Classification

A questo punto del corso, sorge spontanea la domanda: "**Cosa possiamo farci con i dati, una volta che li abbiamo integrati?**" In questa seconda parte del corso, quindi, affronteremo il concetto di **Data Analytics**, ovverosia **l'analisi dei dati per estrarre nuova conoscenza** ed in particolare, ci concentreremo su due aspetti fondamentali, ovvero:

- Classificazione;
- Clusterizzazione.

Ci concentriamo su queste due operazioni (ovvero la classificazione e la clusterizzazione), perchè sono due attività che ricorrono moltissimo ai dati, in quanto:

- la **clusterizzazione** è la capacità di dividere i dati in gruppi (quindi, davanti ad una popolazione, siamo in grado di capire i gruppi in base, ad esempio, al comportamento individuale delle persone e a fronte delle differenze minime nei loro comportamenti individuali, siamo comunque in grado di individuare delle caratteristiche comuni e di conseguenza siamo in grado di individuare dei gruppi);
- la **classificazione** è la capacità di assegnare degli oggetti e/o degli individui a dei gruppi predeterminati (ad esempio, essere in grado di classificare una persona in: risparmiatore, compratore d'impulso oppure compratore oculato).

Iniziamo ad analizzare in maniera più approfondita queste due attività ed in particolar modo, iniziamo a parlare della **Classificazione** → esistono degli algoritmi, che ci consentono di fare una classificazione, ovvero che ci consentono di assegnare ad una popolazione di individui delle classi, anche senza ricorrere a delle reti neurali (in quanto molto spesso, la classificazione è legata al concetto delle reti neurali). In generale, un problema di classificazione permette, data una serie di etichette (per le classi), di andare ad associare un'istanza del dato alla sua corrispondete etichetta. Quindi, ogni istanza del dato è rappresentata da:

- un **insieme di attributi** (che possiamo indicare con **X**);
- l'**etichetta** della classe per l'etichetta (che possiamo indicare con **Y**).

💡 Possiamo, quindi, dire che: **Ogni istanza di dato è caratterizzata dalla tupla (X, Y)**.

Vogliamo, allora, riuscire a costruire un modello, che a fronte di una nuova istanza di dato, ci consente di capire qual è l'etichetta che dobbiamo assegnare all'istanza e di conseguenza, **nella classificazione lo scopo è duplice**:

- da un lato, la classificazione ci serve per descrivere e capire le istanze che abbiamo a disposizione. Quindi, possiamo utilizzare la classificazione per descrivere i dati che abbiamo raccolto (ad esempio, identificare il motivo per cui alcune istanze condividono la stessa etichetta);
- oppure lo scopo della classificazione può essere **predittivo**, ovverosia utilizziamo i dati storici che abbiamo a disposizione, al fine di costruire un modello che ci consente di fare delle previsioni sulle nuove istanze (che quindi non conosciamo) e per le quali non abbiamo un'etichetta.

Riassumendo, quindi, quello che vogliamo fare è di definire un modello o una funzione più o meno complessa, che **dato l'insieme degli attributi di una certa istanza, ci restituisce un'etichetta ¥ e questa classificazione è corretta, se e solo se l'etichetta che ho assegnato (tramite il modello) è effettivamente l'etichetta reale del dato** (ovvero se Y = ¥).

Esistono diversi tipi di classificazione e sicuramente la prima distinzione che dobbiamo fare è tra:

- **Classificazione binaria** → in cui abbiamo a disposizione solamente due etichette (quindi assegniamo agli individui della popolazione una tra queste due etichette disponibili);
- **Classificazione multi-classe** → in cui abbiamo a disposizione diverse etichette.

💡 Naturalmente, la classificazione multi-classe è più generale rispetto alla classificazione binaria, ma notiamo il fatto, che possiamo ridurre il caso multi-classe ad un caso binario, il quale (ovvero il caso binario) si estende in **maniera ricorsiva su se stesso**.

La classificazione, quando la facciamo per scopi predittivi (ovvero per costruire un modello, che ci consente di assegnare delle etichette a delle nuove istanze),

si differenzia da un altro problema (che noi non vediamo), ovvero il problema della **regressione**, in quanto nel caso della regressione vogliamo stimare un valore numerico, mentre nel caso della classificazione abbiamo un numero finito di etichette, le quali sono di tipo nominale.

Uno dei problemi fondamentali, quando siamo di fronte ad un problema di classificazione, è di capire come possiamo estrarre le relazioni, che ci sono tra gli attributi (che descrivono l'istanza) **e l'etichetta** → nel fare questo, la difficoltà sta nel fatto, che non tutti gli attributi potrebbero essere rilevanti e di conseguenza dovremmo essere in grado di determinare quali attributi sono più rilevanti di altri (per il nostro specifico problema di classificazione). Successivamente vedremo, che un altro importante problema è che **gli attributi potrebbero essere tra loro correlati e quindi ci potrebbero essere delle correlazioni per cui il singolo attributo non ci dà alcuna informazione, ma messo insieme ad altri, ci dà invece un'informazione molto rilevante** → capiamo, allora, che anche in questo caso, l'importanza di un attributo dipende dallo specifico problema di classificazione.

Quando si costruisce un **classificatore**, ovverosia uno strumento (che può essere un modello) che ci permette di risolvere un problema di classificazione, tipicamente abbiamo bisogno di due fasi:

1. **Fase induttiva** → nella quale si costruisce un modello di classificazione;
2. **Fase deduttiva** → in cui si applica il modello, su un insieme di istanze che non abbiamo mai visto, al fine di predirre le loro etichette.

Quindi, se il nostro problema di classificazione ha solamente uno scopo descrittivo, allora ci fermeremo alla prima fase (ovvero alla fase induttiva), mentre se successivamente vogliamo utilizzare il modello per fare delle classificazioni su istanze che non conosciamo, allora dovremmo passare alla fase successiva (ovvero la fase deduttiva).

Notiamo, inoltre, che vi sono molteplici tecniche per eseguire la classificazione e quelle che vedremo sono:

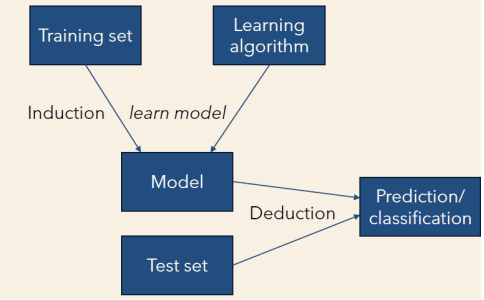
- gli **alberi decisionali** → rappresentano la tecnica più semplice per eseguire la classificazione;
- le **regole di classificazione**;
- le **regole di associazione**;
- le **reti Bayesiane;**

- le **reti neurali** → di cui noi non parleremo.

> 💡 Ognuna di queste tecniche avrà una serie di caratteristiche positive o negative nel costruire il modello, sia in termini di capacità espressive sia in termini di complessità computazionale.

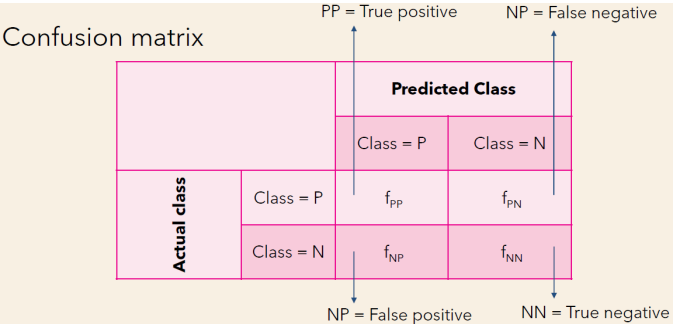
Anche se non arriveremo a parlare delle reti neurali, comunque abbiamo:

- il concetto di **Training set** → ovvero di insieme di dati attraverso cui andremo a costruire il nostro modello;
- una fase di **Learning** → che nei nostri casi non avviene attraverso una rete neurale, bensì avviene attraverso degli algoritmi. Quindi, in questa fase di Learning, andiamo ad utilizzare il Training set per costruire un modello;
- una fase di **Deduction** → in cui andiamo a prendere il modello che abbiamo costruito e lo applichiamo a dei dati di un Test set (che non abbiamo mai visto) e tentiamo di fare la nostra predizione (che in questo caso si tratta di una classificazione). Nel fare ciò, un concetto importante è quello di **generalizzazione** → in quanto, possiamo costruire un modello che va molto bene per descrivere il Training set, ma tale modello deve necessariamente anche estendersi in maniera ottimale a delle istanze che non ha visto prima (ovvero al Test set).



Per valutare le tecniche di classificazione si utilizza la **matrice di confusione** (Confusion Matrix) ed in particolar modo, tale matrice nel caso di una classificazione semplice (ovvero nel caso di una classificazione binaria) è semplicemente una matrice 2X2, dove abbiamo:

- sulle **righe** le **classi reali** (ovvero quelle che descrivono le istanze);
- sulle **colonne** le **classi predette** dal nostro modello;
- nella corrispettive celle abbiamo i corrispettivi valori: **PP, NP, NP e NN**.



Naturalmente, quando costruiamo la matrice di confusione del nostro Test set rispetto al modello che abbiamo creato, più i numeri sono elevati sulla diagonale (ovvero sul True Positive e sul True Negative) e più i numeri sono bassi sulla diagonale opposta (ovvero sul False Negative e sul False Positive), più il nostro modello sarà buono e avrà delle buone caratteristiche di generalizzazione.

Un'altra metrica di valutazione che possiamo costruire, sempre partendo dalla matrice di confusione, è l'**accuratezza** → l'accuratezza mi dice essenzialmente quanti sono il numero di previsioni corrette che abbiamo fatto, rispetto alla cardinalità del Test set. Infatti, l'accuratezza si ottiene sommando gli elementi della diagonale (ovvero True Positive e True Negative) diviso tutte le predizioni eseguite sul Test set. Formalmente, la formula dell'accuratezza è:

$$acc = (Fpp + Fnn)/(Fpp + Fpn + Fnp + Fnn)$$

> 💡 Chiaramente, l'accuratezza è alta se abbiamo molte previsioni corrette.

Allo stesso modo, si può definire un'altra metrica di valutazione, ovvero l'**error rate** → l'error rate è il numero di predizioni sbagliate rispetto al numero di predizioni totali. In questo caso, quindi, andiamo a sommare gli elementi sulla diagonale opposta della matrice di confusione (ovvero False Negative e False Positive) diviso il totale delle predizioni eseguire sul Test set. Formalmente la formula è:

$$err = (Fpn + Fnp)/(Fpp + Fpn + Fnp + Fnn)$$

💡 Chiaramente, l'error rate deve essere un valore basso e di conseguenza dobbiamo avere dei bassi valori sugli elementi della diagonale opposta.

Tipi di classificatori

Abbiamo già detto, che una prima differenza tra i classificatori è tra quelli binari e quelli multi-classe. L'altra differenza che vi è tra i classificatori è che i classificatori possono essere:

- **Deterministici** → sono ad esempio gli alberi decisionali, ovverosia i classificatori deterministici associano ciascuna istanza ad una e una sola etichetta;
- **Probabilistici** → non danno delle "certezze" forti come i classificatori deterministici, in quanto i classificatori probabilistici associano una probabilità di appartenenza a ciascuna delle classi (per esempio mi dicono: un certo individuo, con probabilità 80%, appartiene alla classe degli studenti universitari e con probabilità 20% alla classe dei lavoratori).

