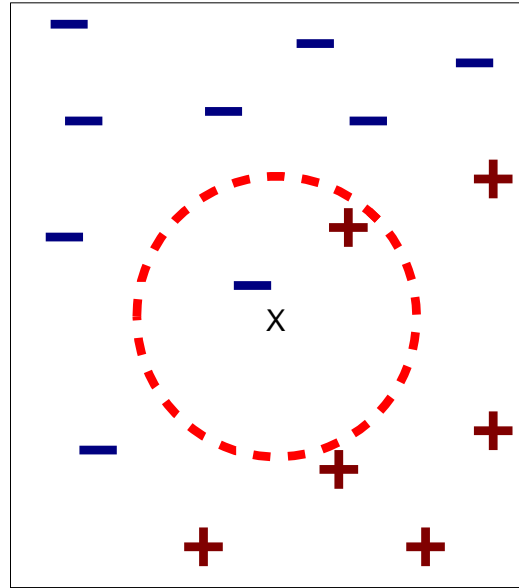
- Calcola la distanza rispetto ai record nel training set
- Identifica  $k$  nearest neighbors
- Utilizza le label delle classi dei nearest neighbor per determinare la classe del record sconosciuto (es. scegliendo quella che compare con maggiore frequenza)



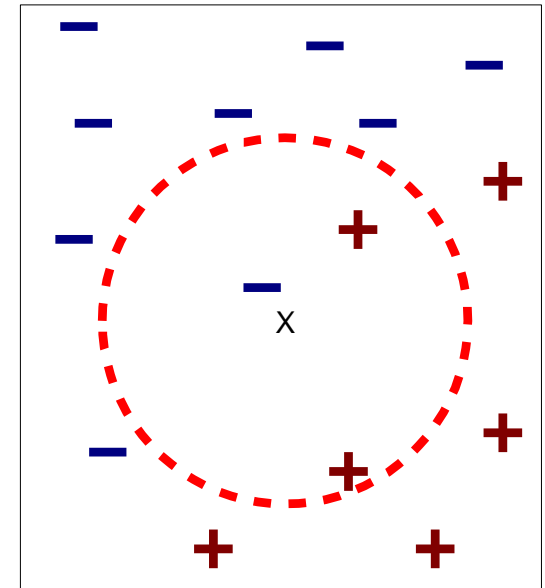
# Definizione di Nearest Neighbor



(a) 1-nearest neighbor



(b) 2-nearest neighbor



(c) 3-nearest neighbor

I  $k$  nearest neighbors di un record  $x$  sono i record del training set che hanno le più piccole  $k$  distance da  $x$

# K-Nearest Neighbor

- La classificazione di un record  $\mathbf{z}$  è ottenuta con un processo di *majority voting* tra i  $k$  elementi  $D_z$  del training set  $D$  più vicini (o simili) a  $\mathbf{z}$

$$\bar{y} = \arg \max_{y \in \mathbf{Y}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(y_i = y)$$

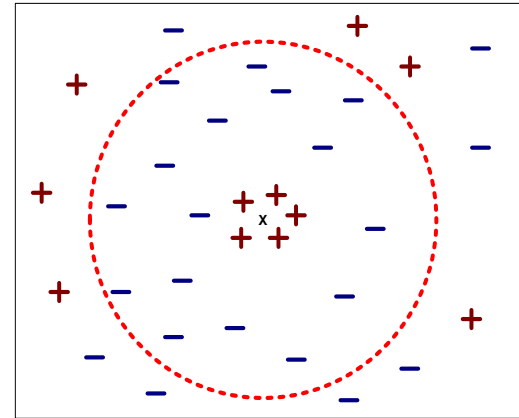
- ✓  $\mathbf{Y}$  è l'insieme delle label di classe
- ✓  $I()$  restituisce 1 se il suo argomento è TRUE, 0 altrimenti

- Tutti i vicini hanno lo stesso peso
  - ✓ L'algoritmo è molto sensibile al valore di  $k$
  - ✓ Questo rischio può essere ridotto pesando il contributo dei vicini in base alla distanza  $w = 1/d(\mathbf{z}, \mathbf{x}_i)$

$$\bar{y} = \arg \max_{y \in \mathbf{Y}} \sum_{(\mathbf{x}_i, y_i) \in D_z} w \times I(y_i = y)$$