💡 **Dato un calcolatore probabilistico, siamo sempre in grado di trasformarlo in un calcolatore deterministico**, andando ad assegnarli un'unica classe con la probabilità più alta.

Altre differenze tra i classificatori sono:

- Classificatori **lineari** e **non-lineari** → un classificatore lineare utilizza un'iperplancia di separazione lineare, mentre un classificatore non-lineare consente di costruire superfici decisionali più complesse e non lineari;
- Classificatori **globali** e **locali** → un classificatore globale adatta un singolo modello all'intero Data set, mentre un classificatore locale suddivide lo spazio di input in regioni più piccole e adatta un modello distinto alle istanze di addestramento di ciascuna regione;
- Classificatori **generativi** e **discriminativi** → i classificatori generativi apprendono un modello generativo di ogni classe nel processo di previsione delle etichette di classe, mentre i classificatori discriminativi predicono direttamente le etichette di classe senza descrivere esplicitamente la distribuzione di ciascuna di esse.

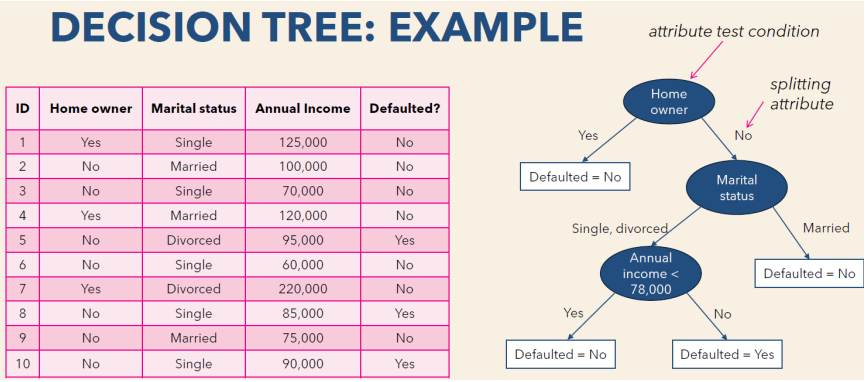
Partiamo ad analizzare gli **alberi decisionali**, che sono il classificatore più semplice e sostanzialmente, consistono nel costruire un albero in cui ad ogni livello viene fatta una domanda (che riguarda gli attributi delle istanze) e in base alla risposta che si ottiene, si dividono le istanze nei figli del livello. Quindi, all'interno di un albero decisionale, riconosciamo:

- un **nodo radice**;
- dei **nodi interni** → i quali contengono delle condizioni sugli attributi delle istanze;
- i **nodi foglia** (detti anche nodi terminali) → a ciascuno dei quali è associato una specifica etichetta.

Quindi, se prendendo un'istanza e rispondendo alle domande sugli attributi si segue il percorso sull'albero, ad un certo punto si arriva ad un nodo foglia, che ci dà l'etichetta da assegnare all'istanza che abbiamo preso → capiamo, allora, che una volta che abbiamo costruito un albero decisionale, è molto semplice fare la classificazione delle istanze, in quanto basta:

- seguire i path all'interno dell'albero;
- rispondere alle domande sugli attributi;
- seguire le risposte che vengono date.

A questo punto, proviamo a vedere un primo esempio di albero decisionale: abbiamo una tabella di 10 istanze e l'etichetta che vogliamo determinare è "Defaulted" e in base ai 3 attributi Home owner, Marital status e Annual Income, vogliamo vedere se un soggetto sarà inadempiente (ovvero defaulted) o meno:



ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes

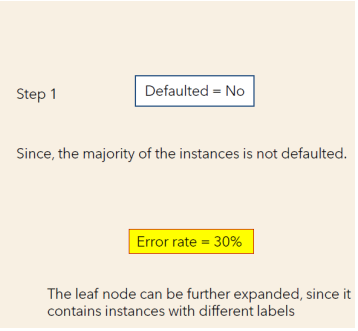
💡 Possiamo notare, che tipicamente l'albero decisionale è binario (può anche essere non-binario) e i vari nodi possono avere un'altezza diversa, ovverosia l'albero non è necessariamente bilanciato.

La costruzione degli alberi decisionali presenta diverse problematiche, come:

- Sicuramente un primo problema, è che data una certa istanza iniziale, ci sono molti modi di costruire gli alberi decisionali e alcuni alberi sono meglio di altri e di conseguenza, non è scontato che partendo da un'istanza casuale costruiamo un albero, che ci porta ad una buona classificazione → trovare, quindi, l'albero decisionale migliore non è un'operazione computazionalmente banale, in quanto potenzialmente dobbiamo esplorare tutto il possibile spazio delle soluzioni.

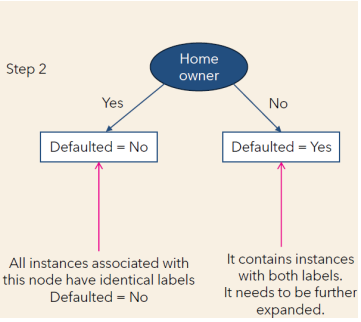
Per evitare di dover esplorare tutto il possibile spazio delle soluzioni, sono stati implementati degli algoritmi, che non necessariamente portano all'ottimo globale (ovvero non necessariamente portano alla costruzione del miglior albero decisionale), ma sono un buon compromesso. Questi algoritmi, sono tipicamente degli algoritmi greedy (ovvero ad ogni passo cercano di fare una scelta localmente ottima, ma non è detto che ci portano all'ottimo globale) ed in particolare, esistono diversi algoritmi greedy per costruire degli alberi decisionali, ma noi vedremo solamente l'**algoritmo Hunt** → attraverso questo algoritmo, l'albero decisionale viene costruito in maniera ricorsiva e quindi, si parte da un albero che contiene un solo nodo, a cui si associano tutte le istanze. A questo punto, se al nodo sono associate istanze appartenenti a classi diverse, questo (ovvero il nodo) viene espanso ulteriormente aggiungendo una nuova condizione su uno dei suoi attributi (quindi andando a definire un criterio di split) e a questo punto viene creato il nuovo nodo (attraverso il criterio di split) e si associano le istanze ai rispettivi figli. Nuovamente, se vi è un figlio in cui tutte le istanze appartengono alla stessa etichetta, allora possiamo fermarci, altrimenti se vi è un nodo in cui le istanze appartengono ad etichette differenzi, procediamo nuovamente ad espandere il nodo in maniera ricorsiva. Quando tutte le istanze associate allo stesso nodo foglia appartengono alla stessa etichetta, allora l'algoritmo può terminare. Vediamo un esempio di costruzione dell'albero tramite l'algoritmo di Hunt:

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes



Vediamo che partiamo con un unico nodo, a cui sono associate tutte le istanze e al nodo mettiamo l'etichetta più frequente nel nostro insieme di istanze (che nel nostro esempio è l'etichetta Defaulted = NO). Siccome al nodo sono associate istanze che hanno sia etichetta "Defaulted = NO" sia l'etichetta "Defaulted = YES", dobbiamo richiamarci ricorsivamente e quindi eseguire un passo di espansione. Il secondo passo dell'algoritmo, allora, è il seguente:

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes



Scegliamo un attributo su cui fare splitting e in questo caso scegliamo l'attributo "Home owner". Tale attributo ha due possibili outcome, ovvero "YES" e "NO" e quindi costruiamo un nodo "Home owner" a cui associo due etichette e a fronte del valore YES dell'attributo, dobbiamo capire quali etichette mettere e a fronte del valore NO quali etichette mettere. Naturalmente sia nel caso YES sia nel caso NO, dobbiamo mettere l'etichetta più probabile, che nel nostro caso è:

- per il valore YES, utilizziamo l'etichetta "Defaulted = NO";
- per il valore NO, utilizziamo l'etichetta "Defaulted = YES".

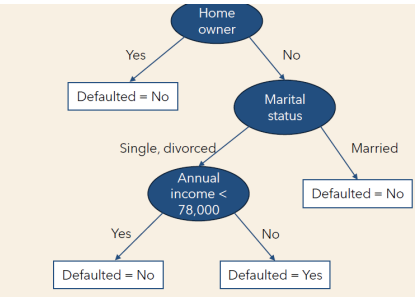
Dato che per questo nodo "Home owner" abbiamo istanze sia con etichetta "Defaulted = NO" sia con etichetta "Defaulted = YES", andremo ad applicare ricorsivamente il passo di espansione. Per questo passo di espansione utilizziamo l'attributo "Marital status" e quindi costruiamo il nodo "Marital status", a cui associo due etichette e a fronte dei valori "Single e divorced" dell'attributo, dobbiamo capire quali etichette mettere e a fronte del valore Married dell'attributo, dobbiamo capire quali etichette mettere. Naturalmente in

entrambi i casi, dobbiamo mettere l'etichetta più probabile, che nel nostro caso è:

- per i valori Single e Divorced, utilizziamo l'etichetta "Defaulted = YES";
- per il valore Married, utilizziamo l'etichetta "Defaulted = NO".

Per il ramo Married ho terminato, in quanto tutte le istanze associate al nodo appartengono alla stessa etichetta, mentre per il ramo "Single, divorced" dobbiamo applicare un ultimo passo di espansione, che è il seguente:

ID	Home owner	Marital status	Annual Income	Defaulted?
1	Yes	Single	125,000	No
2	No	Married	100,000	No
3	No	Single	70,000	No
4	Yes	Married	120,000	No
5	No	Divorced	95,000	Yes
6	No	Single	60,000	No
7	Yes	Divorced	220,000	No
8	No	Single	85,000	Yes
9	No	Married	75,000	No
10	No	Single	90,000	Yes

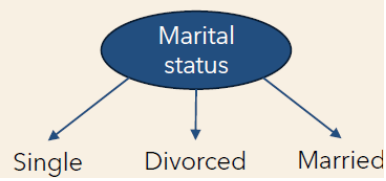


Sorge a questo punto spontanea la domanda: "**Quali sono i limiti dell'algoritmo di Hunt?**" I limiti di questo algoritmo sono:

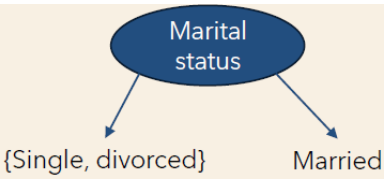
1. La **difficoltà nel trattare gli attributi che non hanno valore binario** (e quindi dobbiamo capire come trattare gli attributi che hanno più di due possibili valori);
2. La **difficoltà nel trattare gli attributi che hanno un valore continuo** (e quindi dobbiamo capire dove "spezzare" il valore dell'attributo);
3. La difficoltà più grande, consiste nel fatto che **potremmo avere che alcuni nodi figli (che abbiamo costruito a fronte della condizione sull'attributo) in realtà sono vuoti**, ovverosia non hanno alcuna istanza associata. In questo caso, quindi dobbiamo capire quale etichetta associare al nodo figlio. Inoltre, se tutte le istanze associate ad un nodo hanno tutti gli attributi uguali, ma appartengono a classi differenti, non riusciamo a trovare un'uniformità nei nodi foglia → in questo caso, si associa a ciascun nodo foglia il valore più frequente e quindi si ammette comunque un errore di classificazione;
4. La **difficoltà nel determinare qual è il criterio migliore per eseguire lo split**, ovvero la difficoltà nello scegliere tra tutti i possibili attributi, quale attributi considerare ad ogni passo ricorsivo → l'idea per risolvere questa difficoltà, è di applicare un algoritmo greedy, in cui ad ogni passo ricorsivo si valuta la

situazione attuale (ovvero la situazione in cui ci troviamo) e cerchiamo di prendere l'attributo migliore in questo momento. Una volta scelto l'attributo, possiamo ricadere nel caso semplice o nel caso complesso:

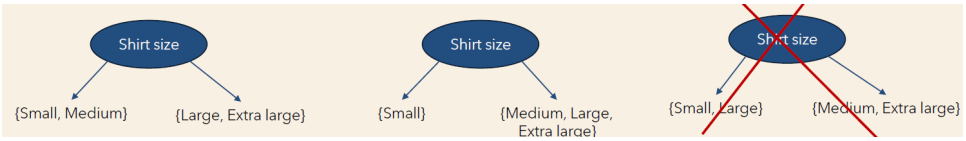
a. caso semplice → ricadiamo nel caso semplice quando l'attributo è binario, ovvero abbiamo solamente due possibili valori dell'attributo (come per esempio: l'attributo Home owner ha solamente i valori Yes e NO);
b. caso complesso → ricadiamo nel caso complesso, quando l'attributo è nominale, ovvero l'attributo ha molti valori possibili. In questo caso, possiamo decidere di adottare due soluzioni:
    i. **Multi-way split** → avere un nodo figlio per ciascun possibili valore dell'attributo (capiamo immediatamente, che questa soluzione è valida solamente nel caso in cui i valori possibili dell'attributo sono pochi);



    ii. **Binary split** → soluzione maggiormente utilizzata, che consiste nell'avere comunque uno split binario, andando ad aggregare i valori in due gruppi.

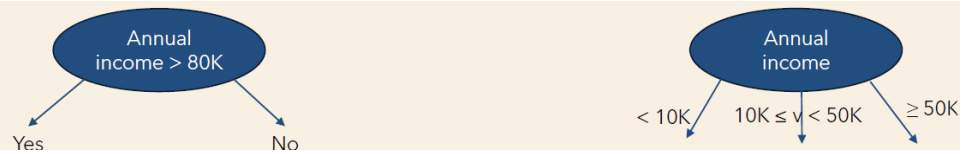


Alle volte i nostri attributi non hanno solamente un valore nominale, bensì hanno un **attributo ordinale**, ovvero: i possibili valori associati all'attributo non sono delle etichette tra loro intercambiabili, ma h**anno un significato anche dal punto di vista dell'ordinamento e di conseguenza, dobbiamo rispettare tale ordine nel definire i gruppi**. Vediamo il seguente esempio per capire meglio:



Il problema più grande, però, lo abbiamo quando gli **attributi sono continui** → in questo caso, la condizione sull'attributo non è più una condizione basata sull'uguaglianza, bensì è una condizione che va espressa con un test di disuguaglianza oppure con una range query → dobbiamo cioè definire una strategia di discretizzazione, la quale (anche in questo caso) può essere una:

- discretizzazione binaria;
- discretizzazione multi-way.



A questo punto, capiamo come selezionare la miglior condizione sugli attributi passo per passo. Come dicevamo prima, ad ogni passo ricorsivo dobbiamo scegliere l'attributo su cui fare lo split ed in questo senso, vi sono diverse metriche per esprimere la bontà della condizione sull'attributo → l'idea generale, comunque, consiste nello scegliere l'attributo che ci consente di individuare dei partizionamenti più puri possibili, ovvero individuare per ogni attributo uno split, che produce una suddivisione delle istanze pura, ovverosia che appartengono alla stessa etichetta → questo ha come effetto immediato, che riusciamo a costruire degli alberi più piccoli e di conseguenza più facilmente generalizzabili e comprensibili (nel senso che, se costruiamo un albero grande, è molto probabile che tale albero vada bene solamente per determinate istanze. Invece, un albero piccolo è più probabile che vada bene per diverse istanze).

Sorge spontanea la domanda:"**Come facciamo a determinare l'impurità di un nodo?**" Intuitivamente, più è uniforme la classe delle istanze che appartengono ad un certo nodo, più il nodo è puro. In realtà, sono state definite tre misure di impurità, che sono:

- l'**entropia**;
- l'**indice Gini**;
- l'**errore di classificazione**.

💡 Queste tre misure ci possono dare dei risultati diversi, ma tra di loro sono comunque **consistenti**, ovvero: se confronto l'entropia di due nodi e l'entropia del 1°nodo è minore dell'entropia del 2°nodo, allora anche l'indice Gini del 1°nodo sarà più piccolo dell'indice Gini del 2°nodo.

In particolare le formule di queste tre misure sono le seguenti:

- Impurity of a node t (less is better):
  - Entropy = $-\sum_{i=0}^{c-1} p_i(t) log_2\, p_i(t)$
  - Gini index = $1 - \sum_{i=0}^{c-1} p_i(t)^2$
  - Classification error = $1 - max_i(p_i(t))$

$p_i(t)$ is the relative frequency of training instances that belong to class *i* at node *t*

c is the total number of classes

0 log_2 0 = 0

💡 **Naturalmente minori sono i valori di tali misure, migliore sarà la classificazione sul nodo (che è il nostro obiettivo). Quindi quello che vorremmo è di ottenere zero in tutte e tre le misure (rappresenta il caso migliore) e vorremmo evitare di ottenere 1 nelle misure (rappresenta il caso peggiore).**

Vediamo alcuni esempi per capire come vengono calcolate le tre misure di impurità:

Node N1	Count
Class = 0	0
Class = 1	6

Entropy = $-$ [(0/6)$\log_2$(0/6) + (6/6)$\log_2$(6/6)] = 0
Gini index = 1 - [(0/6)^2 + (6/6)^2] = 0
Error = 1 - max[0/6, 6/6] = 0

Node N2	Count
Class = 0	1
Class = 1	5

Entropy = $-$ [(1/6)$\log_2$(1/6) + (5/6)$\log_2$(5/6)] = 0.650
Gini index = 1 - [(1/6)^2 + (5/6)^2] = 0.278
Error = 1 - max[1/6, 5/6] = 0.167

Node N3	Count
Class = 0	3
Class = 1	3

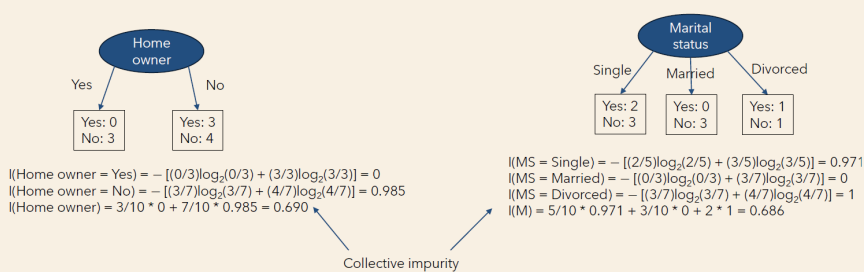
Entropy = $-$ [(3/6)$\log_2$(3/6) + (3/6)$\log_2$(3/6)] = 1
Gini index = 1 - [(3/6)^2 + (3/6)^2] = 0.5
Error = 1 - max[3/6, 3/6] = 0.5

Queste tre misure riguardano l'impurità del singolo nodo, però **quando andiamo a costruire un albero, ci interessa sapere anche qual è l'impurità collettiva**. Immaginiamoci, allora, di avere un nodo N e di avere un ulteriore livello, che mi

va a divedere le istanze del nodo in K figli e indichiamo con N(Vj) quali sono le istanze che appartengono a ciascuno dei suoi figli, possiamo definire l'impurità dei figli come:

$$I(\text{children}) = \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$

(l'impurità dei figli, quindi, è la sommatoria su tutti i figli del numero di istanze sul figlio diviso il numero totale di istanze sul nodo padre moltiplicata per l'impurità del nodo figlio). Vediamo un esempio:

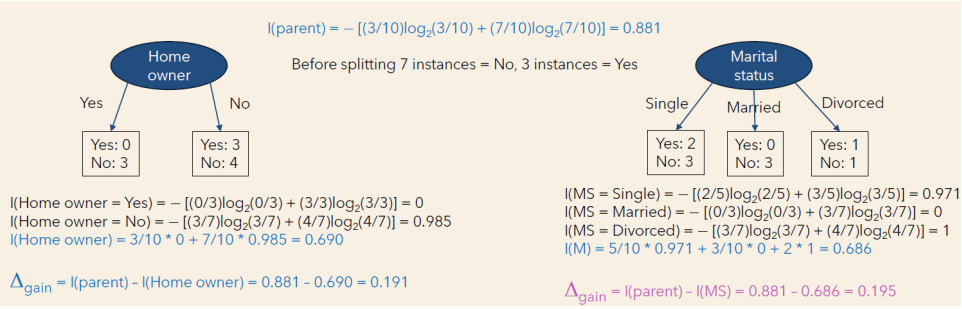


💡 **Notiamo che prima di calcolare l'impurità collettiva, andiamo a calcolare l'impurità dei singoli nodi figli attraverso la misura di entropia (ma lo possiamo fare anche attraverso le altre due misure). L'impurità collettiva, invece, viene calcolata come la media pesata delle impurità dei figli.**

L'impurità collettiva, calcolata sui nodi, ci farà decidere (ad ogni passo) qual è l'attributo su cui eseguire lo split e ovviamente sceglieremo l'attributo che ci darà l'impurità collettiva più bassa. Per riuscire a fare questo, dobbiamo determinare qual è il vantaggio che otteniamo nell'andare ad aggiungere un'ulteriore condizione di test dato il nodo che abbiamo. Quindi, nel momento in cui andiamo ad eseguire un passo ricorsivo (e quindi aggiungere un nodo figlio al nodo corrente) dobbiamo capire qual è il vantaggio nel fare ciò. Questo vantaggio, si calcola come la differenza tra l'impurità del nodo padre e l'impurità dei nodi figlio.

💡 Il vantaggio (chiamato anche Gain) è sempre positivo, in quanto l'impurità del padre è sempre maggiore o uguale all'impurità dei figli e maggiore è la differenza tra le due impurità, maggiore è il guadagno di informazioni che stiamo ottenendo e di conseguenza migliore è la scelta.

Vediamo un esempio:

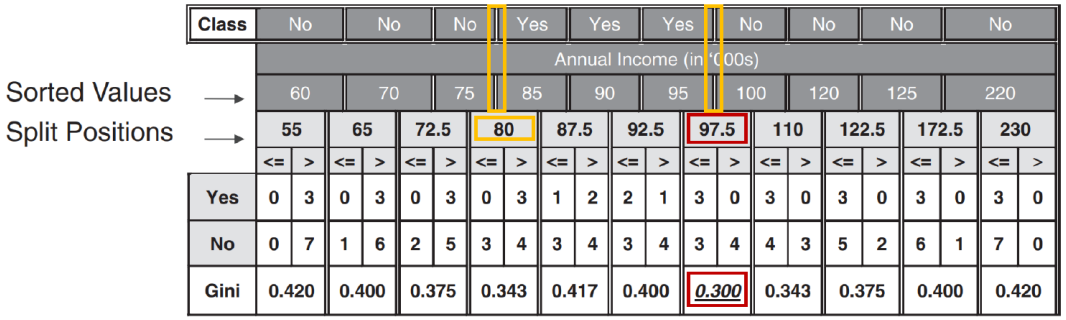


Riassumendo, quindi, l'algoritmo di costruzione di un albero decisionale è un algoritmo che si comporta in maniera ricorsiva, andando a costruire i vari livelli dell'albero, partendo da un nodo radice (il quale inizialmente contiene tutte le istanze) e via via si crea un nuovo nodo, andando a trovare il miglior criterio di split per ogni livello. A questo punto, si dividono le istanze del nodo padre tra i possibili nodi figli, ovverosia rispetto all'attributo di split si costruiscono le possibili alternative (che abbiamo visto essere 2 o anche di più) e si suddividono le istanze sui nodi figli. A questo punto, l'algoritmo continua fino a quando non si è raggiunta la **condizione di stop**, la quale può essere:

- tutte le istanze, che appartengono ad un certo nodo, abbiano tutte la stessa etichetta;
- il livello di impurità sia inferiore ad una certa soglia.