# K-Nearest Neighbor

- La scelta di  $k$  è importante perchè:
  - ✓ Se  $k$  è troppo piccolo, l'approccio è sensibile al rumore
  - ✓ Se  $k$  è troppo grande, l'intorno può includere esempi appartenenti ad altre classi
- Per operare correttamente gli attributi devono avere la stessa scala di valori e vanno quindi normalizzati in fase di pre-processing
  - ✓ Esempio: su quale attributo una differenza di 0.5 vale di più?
    - l'altezza di un adulto varia 1.5m to 2.1m
    - il peso di un adulto varia da 40kg a 150kg
    - Lo stipendio di una persona varia da 10K€ to 1M€
- Normalizzare la scala può non essere sufficiente in presenza di diverse distribuzioni dei dati
  - ✓ Mahalanobis distance





# K-Nearest Neighbor: Pro & Contro

## ■ Pro

- ✓ Non richiedono la costruzione di un modello
- ✓ Rispetto ai sistemi basati su regole o decision tree permettono di costruire “contorni” delle classi non lineari e sono quindi più flessibili

## ■ Contro

- ✓ Richiedono una misura di similarità o distanza per valutare la vicinanza
- ✓ Richiedono una fase di pre-processing per normalizzare il range di variazione degli attributi
- ✓ La classe è determinata localmente e quindi è suscettibile al rumore dei dati
- ✓ Sono molto sensibili alla presenza di attributi irrilevanti o correlati che falseranno le distanze tra gli oggetti
- ✓ Il costo di classificazione può essere elevato e dipende linearmente dalla dimensione del training set in mancanza di opportune **strutture ad indice**