Parliamo, infine, degli **attributi quantitativi** → tornando all'esempio inziale, come abbiamo stabilito l'attributo "Income < 78000"? Questi 78000 rappresentano il nostro T, il quale può essere scelto pari a qualsiasi valore compreso tra il minimo e il massimo. È sufficiente considerare solo i valori osservati nell'insieme di addestramento come posizioni di suddivisione candidate e possiamo calcolare l'indice di Gini in ogni posizione candidata e scegliere il T che produce il valore più basso. Una volta che abbiamo ordinato le istanze di training in base al valore dell'attributo di divisione, abbiamo che le

posizioni di divisione candidate sono date dai punti medi tra ogni due valori adiacenti ordinati. Vediamo un esempio:



Class	No		No		No		Yes		Yes		Yes		No		No		No		No	
Annual Income (in '000s)																				
Sorted Values →	60		70		75		85		90		95		100		120		125		220	
Split Positions →	55		65		72.5		80		87.5		92.5		97.5		110		122.5		172.5		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400		0.420	

A questo punto, analizziamo in maniera più approfondita quali sono i vantaggi e gli svantaggi degli alberi decisionali:

- Uno dei vantaggi fondamentali, che riguarda l'applicabilità di questa tecnica, è il fatto che si tratta di un **approccio non-parametrico**, ovverosia lo possiamo utilizzare senza conoscere alcuna informazione sulle distribuzione di probabilità delle classi o degli attributi. Ovviamente, questa caratteristica, rende gli alberi decisionali applicabili ad un'ampia varietà di problemi;
- Un altro vantaggio, è che **lo possiamo applicare sia ad attributi che hanno valori categorici** (dove ricordiamo, che i valori categorici possono essere sia binari sia multi-valore) **sia ad attributi che hanno valori contigui**;
- Un altro vantaggio, è che dopo aver costruito l'albero decisionale, quest'ultimo è **"auto-esplicativo", ovverosia riusciamo ad interpretarlo in maniera semplice** → questa è una caratteristica, che lo differenza dalle metodologie black-box del Machine Learning, in cui la macchina ottiene in input i dati e produce automaticamente le etichette, senza che noi utenti sappiamo il motivo di tali classificazioni;
- Dal punto di vista computazionale, costruire l'albero di decisione non è molto efficiente, perchè dovremmo costruire tutti i possibili alberi e successivamente valutare il migliore. Vi sono degli **algoritmi (anche greedy), però, che ci consentono di costruire l'albero di decisione in maniera più efficiente, anche se non è detto che sia l'albero ottimo** (ma comunque si avvicina all'ottimo);
- **Una volta compiuto lo sforzo di costruire l'albero decisionale, che solitamente è un'operazione che viene fatta una sola volta, poi**

l'operazione di classificazione è estremamente veloce, in quanto dobbiamo solamente eseguire i test sui vari nodi (dell'albero) e vedere qual è il percorso che dobbiamo seguire sull'albero.

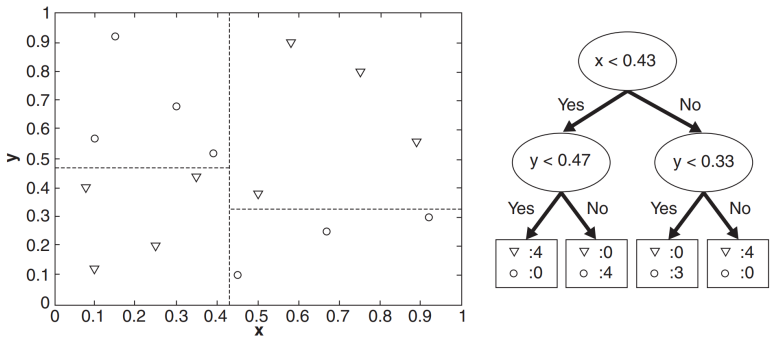
Gli svantaggi, invece, sono:

- Uno dei problemi, che possiamo incontrare nel costruire un albero decisionale e successivamente anche nel classificare un albero decisionale, è di capire cosa succede nel momento in cui abbiamo dei valori mancanti su uno degli attributi. Vi sono diverse implementazioni per risolvere questo problema, come per esempio:
  - implementazione C4.5 → utilizza uno **split probabilistico**, ovverosia divide le istanze rispetto alla probabilità, che l'attributo mancante abbia un certo valore;
  - implementazione CART → considera le relazioni tra gli attributi e identifica un **attributo surrogato**, ovverosia nel momento in cui abbiamo un valore mancante su un determinato attributo, possiamo utilizzare il valore assunto da un altro attributo.
- Un altro problema, consiste nel gestire gli attributi che hanno interazioni fra di loro, ovvero: se ci sono degli attributi che interagiscono tra di loro, ovvero che sono in grado di portarci ad una classificazione solamente se vengono considerati insieme e quindi singolarmente non ci danno alcuna informazione, allora gli alberi decisionali non funzionano molto bene, perchè essi (ovvero gli alberi) considerano un attributo alla volta.

> 💡 Gli attributi irrilevanti e gli attributi ridondanti (ovvero gli attributi che ci danno la stessa informazione), invece, vengono gestiti in maniera molto semplice dagli alberi decisionali.
> Notiamo, inoltre, che
> **la scelta di una misura di impurità piuttosto che un'altra, non ha alcun effetto sull'albero che costruiamo, perchè sono misure consistenti tra di loro, ovvero mi portano alla stessa scelta in un certo punto di costruzione dell'albero.**

- Un altro grave problema degli alberi decisionali, è che essi ci portano a degli **split rettilinei**, ovvero il fatto che andiamo ogni volta a considerare un singolo attributo di split, può essere visto come un partizionamento dello

spazio delle istanze utilizzando un piano, il quale è parallelo ad uno degli assi. Supponiamo, per esempio, di avere due attributi:

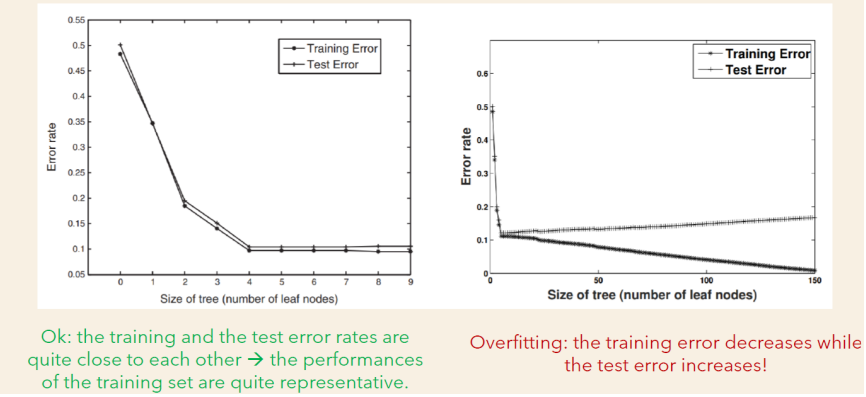


Dividere l'istanza rispetto ai due attributi (prima uno e poi l'altro) corrisponde a costruire dei piani paralleli agli assi, i quali dividono le nostre istanze. Vediamo, nell'esempio, che come prima condizione abbiamo 'x < 0.43' e questo corrisponde a costruire una linea di separazione verticale, che divide le istanze che hanno un valore minore di 0.43, dalle istanze che hanno un valore maggiore o uguale di 0.43 (stesso discorso vale anche per le altre due condizioni).

> 💡 Notiamo, allora, che abbiamo solamente delle linee parallele agli assi, ognuno dei quali corrisponde ad un attributo, e tali linee parallele prendono anche il nome di **Decision Boundary**.

Fino a qua tutto bene, ma il vero problema è che gli alberi decisionali non ci consentono di costruire delle linee oblique, quindi non ci permettono di costruire dei Decision Boundary obliqui. Quindi, un albero decisionale può costruire, al massimo, un'approssimazione della diagonale, come un insieme di linee parallele ad uno degli assi → ovviamente, tali Decision Boundary (che appunto sono un'approssimazione della diagonali) sono molto complessi e ci portano al **problema dell'overfitting**. Sorge a questo punto spontanea la domanda: "**Cos'è il problema dell'overfitting?**" Il problema dell'overfitting si verifica quando il modello calza perfettamente per le istanze che abbiamo utilizzato per il training, ma non è in grado di generalizzare bene per le istanze che non ha mai visto (ovvero per istanze che andremo a classificare una volta che il modello è stato costruito). Al contrario, possiamo incorrere anche nel **problema dell'underfitting**, cioè quando abbiamo un modello troppo semplicistico, che è incapace di rappresentare realmente le relazioni che vi sono tra gli attributi e le classi.

“**Come facciamo a verificare se siamo andati effettivamente in overfitting o meno?**” Per verificare se siamo andati in overfitting o meno, dobbiamo andare a stampare l’errore commesso sulle istanze di training e l’errore commesso sulle istanze di test rispetto all’aumentare della grandezza dell’albero decisionale e quindi rispetto all’aumentare del numero di nodi foglia. Se all’aumentare della grandezza dell’albero (e quindi all’aumentare del numero di nodi foglia) l’andamento dei due errori è molto simile (in particolare l’errore di test è leggermente più alto dell’errore di training), allora vuol dire che il nostro modello è in grado di generalizzare molto bene. Al contrario, se l’andamento dei due errori è molto diverso all’umentare della grandezza dell’albero, con l’errore di training continua a diminuire fino ad arrivare quasi a zero, mentre l’errore di test aumenta, allora siamo andati in overfitting. Rappresentiamo graficamente questi concetti per capire meglio:



Ok: the training and the test error rates are quite close to each other → the performances of the training set are quite representative.

Overfitting: the training error decreases while the test error increases!

Le cause dell’overfitting possono essere le seguenti:

- una delle cause tipiche per il cui il modello va in overfitting, è la situazione in cui siamo andati a “cucire” troppo perfettamente il nostro modello sui dati di training e quindi abbiamo un modello che cattura tutte le caratteristiche dei dati di training, ma non è in grado di generalizzare agli altri dati → per risolvere questo problema, la best practice è:
    - da un parte limitare la complessità del modello → per fare ciò, si va ad imporre la condizione di termine di costruzione del modello non più come il fatto di avere delle classi completamente pure, bensì si impone come condizione di termine una determinata altezza dell’albero. In questo modo, si ammette di avere comunque un errore di classificazione per le istanze di training, ma si evita di andare in overfitting;

    - dall'altra parte si cerca di fornire al modello un insieme di istanze di training più ampio (in quanto una causa di overfitting è di fornire al modello un insieme di istanze di training troppo limitato), in modo tale da fornire al modello tutte le possibile casistiche che dobbiamo considerare.