# AutoMachineLearning

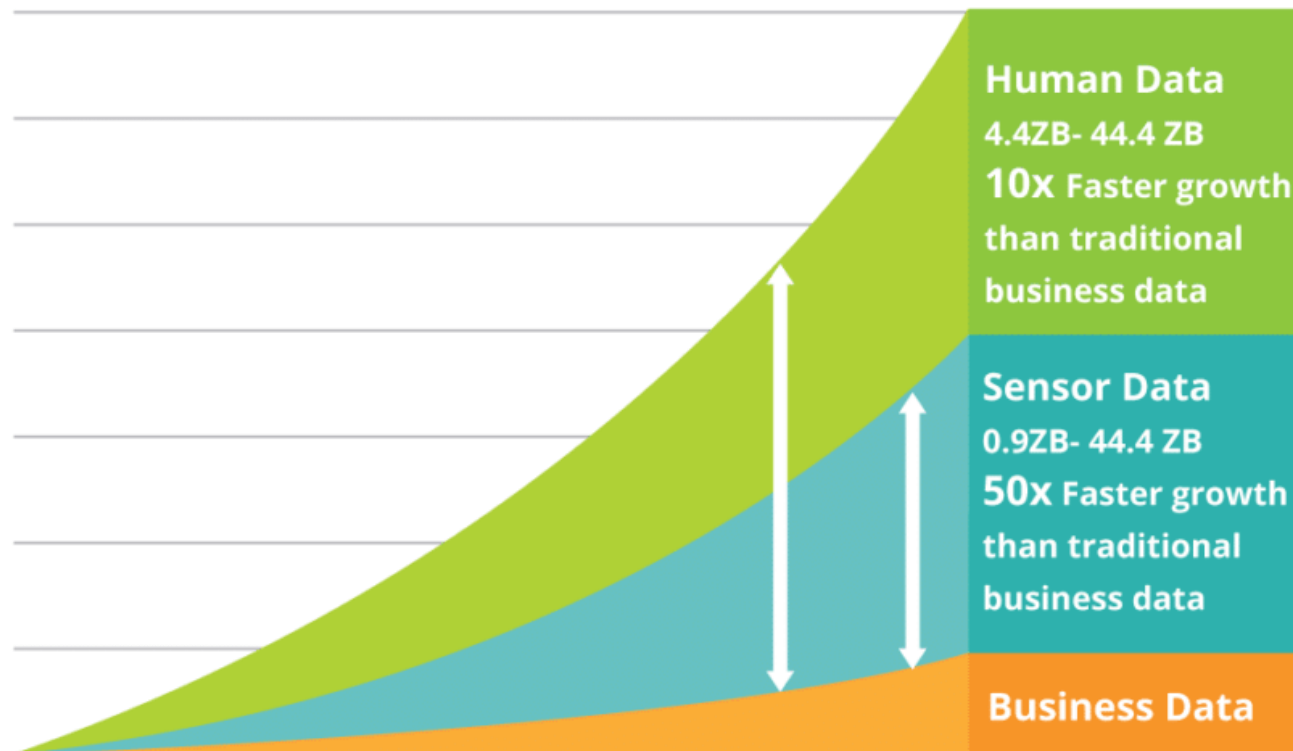


Prof. Matteo Golfarelli

Alma Mater Studiorum - Università di Bologna

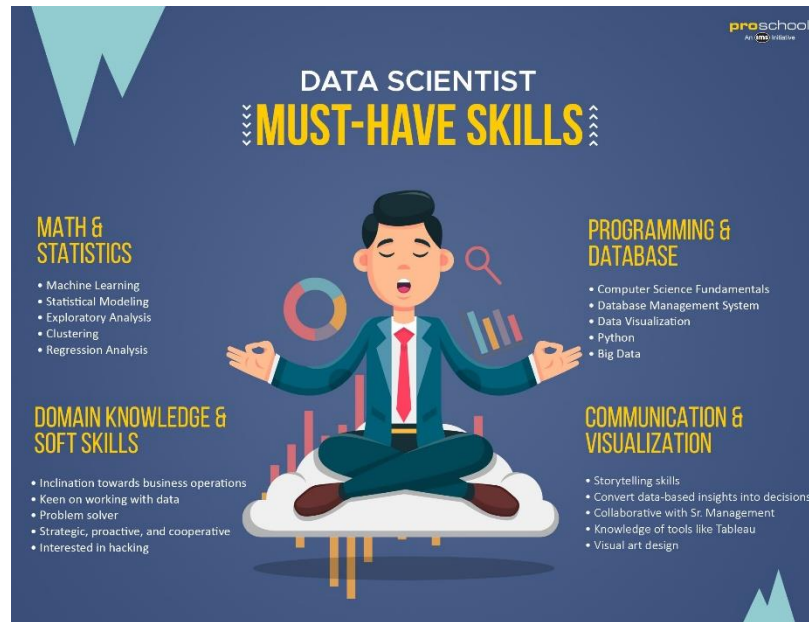
# La Crescita dei dati

- La crescita esponenziale dei dati rende difficili analizzarli manualmente, ma....
  - ✓ Anche le tecniche di ML richiedono un processo di tuning oneroso
  - ✓ I data scientist non scalano!