- un'altra causa dell'overifitting è la presenza del **rumore** → dobbiamo trovare un compromesso tra l'errore che commettiamo nel training set (ovvero nell'insieme delle istanze di training) e la capacità di generalizzazione del nostro modello. Dobbiamo, quindi, trovare un modo per misurare la capacità di generalizzazione e per riuscire a fare ciò, invece di avere una suddivisione del nostro insieme di istanze in training set e test set, vi è una ulteriore suddivisione del training set in quello che viene chiamato **validation set**. Quindi, andiamo a suddividere il training set iniziale in due sotto-insiemi:
    - la parte più grande (circa i 2/3) viene utilizzata per costruire il training set vero e proprio, ovvero quello che utilizziamo per costruire l'albero decisionale;
    - la parte rimanente (ovvero 1/3) viene utilizzata come validation set.

    A questo punto, quando andiamo a valutare la bontà del modello, andiamo a considerare qual è l'errore che viene commesso nella classificazione del validation set.

Riassumiamo quello che abbiamo detto fino a questo momento: Abbiamo costruito il modello e abbiamo notato che tale modello va in overfitting. Allora, possiamo applicare delle tecniche per cercare di migliorare il modello che abbiamo costruito, andandone sostanzialmente a ridurre la complessità (dato che precedentemente abbiamo detto, che un modello complesso sarà più facilmente soggetto ad overfitting). In particolare, vi sono due famiglie di tecniche per riuscire a fare ciò:

1. **Tecniche di Pre-pruning** → esse vengono adottate prima di costruire il modello e consistono sostanzialmente, nel fermare l'algoritmo prima di aver costruito un albero decisionale completo (quindi prima di aver utilizzato tutti gli attributi). Possiamo, ad esempio, considerare sufficiente che la maggior parte delle istanze appartengano ad una certa classe. Il vantaggio di questa tecnica, è che ci permette di non dover costruire un albero completo e di conseguenza, è computazionalmente meno costoso costruire l'albero

decisionale. Lo svantaggio, invece, è che se ci limitiamo nell'altezza dell'albero, potremmo non raggiungere un sotto-albero ottimo;

2. **Tecniche di Post-pruning** → esse ci permettono di costruire l'albero decisionale completo e una volta costruito lo semplificano. In particolare, vanno a semplificare l'albero decisionale nel seguente modo:
   a. andando a rimpiazzare o addirittura eliminare una parte di un sotto-albero, se si vede che questa operazione migliora la generalizzazione dell'albero stesso.

   In questo caso, il vantaggio è che otteniamo dei risultai migliori rispetto alla tecnica di Pre-pruning, in quanto possiamo ottenere un albero ottimale. Lo svantaggio, invece, è che dobbiamo innanzitutto costruire tutto l'albero e successivamente dobbiamo semplificarlo.

Classificatori basati su regole

Una tecnica alternativa agli alberi decisionali, ma che è comunque molto simile a quest'ultimi, anche in termini di capacità espressiva, è la tecnica dei **classificatori basati su regole**. Un classificatore basato su regole utilizza una collezione di **regole del tipo IF THEN** e quindi abbiamo un antecedente, che è rappresentato da una serie di condizioni e se la nostra istanza soddisfa un certo insieme di condizioni, allora tale istanza appartiene ad una determinata classe. Queste condizioni sono semplicemente delle congiunzioni di condizioni semplici sugli attributi (quindi sulle coppie attributo-valore) e questo ci fa capire, che **un albero decisionale lo possiamo tradurre in una regola, perchè se seguiamo il path che parte dalla radice e arriva fino alla foglia e mettiamo in AND le condizioni che abbiamo definito sui nodi (ovvero se lego le condizioni con degli AND) otteniamo una regola**.

> 💡 Possiamo, quindi, rappresentare un albero come un classificatore basato su regole.

Attraverso tali classificatori, la **qualità** di una regola si misura in base a due parametri:

1. **Coverage (Copertura)** → dato un insieme di istanze 'D' e una regola di classificazione 'r' del tipo "A → y", allora la Coverage è il rapporto tra il

numero di istanze di D e il numero di istanze che soddisfano le pre-condizioni.

2. **Accuratezza** → dato un insieme di istanze 'D' e una regola di classificazione 'r' del tipo "A → y", allora l'Accuratezza è la frazione di istanze che sono attivate dalla regola e che hanno un etichetta uguale a 'y', ovvero: tra tutte le istanze A, che sono attivare dalla regola 'r', prende le istanze che vengono classificate correttamente.

Notiamo, che quando andiamo a costruire un classificatore basato su regole, dobbiamo costruire delle regole che soddisfino due caratteristiche, ovvero:

- **Set di regole mutualmente esclusive** → le regole di un set di regole devono essere mutualmente esclusive, se nessuna regola del set viene attivata dalla stessa istanza. Quindi, ogni istanza è coperta al massimo da una regola del set di regole e di conseguenza, se una regola è vera non possiamo avere un'altra regola vera nello stesso momento;
- **Insieme di regole esaustivo** → un insieme di regole ha una copertura esaustiva, se esiste una regola per ogni combinazione di valore degli attributi. Quindi, ogni istanza è coperta da almeno una regola del set di regole.

Non è detto, che possiamo costruire un set di regole esaustivo, quindi per risolvere questo problema (e di conseguenza semplificare anche l'insieme di regole) c'è la possibilità di definire una **regola di default** → tale regola di default è rappresentata nel seguente modo:

$$r_d : () \rightarrow y_d$$

notiamo che abbiamo l'antecedente vuoto e il conseguente è una classe ed in particolare, la regola di default ci indica che se non abbiamo alcuna regola che ci va a soddisfare gli antecedenti delle regole che abbiamo definito, allora la classe che andiamo ad associare di default è la classe 'y'.

Inoltre, se abbiamo definito delle classi non esclusive, allora possiamo definire un **ordine sulle regole**, ovvero: se esistono più regole attivabili da una certa istanza, queste (ovvero le regole) hanno un ordine di attivazione e quindi la prima dell'ordine, sarà la prima che verrà attivata.

Sorge a questo punto spontanea la domanda:"**Come vengono costruite le regole?**" Le regole si costruiscono partendo:

- dall'insieme delle istanze E;
- dall'insieme degli attributi A.

e **ordiniamo le classi**, che dobbiamo andare a classificare, **rispetto all'ordine di rilevanza**, ovverosia rispetto a quante istanze ci sono per ogni classe e di conseguenza, la classe più rilevante (ovverosia la classe che ricopre il maggior numero di istanze) viene messa in ultima posizione. A questo punto, definiamo l'insieme di regole, che inizialmente è vuoto, e per ogni classe (ordinata rispetto alla rilevanza) costruiamo una nuova regola in modo ricorsivo, attraverso una funzione (che nell'algoritmo sotto prende il nome di '**Learn-One-Rule**'). Tale idea, la possiamo descrivere con il seguente algoritmo:

Algorithm 5.1 Sequential covering algorithm.

1: Let E be the training records and A be the set of attribute-value pairs, $\{(A_j, v_j)\}$.
2: Let Y_o be an ordered set of classes $\{y_1, y_2, \ldots, y_k\}$.
3: Let $R = \{\ \}$ be the initial rule list.
4: **for** each class $y \in Y_o - \{y_k\}$ **do**
5: **while** stopping condition is not met **do**
6: $r \leftarrow$ Learn-One-Rule (E, A, y).
7: Remove training records from E that are covered by r.
8: Add r to the bottom of the rule list: $R \longrightarrow R \vee r$.
9: **end while**
10: **end for**
11: Insert the default rule, $\{\} \longrightarrow y_k$, to the bottom of the rule list R.

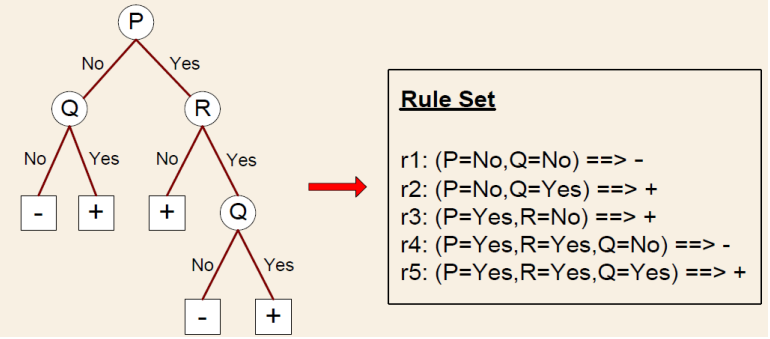
Possiamo immediatamente capire, che anche in questo caso, costruire l'albero ottimale è un'operazione computazionalmente costosa e la funzione '**Learn-One-Rule**' (che possiamo vedere nell'algortimo sopra) cerca di rendere efficiente la ricerca della prossima regola ottimale, tramite la strategia greedy. Per ricercare la prossima regola ottimale, tale funzione utilizza una misura di gain d'informazione, che prende il nome di **FOIL** → tale misura (ovvero il FOIL) ci consente di scegliere la migliore congiunzione da aggiungere all'antecedente delle regole ed in particolare, il FOIL si basa su due informazioni:

1. l'**accuratezza**;
2. il **supporto della regola**.

Anche con questa tecnica, però, abbiamo il problema dell'overfitting, ovverosia costruiamo delle regole che nella peggiore dell'ipotesi sono una condizione su tutti gli attributi della riga e ci dicono, che se abbiamo tutti gli attributi della riga, allora abbiamo tale classe. Ad un certo punto, quindi, dovremmo svolgere il **pruning**, ovvero semplificare le regole in modo tale da ottenere un miglior livello di generalizzazione. Per svolgere il pruning, si utilizza il validation set e quindi, si fa il training sul sotto-insieme delle istanze e si va a validare, su delle istanze

nuove (mai viste), per vedere se le regole riescono a catturare il comportamento generale. In particolare, il pruning va a prendere l'insieme delle congiunzioni che abbiamo nell'antecedente della regola e prova a rimuovere una condizione e la valuta, al fine di vedere qual è la regola più semplice, ma che comunque è in grado di fornire un buona accuratezza di classificazione.