# Data scientist & Data Enthusiast

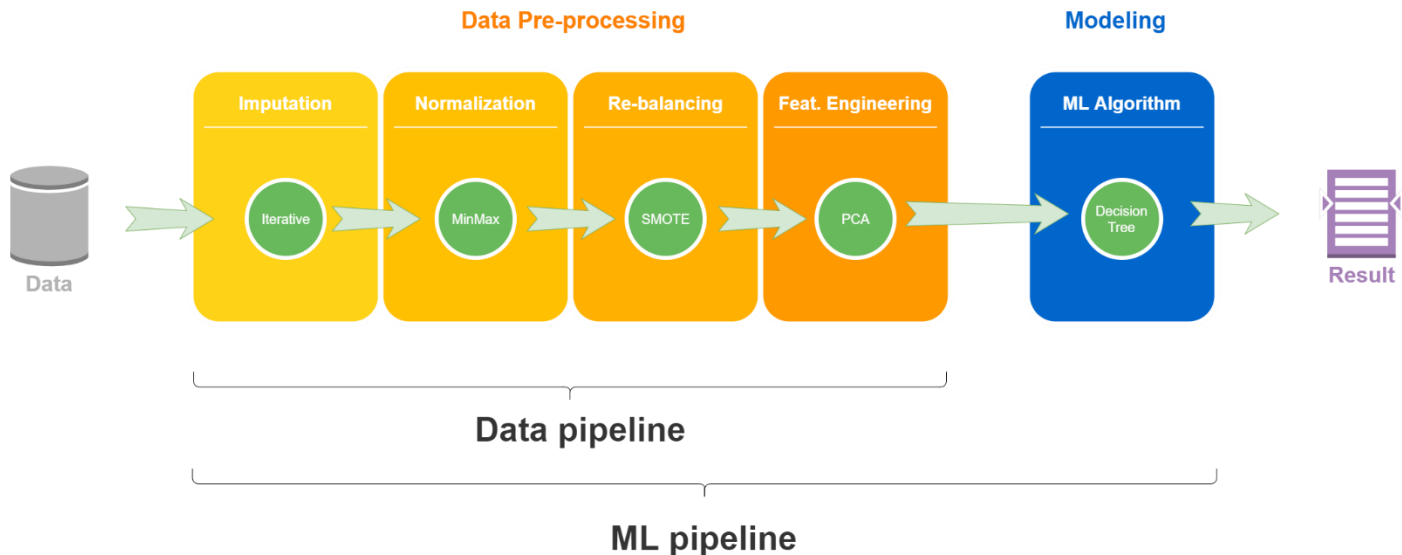
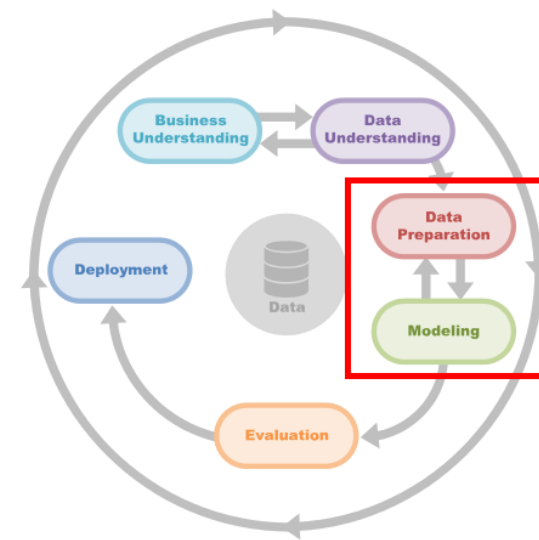
- I data scientist sono una delle figure professionali più ricercate del momento
  - ✓ Sono figure professionali interdisciplinari difficili da formare



- Sempre più utenti con competenze di base utilizzano tool di Data mining

# AutoML

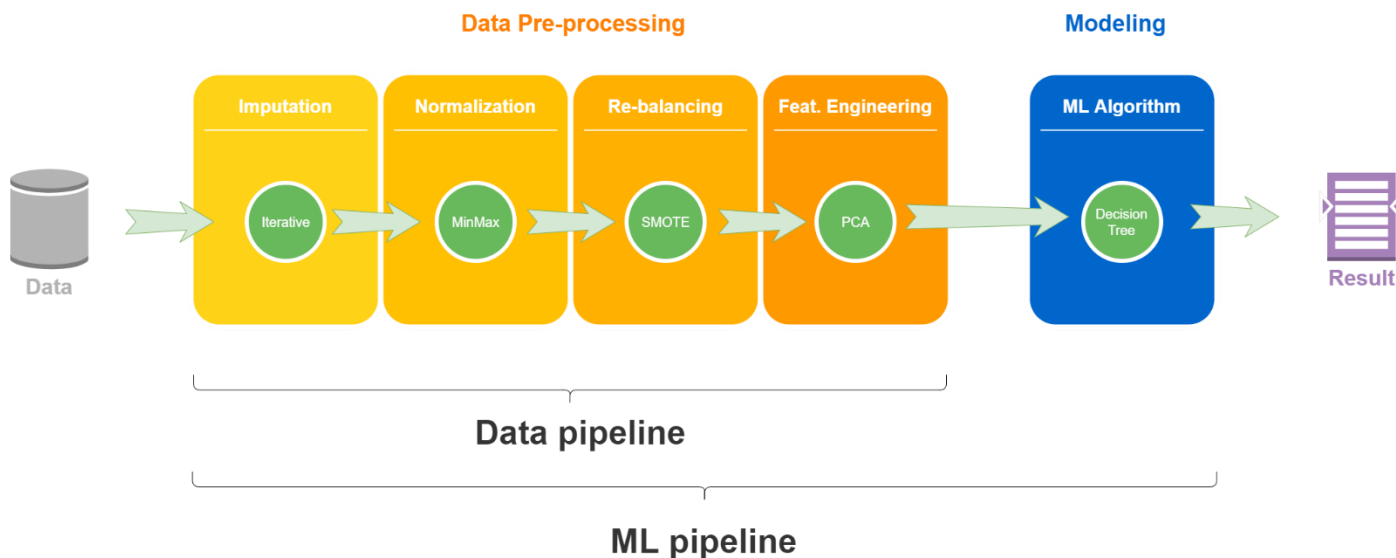
- Automated Machine Learning è il processo di automazione nell'applicazione del ML
- Ha l'obiettivo di identificare la pipeline ottima
  - ✓ Una pipeline è formata da una sequenza di trasformazioni
  - ✓ Ogni trasformazione può essere scelta tra un insieme di alternative
  - ✓ Ogni operazione ha un insieme di iper-parametri
  - ✓ Ogni iper-parametro ha il suo search space





# Parametri vs Iper-Parametri

- In un sistema di machine learning ci sono due tipi di parametri:
- **Parametri del modello**: sono i parametri che sono individuati dal processo di allenamento che porta al fitting.
  - ✓ Decision tree: attributi di split, condizioni di split
- **Iper-parametri**: Questi sono parametri regolabili che devono essere fissati per ottenere un modello con prestazioni ottimali.
  - ✓ K-nearest neighbor: k, funzione distanza...
  - ✓ Decision tree: split a 2-vie o a molte-vie, funzione di purezza...
  - ✓ PCA: numero dimensioni

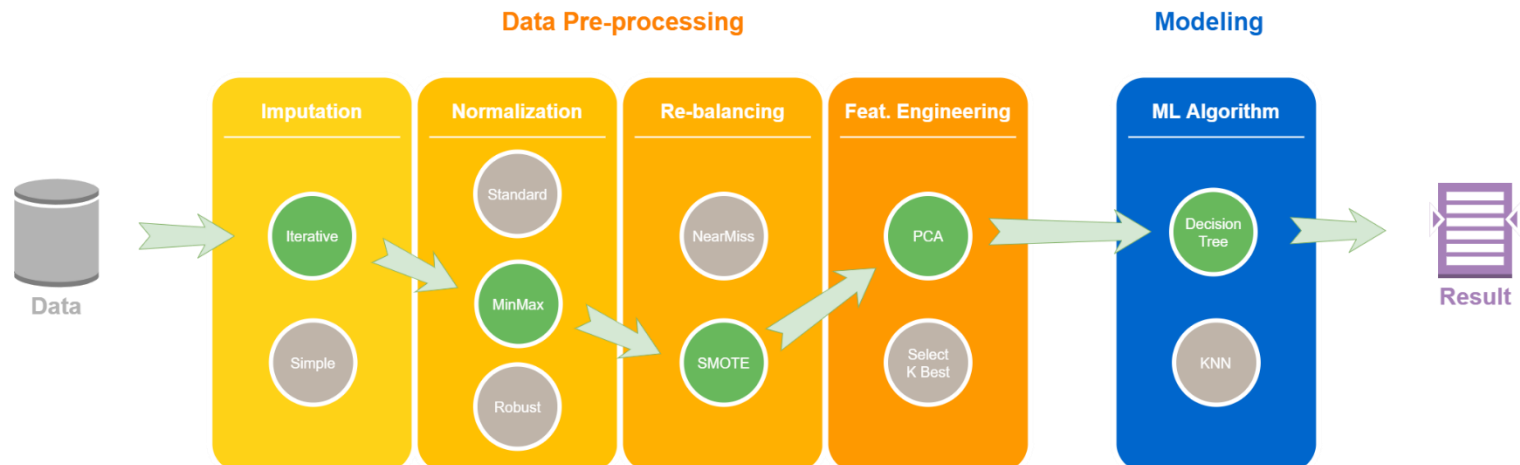


# CASH: Combined Algorithm Selection and Hyper-parameter optimization problem

- Formalizzato per un singolo operatore la prima volta in Auto-Weka
  - ✓ Dato un dataset  $D$  suddiviso in  $D_{\text{train}}$  e  $D_{\text{test}}$
  - ✓ Un insieme di algoritmi  $A = \{A_1, \dots, A_j, \dots, A_n\}$  con gli associati iper-parametri  $\{\Theta_1, \dots, \Theta_j, \dots, \Theta_n\}$