Possiamo costruire le regole di classificazione anche in maniera indiretta, ovvero partendo dall'albero decisionale, in quanto ogni percorso che parte dalla radice ed arriva ad una foglia costituisce una regola, in cui tutte le condizioni che troviamo nel path rappresentano l'antecedente e l'etichetta che troviamo nella foglia rappresenta la classe. Vediamo la seguente immagine, per capire meglio questo concetto:



Vediamo, a questo punto, le caratteristiche delle regole di classificazione:

- **L'espressività delle regole di classificazione è del tutto simile all'espressività degli alberi decisionali**, infatti posso codificare uno nell'altro (e viceversa), nel senso che posso codificare le regole di classificazione con gli alberi decisionali e viceversa. In realtà, se vogliamo essere precisi, se non consideriamo che le regole siano strettamente disgiunte (e quindi consideriamo anche la possibilità di avere delle regole sovrapposte), allora in questo caso il classificatore basato sulle regole si può considerare anche più espressivo degli alberi decisionali, in quanto ci permette di costruire dei modelli più complessi;
- Anche per quanto riguarda gli attributi ridondante, le regole di classificazione sono simili agli alberi decisionali, ovverosia **se abbiamo degli attributi tra di loro ridondanti, semplicemente questi non vengono considerati nella costruzione delle regole**;
- Anche per gli attributi interagenti abbiamo le stesse performance (che sono pessime) degli alberi decisionali;

- Anche per gli attributi che mancano, all'interno di un attributo, abbiamo lo stesso problema degli alberi decisionali e si possono utilizzare più o meno le stesse tecniche, con la differenza che se una regola coinvolge un attributo che manca nell'istanza di test, è difficile ignorare la regola e procedere con le successive dell'insieme;
- Nel caso di classi non bilanciate, possiamo risolvere meglio questo problema con le regole di classificazione rispetto agli alberi decisionali, in quanto dando un ordinamento sulle regole, possiamo comunque dare priorità a delle classi, che nel training sono poco rappresentate;
- Come gli alberi decisionali, anche le regole di classificazione hanno un'ottima prestazione nella fase di classificazione, in quanto una volta costruite le regole, valutare le condizioni è molto efficiente → da notare, che la costruzione delle regole, si avvale di algoritmi greedy.

Classificatori Nearest Neighbor

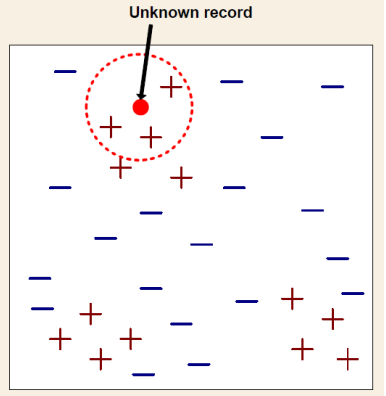
Un altro tipo di classificatore che possiamo costruire, sono i **classificatori Nearest Neighbor** → l'idea di questi classificatori è: noi abbiamo delle istanze classificate e abbiamo una nuova istanza e l'idea alla base di questi classificatori è di associare tale nuova istanza (alle istanza classificate), dandogli la classe dell'istanza più simile che abbiamo nel training set. Tali classificatori, quindi, si basano sulla nozione di **distanza** (detta anche **similarità**) **definita sugli attributi**. Data, quindi, un'istanza di test si va a calcolare la sua distanza (ovvero la sua similarità) rispetto alle istanze che abbiamo nel training set e li assegniamo la classe dell'istanza più simile.

> 💡 Vi è una variante dei classificatori Nearest Neighbor, ovverosia i classificatori **K-Nearest Neighbor**, in cui si richiede di trovare i K valori vicini.

Dato, quindi, un insieme di Training records e il nostro Test record, l'algoritmo di Nearest Neighbor ci restituirà l'istanza (tra i Training records) più simile al nostro Test record.

Sorge a questo punto spontanea la domanda: "**Come si procede con i classificatori Nearest Neighbor?**" Inizialmente, abbiamo un insieme di istanze etichettate. Ad esempio, possiamo avere delle istanze descritte da due attributi, in modo tale da poterle rappresentare in uno spazio. Abbiamo, inoltre, la nostra

istanza da classificare (che nell'immagine sotto è rappresentata dal pallino rosso) e definiamo una misura di distanza (che sul piano, ad esempio, potrebbe essere la distanza euclidea) e conseguentemente troviamo i K valori più vicini, ovverosia troviamo le K istanze più vicine, guardiamo la loro classe (ovvero guardiamo la classe delle K istanze più vicine) e assegniamo la classe (per esempio, attraverso il criterio di maggioranza) alla nostra istanza da classificare (quindi, se la maggioranza delle K istanze più vicine alla nostra istanza da classificare, sono della classe +, allora assegniamo tale classe anche alla nostra istanza da classificare). Vedi l'immagine sotto per capire meglio questo concetto:



💡 Capiamo immediatamente, che più aumentiamo il valore di K, più estendiamo la ricerca e più K diventa piccolo, minore sarà la ricerca. Quindi, la decisione del valore di K è fondamentale per far funzionare in maniera ottimale l'algoritmo di K-Nearest Neighbor, perchè:

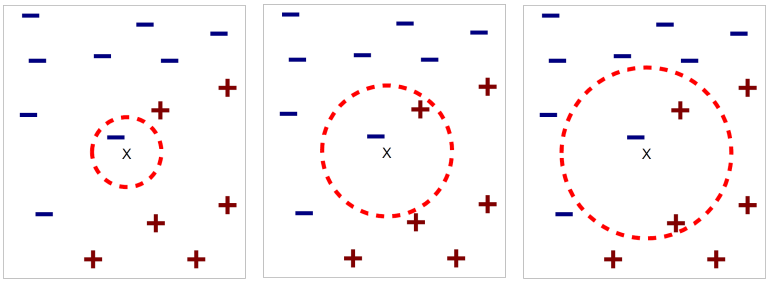
-

se K è troppo piccolo e di conseguenza prendiamo solamente l'istanza più vicina, riscontriamo il problema dell'overfitting;

-

se K è troppo grande, potremmo classificare in maniera errata delle istanze, perchè includiamo tra i suoi vicini, anche delle istanze molto distanti.

Vediamo un esempio, su come cambia la classificazione a seconda del valore di K che andiamo a prendere:



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Possiamo vedere dall'immagine sopra che: se K = 1, allora andiamo a prendere solamente l'istanza più vicina e di conseguenza classifichiamo la nostra istanza con la classe -. Invece, se k = 2, allora andiamo a prendere le 2 istanze più vicine, le quali hanno classi diverse e di conseguenza possiamo scegliere come classificare la nostra istanza. Infine, con K = 3 andiamo a prendere le 3 istanze più vicine e di conseguenza classifichiamo la nostra istanza con la classe +.

"**Cosa fa, allora, il classificatore K-Nearest Neighbor?**" Prende in input il K che gli diamo ed essendo importante scegliere il suo corretto valore, molto spesso esso viene scelto in base alla natura dei dati che stiamo considerando. A questo punto, per ogni istanza che sta nel Test set, si calcola la distanza tra l'istanza e le istanze che abbiamo nel Training set e a questo punto, selezioniamo le K istanze più vicine e scegliamo la classe più frequente. Il corrispondente algoritmo è il seguente:

Algorithm 6.2 The k-nearest neighbor classifier.

1: Let k be the number of nearest neighbors and D be the set of training examples.
2: **for** each test instance $z = (\mathbf{x}', y')$ **do**
3: Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every example, $(\mathbf{x}, y) \in D$.
4: Select $D_z \subseteq D$, the set of k closest training examples to z.
5: $y' = \underset{v}{\text{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
6: **end for**

Da notare, che potrebbe anche esserci un **voto** (ovvero 'argmax'), il quale calcola la distanza più frequente, ma potrebbe essere anche un voto pesato rispetto alla distanza. Quindi, non solo guardo quante istanze ci sono per ogni classe, bensì potremmo anche considerare le distanze.

Infine, diciamo che le caratteristiche di tali classificatori sono:

- queste tecniche di K-Nearest Neighbor sono molto utilizzate in ambito spaziale:
- rispetto ai classificatori Rule-based e agli alberi decisionali, il classificatore Nearest Neighbor è un **Instance-based learning**, ovverosia non costruire

un modello generale, ma permette di classificare un'istanza rispetto ad una definizione di prossimità con le istanze che abbiamo a disposizione nel Test set. Quindi, **non abbiamo un modello generale da poter riutilizzare, bensì abbiamo un classificatore strettamente dipendente dalle istanze di test**;

- è molto più espressivo degli alberi decisionali, perchè ci permette di costruire dei decision boundary di qualsiasi forma;
- da un lato è più efficiente rispetto agli alberi decisionali, in quanto non richiede di costruire un modello, dall'altro lato è leggermente meno efficiente, in quanto per ogni istanza di test dobbiamo calcolare la prossimità rispetto alle istanze presenti nel Training set e successivamente ordinarle, in modo tale da riuscire a prendere le K istanze più vicine;
- può gestire, in maniera semplice, attributi che interagiscono tra di loro, in quanto non consideriamo un attributo alla volta, bensì consideriamo tutti gli attributi insieme;
- **il calcolo della distanza richiede che tutti gli attributi abbiano un valore**;
- la **robustezza è strettamente collegata al valore di K**, in quanto la scelta del valore di K è essenziale sia per garantire che la classificazione sia robusta, sia per non avere delle previsioni errate.

> 💡 Lasciando da parte per un istanze le differenze tra i classificatori, dobbiamo capire che in alcuni casi è migliore adottare un albero decisionale (soprattutto se vogliamo costruire un modello, che sia anche esplicativo), mentre se abbiamo delle istanze di dati, in cui non riusciamo ad avere degli attributi ben separati tra loro e abbiamo una minore conoscenza della struttura dei dati, allora in questo caso utilizzare i Nearest Neighbor è la soluzione migliore.

Notiamo, inoltre, che **tutti i classificatori visti fino a questo momento hanno una caratteristica in comune, ovvero: sono tutti classificatori, che a fronte di un'istanza, restituiscono una classe singola e quindi, hanno una risposta "sicura" per la nostra istanza**. In alcuni casi, però, dare una risposta sicura oppure dare una risposta univoca non è corretto ed è per questo motivo, che sono stati introdotti i **classificatori Bayesiani**.

Classificatori Bayesiani

I classificatori Bayesiani cercano di catturare tutte quelle situazioni in cui vi è incertezza. Ad esempio, gli attributi e le etichette (che diamo alle classi) non sono del tutto affidabili, oppure l'insieme degli attributi che abbiamo scelto, non è complessivamente rappresentativo delle classi → quindi, dobbiamo essere in grado di gestire il caso, in cui non vi è sicurezza dell'etichetta che vogliamo dare ad una classe, ovvero dobbiamo essere in grado di gestire i casi, in cui al massimo possiamo dire, con una certa misura di probabilità, che un'istanza appartenga ad una certa classe. **I classificatori bayesiani, quindi, sono classificatori di tipo probabilistico e per ogni possibile classe, ci danno qual è la probabilità che la nostra istanza appartenga a ciascuna possibile classe**.

> 💡 Naturalmente, **un classificatore bayesiano lo possiamo trasformare in un classificatore deterministico**, decidendo di andare a prendere la classe più probabile tra quelle possibili.