$A_1 = \text{Decision tree}$   
 $\Theta_1 = \{\text{num\_obj}, \text{pruning}\}$   
     $\text{num\_obj} = \{2, 3, 4\}$   
     $\text{pruning} = \{\text{true}, \text{false}\}$

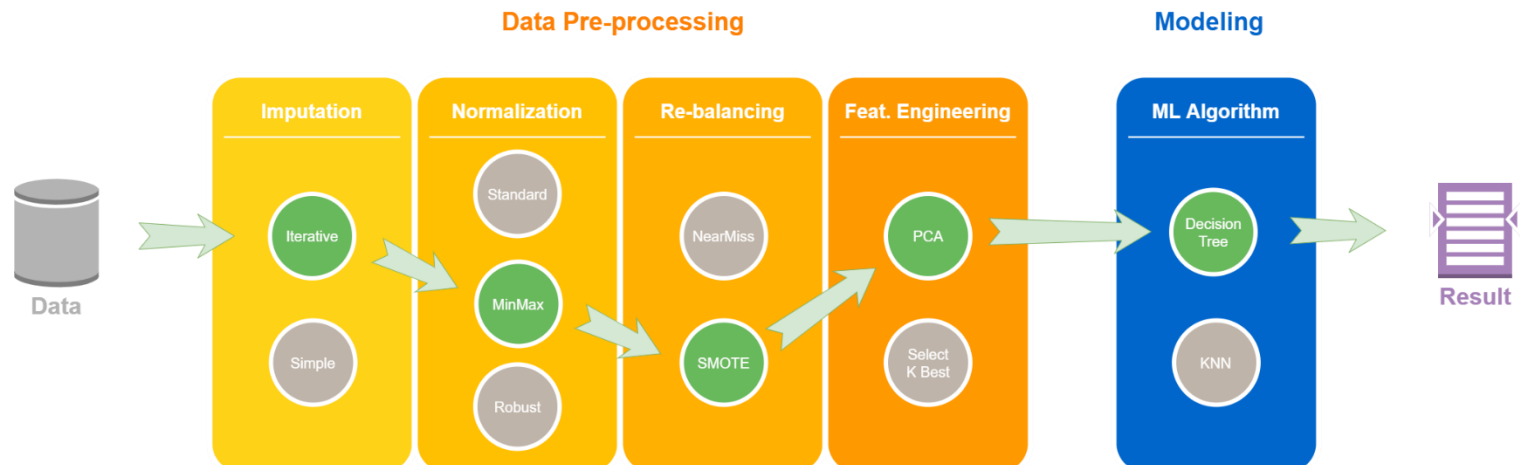
$A_2 = \text{KNN}$   
 $\Theta_2 = \{K, \text{weight}\}$   
     $K = \{2, 3, 4, 5\}$   
     $\text{weight} = \{1/\text{dist}, 1/\text{dist}^2\}$