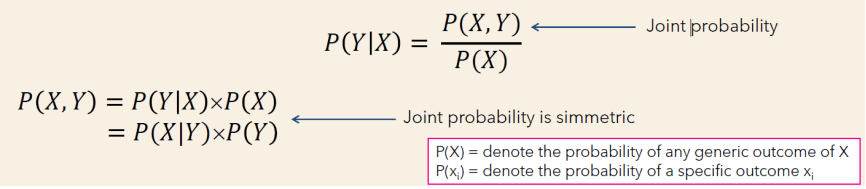
Per comprendere i classificatory bayesiani, dobbiamo riprendere alcuni concetti:

- la **probabilità** di un evento ci dice quant'è probabile che si verifichi l'evento stesso e di conseguenza, consideriamo la probabilità come la **frequenza relativa** dell'evento;
- le variabili che hanno probabilità associate a ogni possibile risultato, sono note come **variabili casuali**;
- per costruire un classificatore bayesiano ci interessa saper calcolare la **probabilità congiunta**, ovvero: quando abbiamo due variabili casuali che prendono K valori distinti, qual è la probabilità di osservare che una variabile X assuma valore X1 e che una variabile Y assuma valore Y1. La probabilità congiunta si ottiene come il rapporto tra:

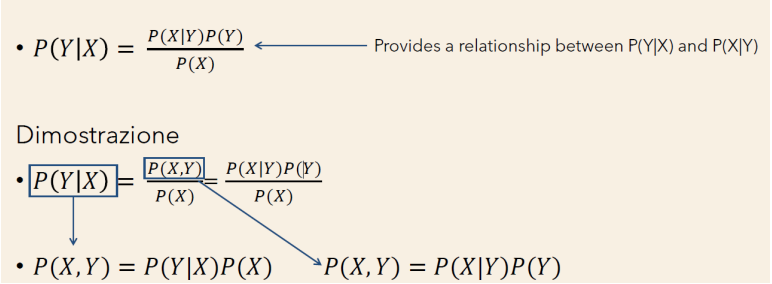
  numero di volte in cui osserviamo tale situazione (X = X1 e Y = Y1) **/ numero totale di occorrenze**.

  A questo punto, data la probabilità congiunta e la sommo rispetto alla probabilità di una delle variabili, allora otteniamo la probabilità di osservare una variabile X indipendentemente da Y, ovvero riusciamo ad osservare solamente la probabilità di X = X1 → questa probabilità prende il nome di **probabilità marginale**.

- un'altra misura di probabilità che ci interessa è la **probabilità condizionata**, ovvero: vogliamo calcolare la probabilità di osservare la variabile Y, quando la variabile X prende un particolare valore, ovverosia la probabilità di osservare Y condizionato al valore di X. Ricordiamoci, allora, la formula della probabilità condizionata:



Alla base dei classificatori bayesiani vi è il **Teorema di Bayes**, che era il seguente:



Utilizziamo il Teorema di Bayes, perchè siamo interessati a calcolare la probabilità di osservare una certa etichetta Y, a fronte del fatto che le nostre istanze abbiano determinati valori sugli attributi X. Quindi, vogliamo conoscere la cosiddetta **probabilità a posteriori**, ovvero vogliamo sapere la probabilità che la nostra classe sia Y, dato l'insieme dei valori degli attributi. Però, quello che conosciamo dai dati è la probabilità che una certa istanza abbia una certa etichetta. Quindi, applichiamo il Teorema di Bayes per andare a calcolare la probabilità condizionata delle classi, che appunto rappresenta la misura di likelihood di osservare un certo insieme di valori degli attributi, data la distribuzione delle istanze che appartengono alla classe Y.

💡 **Il problema è che più aumentano i possibili valori dei nostri attributi e più aumenta il numero di attributi, allora anche con la probabilità condizionata possiamo avere delle stime**, le quali non sono molto significative e di conseguenza, le performance del classificatore bayesiano diminuiscono → Per questo motivo, è stata introdotta l'**assunzione di Bayes**, la quale tenta di darci delle classi di probabilità che siano comunque verosimili, anche in presenta di un elevato numero di attributi.

Tale assunzione di Bayes, si basa innanzitutto sul fatto che:

- gli **attributi X** siano tra loro **indipendenti** → questo ha come conseguenza diretta, il fatto che invece di dover calcolare tutte le possibili combinazioni dei valori degli attributi, al fine di calcolare P(X|Y), posso trasformare quest'ultima probabilità (ovvero P(X|Y)) come il prodotto delle probabilità condizionate dei singoli attributi X.

$$P(\boldsymbol{x}|y) = \prod_{i=1}^{d} P(x_i|y)$$

  Quindi, posso calcolare le probabilità condizionate dei singoli attributi X in maniera indipendente e successivamente, al fine di calcolare la specifica probabilità condizionata sullo specifico insieme di attributi, possiamo semplicemente fare il prodotto delle singole probabilità condizionate.

I classificatori bayesiani hanno le seguenti caratteristiche:

- funzionano molto bene, quando ci sono delle informazioni complete, ma richiedono che gli attributi siano indipendenti tra di loro e di conseguenza, non abbiamo ancora un sistema che funzioni bene, quando abbiamo gli attributi correlati tra di loro;
- sono **robusti** rispetto a **punti di rumore isolati**, perchè questi (ovvero i punti di rumore isolati) non impattano in maniera significativa le probabilità condizionate;
- sono **robusti** rispetto agli **attributi irrilevanti**, perchè questi non impattano in maniera significativa le probabilità condizionate;
- la costruzione del **modello e** la **classificazione** sono **molto efficienti**.

💡 Ovviamente, anche in questo caso, la scelta di utilizzare l'albero decisionale oppure il classificatore bayesiano, dipende dal problema stesso e dal risultato che vogliamo ottenere. Ricordiamo, inoltre, che un classificatore bayesiano lo possiamo sempre trasformare in un classificatore decisionale.

Accenniamo, al fatto che esistono anche le **Reti Bayesiane**, le quali sono una evoluzione dei classificatori bayesiani (i quali hanno come presupposto fondamentale, che gli attributi siano indipendenti tra di loro) e **consistono sostanzialmente nel "rilassare" l'ipotesi di indipendenza fra gli attributi**.

Un'altra tecnica di classificazione è la **Logistic Regression**, in cui oltre all'assegnamento delle classi, abbiamo anche la costruzione di un modello.

Clustering

Il clustering è un'altra attività che viene fatta nel campo dell'analisi dei dati e **consiste sostanzialmente nel raggruppare e/o dividere in gruppi le nostre istanze, in modo tale che le istanze simili stiano nello stesso gruppo, in modo tale (anche) da dare all'utente, una rappresentazione più compatta delle informazioni**. L'obiettivo del clustering tipicamente è duplice:

1. da una parte, utilizziamo il clustering per "**capire**", ovvero per mettere insieme le istanze che hanno delle caratteristiche simili → quando facciamo un clustering per "capire", possiamo anche dire che essenzialmente un cluster non è altro che una classe;
2. dall'altra parte, utilizziamo il clustering per **astrarre** i dati dai dettagli e capire sostanzialmente quali sono le classi di oggetti.

> 💡 Già da questi primi concetti, possiamo capire che se facciamo una comparazione rispetto alle tecniche di classificazione, nel clustering non abbiamo delle classi pre-costituite (che già sappiamo) a cui assegnare le istanze, bensì tramite il clustering scopriamo quali sono le classi e quindi le etichette più interessanti.

Quando si parla di clustering, quindi, si parla anche di **prototipo**, perchè l'idea è: a fronte di migliaia di istanze identifichiamo, ad esempio, una decina di prototipi che rappresenteranno le categorie informative più interessanti. In questo senso, viene utilizzata la tecnica di '**Summarization**', **il cui obiettivo è di ridurre il numero di istanze ad un insieme molto più compatto di prototipi**.

Riassumendo, quindi, possiamo dire che l'idea alla base del clustering è di trovare i gruppi (ovvero i **cluster**) simili e una volta fatto ciò, **andremo anche ad individuare il rappresentante per ciascun gruppo**. Dovremmo, allora, definire per bene il concetto di similarità, in quanto la caratteristica fondamentale dei cluster è: **gli oggetti all'interno di uno stesso cluster sono molto simili tra di loro ed invece sono estremamente diversi rispetto agli oggetti presenti negli altri cluster**.

Come abbiamo detto sopra, possiamo vedere un'analogia con la classificazione, perchè una volta individuati i cluster, possiamo decidere di dare un'etichetta a ciascun cluster individuato, ossia sostanzialmente dare

un'etichetta alle istanze che appartengono allo stesso cluster. Notiamo, però, che vi è una differenza fondamentale dal punto di vista pratico (tra il cluster e la classificazione), ossia: mentre la classificazione è un'**operazione** chiamata **supervised** (in quanto prima di procedere con la classificazione, abbiamo la necessità che qualcuno abbia già etichettato le istanze del training set), il clustering invece è un'**operazione** chiamata **unsupervised**, ovvero non c'è bisogno che qualcuno etichetti le istanze prima di iniziare il clustering (quindi possiamo iniziare a fare l'attività di clustering sulle istanze raw).

Esistono diversi tipi di clusterizzazione, ma prima di vedere i diversi tipi di clusterizzazione, dobbiamo fare una precisazione: Il termine '**clustering**' è utilizzato per identificare una collezione di cluster. Quindi, utilizziamo questo termine sia per identificare l'attività di clustering (ovvero di costruzione dei cluster) sia per indicare il risultato dell'operazione. Una volta fatta questa precisazione, possiamo analizzare le diverse tipologie di clusterizzazione, che sono:

- **Gerarchico** o **Partizionale** → il clustering partizionale è una semplice suddivisione del nostro Data set, in cluster che tra loro non siano sovrapposti → questo vuol dire, che **ogni oggetto sta in uno ed uno solo dei cluster in cui abbiamo suddiviso il Data set**. Potremmo, però, avere la necessità di realizzare un clustering gerarchico, in cui abbiamo una struttura innestata, ovverosia partiamo dalla radice (che sarà un cluster che contiene tutti gli oggetti di tutte le nostre istanze) e poi abbiamo una struttura ad albero, in cui ad ogni livello dell'albero ci è un clustering partizionale, la cui unione dà il cluster che si trova al nodo padre;
- **Esclusivi** o **Sovrapposti** o **Fuzzy**→ il clustering esclusivo assegna ogni oggetto ad un singolo cluster, mentre con il clustering sovrapposto abbiamo che ogni oggetto viene assegnato simultaneamente a molteplici gruppi (pensiamo, ad esempio, agli studenti lavoratori, che possono essere assegnati simultaneamente ad un cluster che rappresenta gli studenti e ad un cluster che rappresenta i lavoratori). Nel caso del clustering fuzzy, invece, non abbiamo un assegnamento preciso (nel senso, che non abbiamo un assegnamento unico), bensì con questa tipologia di clustering abbiamo che ogni oggetto appartiene ad ogni cluster, ma con un **grado di appartenenza**, che va a 0 a 1 → questo significa, che ciascuna istanza del nostro Data set sarà assegnata ad ogni cluster e se il grado è 0, significa che in realtà non vi ci appartiene, mentre se il grado è 1, allora l'istanza

effettivamente appartiene al cluster e qualsiasi altro valore compreso nell'intervallo [0, 1] significa che vi è una possibilità di appartenenza.

> 💡 I clustering fuzzy possono essere ricondotti a dei clustering esclusivi, assegnando a ciascun oggetto una ed una sola classe, che sarà ovviamente quella con il grado di appartenenza più alto.

- **Completo** o **parziale** → con il clustering completo abbiamo che ciascuna istanza viene assegnata ad (almeno) un cluster, mentre nel clustering parziale potremmo avere delle istanze che non sono assegnate ad alcuna classe (per esempio, se le istanze vengono identificate come del rumore oppure delle outliners, esse non vengono assegnate ad alcuna classe).

> 💡 **Per ciascuna tipologia di clustering esistono degli algoritmi diversi**, che ci permettono di eseguire la clusterizzazione in maniera ottimale.