# CASH: Combined Algorithm Selection and Hyper-parameter optimization problem

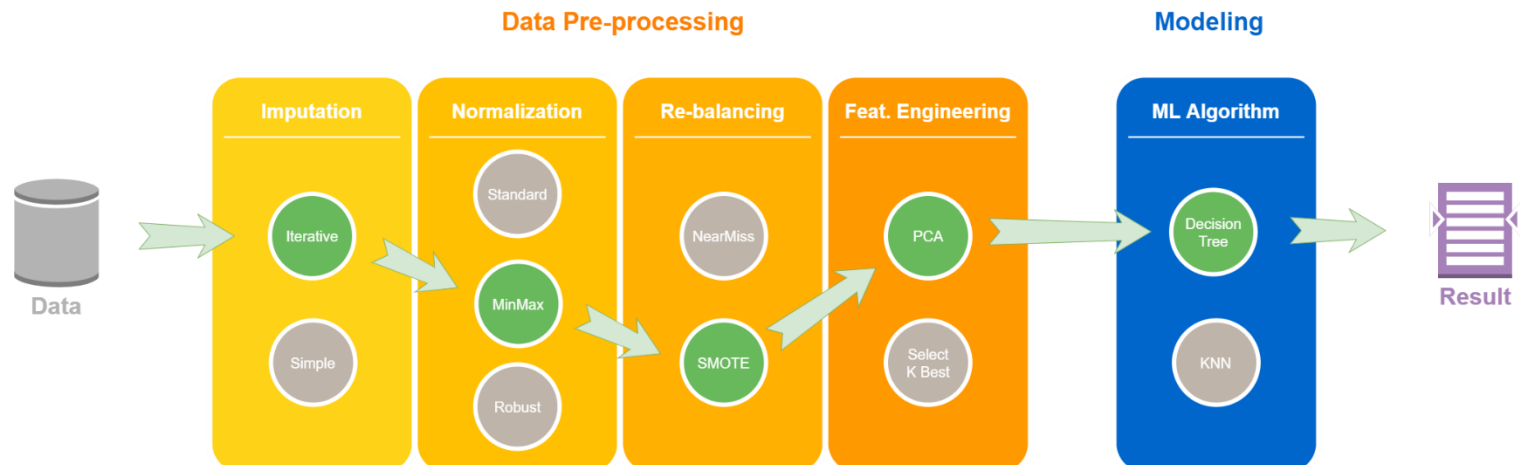
- Formalizzato per un singolo operatore la prima volta in Auto-Weka
  - ✓ Dato un dataset  $D$  suddiviso in  $D_{\text{train}}$  e  $D_{\text{test}}$
  - ✓ Un insieme di algoritmi  $A = \{A_1, \dots, A_j, \dots, A_n\}$  con gli associati iper-parametri  $\{\Theta_1, \dots, \Theta_j, \dots, \Theta_n\}$
  - ✓ Una metrica di valutazione  $M(A_j, D_{\text{train}}, D_{\text{test}})$ 
    - Accuracy, Precision-Recall
- Trova l'insieme di algoritmi e di iper-parametri che massimizzano la metrica di valutazione

$$A_{\theta^*}^* = \underset{A_j \in A, \theta \in \Theta_j}{\operatorname{argmax}} M(A_j, D_{\text{train}}, D_{\text{test}})$$



# CASH: Combined Algorithm Selection and Hyper-parameter optimization problem

- Formalizzato per un singolo operatore la prima volta in Auto-Weka
  - ✓ Dato un dataset  $D$  suddiviso in  $D_{\text{train}}$  e  $D_{\text{test}}$
  - ✓ Un insieme di algoritmi  $A = \{A_1, \dots, A_j, \dots, A_n\}$  con gli associati iper-parametri  $\{\Theta_1, \dots, \Theta_j, \dots, \Theta_n\}$
  - ✓ Una metrica di valutazione  $M(A_j, D_{\text{train}}, D_{\text{test}})$ 
    - Accuracy, Precision-Recall
- Trova l'insieme di algoritmi e di iper-parametri che massimizzano la metrica di valutazione
$$A_{\theta^*}^* = \underset{A_j \in A, \theta \in \Theta_j}{\operatorname{argmax}} M(A_j, D_{\text{train}}, D_{\text{test}})$$
- La formulazione precedente è valida per un operatore e va estesa a tutta la pipeline





# Risoluzione di CASH

- Lo spazio di ricerca può essere enorme sono necessarie tecniche euristiche (che trovino una soluzione ottimale senza esplorarlo tutto)
  - ✓ Grid search
  - ✓ Random search
  - ✓ Heuristics
    - Ant colony optimization
    - Particle Swarm Optimization
    - Simulate Annealing
  - ✓ Genetic algorithms
  - ✓ Multi-resolution optimization
    - Successive Halving
    - Hyper-Band
  - ✓ Bayesian optimization
    - Sequential Model-Based Optimization (SMBO)



# AutoML Tool

- Il numero di tool che forniscono funzioni di AutoML è in continua crescita
  - ✓ **Cloud-Based**
    - Google AutoML
    - Amazon AutoML
    - Azure AutoML
    - Data Iku
    - Data Robot
  - ✓ **Distributed**
    - MLBase
    - TrasmogrifAI
    - MLBox
    - ATM
    - Rafiki
  - ✓ **Centralised**
    - Auto-Weka
    - Auto-Sklearn
    - HyperOpt
    - HyperOpt-Sklearn
    - TPOT
    - SmartML
    - H2O