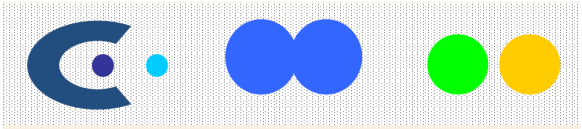
Quando parliamo di cluster, vi sono delle proprietà che vogliamo rispettare e di conseguenza, tanto più tali proprietà vengono soddisfatte, tanto migliore sarà il clustering che otteniamo. In particolare, le proprietà sono le seguenti:

- essere **ben separato** → come abbiamo detto precedentemente, un cluster è un gruppo di oggetti, tale per cui la distanza tra gli oggetti che appartengono allo stesso cluster è più piccola rispetto alla distanza verso gli oggetti, che appartengono agli altri cluster. Quindi, **tanto più la distanza tra i cluster è marcata, tanto più i cluster saranno ben separati**.

- i cluster possono essere classificati come **prototype-based** → ovverosia i cluster sono basati sull'individuazione del prototipo. In questo caso, per definire i cluster dobbiamo individuare dei prototipi (ovverosia degli oggetti più significativi e quindi per ogni cluster definiamo un prototipo, il quale diventa il rappresentante del cluster. Solitamente, il prototipo è il centro del cluster) e assegniamo le istanze ad un cluster, in modo tale che la distanza tra l'oggetto che stiamo assegnando al cluster e il prototipo del cluster, sia

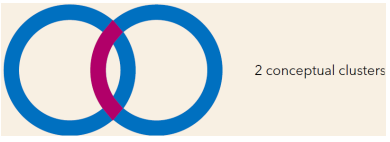
la **minore** possibile. L'idea, quindi, è che **una volta che abbiamo definito il rappresentante per ciascun cluster, abbiamo che le istanze vengono assegnate al cluster, il cui prototipo è il più vicino all'istanza stessa**.

- i cluster possono essere classificati come **graph-based** → ovverosia i dati vengono rappresentati come dei grafi e a questo punto, il cluster viene definito come una **componente connessa**. Pertanto, **il cluster è semplicemente un gruppo di oggetti connessi tra di loro, ma essi** (ovvero gli oggetti) **non hanno connessioni con l'esterno**. Nel caso dei cluster graph-based, quindi, deve esistere non tanto una nozione di prossimità o di similarità, bensì deve esistere una nozione di **continuità** → da notare, che tramite una clusterizzazione graph-based, siamo in grado di rappresentare anche i cluster con delle forme diverse e più complesse rispetto a quelle del prototype-based.
- i cluster possono essere classificati come **density-based** → in questo caso, quindi, non utilizziamo più una nozione/classificazione né basata sulla distanza né sulla contiguità, bensì utilizziamo una classificazione basata sulla densità. In questo caso, quindi, il cluster è una regione molto densa, che è circondata da una regione a bassa densità. Queste tecniche, quindi, vengono utilizzate quando i cluster sono irregolari e quando vi è la presenta di rumore e/o outliner.



6 density-based clusters

- i cluster possono essere classificati come **shared properties** → si tratta di una tipologia molto più versatile rispetto alle tre precedenti, in quanto si basa sul fatto di condividere delle proprietà e quindi, in questo caso, si parla di **cluster concettuali**. In questo caso, allora, il cluster è un insieme di istanze che condividono una determinata proprietà (per esempio, clusterizziamo gli studenti rispetto all'anno di immatricolazione e quindi tutti gli studenti che condividono l'anno di immatricolazione vengono messi nello stesso gruppo).



2 conceptual clusters

Notiamo immediatamente, che nei cluster concettuali non vale né la nozione di distanza, né la nozione di densità e nemmeno la nozione di contiguità e di grafo. Possiamo, allora, ottenere delle nuove forme di cluster, ma il problema è che (nell'esempio) stiamo dividendo in base ad una sola proprietà, il che non è particolarmente espressivo, in quanto non stiamo individuando dei cluster che non conosciamo (quindi non stiamo individuando dei nuovi cluster).

K-means

L'algoritmo di K-means è basato sull'idea di trovare un prototipo e di conseguenza si tratta di una tecnica prototype-based. L'idea, quindi, è di trovare una suddivisione in K cluster (dove K è un parametro che imponiamo dall'alto) e ciascun cluster viene rappresentato (dato che si tratta di una tecnica prototype-based) dal suo **centroide**, ovverosia dal suo elemento centrale all'interno del cluster. Tipicamente, il centroide non corrisponde ad alcuna istanza reale e di conseguenza, troviamo un prototipo che rappresenta tutte le istanze del cluster, ma tipicamente il prototipo non coincide con alcuna istanza del cluster. Sorge a questo punto spontanea la domanda: "**Come funziona effettivamente il K-means?**" Innanzitutto, dobbiamo avere il parametro K, il quale viene imposto dall'esterno. A questo punto, dato il nostro spazio di riferimento, selezioniamo K punti casuali come centroidi iniziali. Fino a quando non raggiungiamo un punto fisso, ovverosia fino a quando i centroidi non cambiano, assegniamo i punti ai K cluster utilizzando la distanza rispetto al centroide e ricalcoliamo il centroide di ogni cluster. Quindi, una volta che abbiamo assegnato i punti, riposizioniamo il centroide al centro del cluster. Questo definizione può essere descritta in maniera più formale attraverso il seguente algoritmo:

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

Una volta che abbiamo capito l'algoritmo del K-means, evidenziamo le proprietà di tale algoritmo, che sono:

- **converge sempre ad una soluzione** → quindi, dopo un certo numero finito di iterazioni, arriviamo al punto in cui non è più necessario spostare i centroidi e di conseguenza non sarà più necessario ricostruire il cluster. In alcuni casi, pur avendo la garanzia di arrivare ad un punto fisso, potrebbe succedere che il numero di iterazioni è eccessivamente elevato e di conseguenza, si può rilassare la condizione di terminazione attraverso un threshold (ovvero una soglia).

Possiamo notare immediatamente, alcune problematiche relative all'utilizzo dell'algoritmo K-means, che possono essere osservare tramite le seguenti domande:

- **La distanza dai centroidi come viene calcolata dall'algoritmo?** Per prima cosa, quindi, dobbiamo stabilire la nozione di distanza;
- **Come viene scelto il valore di K, dato che non conosciamo effettivamente i dati?**

Rispondiamo a queste due domande e iniziamo con la prima: Per assegnare un punto al suo centroide più vicino, abbiamo bisogno di una nozione di prossimità (ovvero di distanza), che misura sostanzialmente qual è il centroide più vicino. Esistono diverse alternative per definire la nozione di prossimità:

- **distanza euclidea** → è molto intuitiva nel caso di 2 dimensioni e la relativa formula è:

$$\text{Euclidean distance: } d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

- **cosine similarity** → misura basata sul coseno e la relativa formula è:

$$\text{Cosine similarity: } CS(A,B) = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2}\sqrt{\sum_{i=1}^{n} {B_i}^2}}$$

- altre possibili misure per la distanza, sono la distanza di Manhattan e la misura di Jaccard (la quale viene utilizzata nei documenti).

L'obiettivo del clustering, quindi, diventa di andare a minimizzare le distanze (che abbiamo appena sopra citato) e di conseguenza, (il clustering) lo possiamo rappresentare anche come una **funzione obiettivo, la quale dipende dalla nozione di prossimità tra un punto e il centroide del proprio cluster**. **In questo caso, la qualità di un cluster viene definita come la capacità di**

minimizzare la distanza di ciascun punto dal proprio centroide più vicino → questa proprietà viene rappresentata:

- nel caso dello spazio euclideo, come la **somma degli errori quadratici** (che prende il nome di **SSE**);
- nel caso dei documenti, prende il nome di **coesione →** nel caso dei documenti, in cui non possiamo utilizzare una nozione di distanza euclidea, utilizzeremo altre formule. In particolare, spesso i documenti vengono rappresentati come una matrice (chiamata Document-term matrix), dove per ogni termine del documento, viene misurata la sua frequenza e la qualità del cluster viene misurata tramite la coesione.

Approfondiamo meglio il concetto di **SSE**: nel caso dello spazio euclideo, ovverosia nel caso in cui gli attributi possono essere rappresentati come coordinate in uno spazio euclideo, la misura della qualità di un cluster viene individuata dall'**SSE** (ovvero la **somma degli errori quadratici**). Quindi, andiamo a calcolare l'errore in ciascun Data Point, ovverosia andiamo a calcolare la distanza euclidea di ciascun elemento rispetto al centroide più vicino, e a questo punto calcoliamo la somma degli errori quadratici.

> 💡 **Se dobbiamo confrontare due cluster, andremo a prediligere il cluster che ha un valore di SSE più piccolo**, in quanto i propri centroidi danno una migliore rappresentazione dei punti, che sono all'interno del cluster stesso. Quindi, **la clusterizzazione migliore sarà quella che ha il valore di SSE più piccolo**.
> Notiamo, allora, che se usiamo il K-means, ad ogni iterazione dell'algoritmo, il valore dell'SSE dovrebbe diminuire, fino ad arrivare alla clusterizzazione finale, la quale avrà il valore di SSE più piccolo.

Quindi, quello che inizialmente fa l'algoritmo di K-means, è di generare e scegliere casualmente K centroidi e così viene prodotto il primo clustering. **Questa scelta casuale dei K centroidi non è un'operazione banale, in quanto a seconda di come partiamo (ovvero a seconda di quali centroidi scegliamo inizialmente) possiamo arrivare o meno alla soluzione ottima. Notiamo, infatti, che il K-means ci garantisce di arrivare alla terminazione, ma non è detto che la soluzione finale sia la soluzione ottimale** → gli algoritmi, quindi, posizionano casualmente i centroidi iniziali e talvolta il K-means utilizza molteplici iterazioni (ovverosia si fanno più iterazioni del K-means), partendo da

punti randomici diversi e successivamente si vede la qualità del clustering finale e si prende il cluster con il valore SSE più basso → **la prima soluzione al problema della scelta dei centroidi, quindi, consiste nell'eseguire molteplici esecuzioni del K-means**.

Un'altra soluzione al problema della scelta dei centroidi, può essere quella di andare a costruire un **clustering gerarchico** e utilizzare tale clustering gerarchico come punto di partenza per posizionare i centroidi (e non come risultato). Questa soluzione si può applicare solamente quando:

- i dati che abbiamo a disposizione sono semplici;
- cerchiamo un valore di K, che è piccolo rispetto alla dimensione del nostro Data set.

Capiamo immediatamente, che il clustering gerarchico non può essere fatto sull'intero Data set, bensì viene fatto solamente su un **sample** e in questo caso, quindi, definire il sample migliore può risultare problematico.

Un'altra tecnica che si può utilizzare per scegliere i centroidi iniziali, è un'evoluzione del K-means e prende il nome di **K-means++** → l'idea alla base di questa tecnica consiste in: scegliere un punto iniziale, ovverosia il primo centroide, in maniera casuale e a questo punto, il secondo centroide viene scelto il più distanze possibile da ogni altro centroide che abbiamo già selezionato. Quindi, il K-means++ sceglie come prima cosa un punto iniziale casuale, il quale di fatto rappresenta il nostro primo centroide (che appunto viene scelto in maniera casuale). Poi, per un certo numero di tentativi (il quale sarà un parametro che dovrà essere impostato) per ognuno degli N punti che abbiamo, selezioniamo la distanza che vi è tra il punto e il suo centroide più vicino (ovvero per ognuno degli N punti, andiamo a selezionare/calcolare la distanza che c'è tra il punto e il suo centroide più vicino) e assegniamo a ciascun punto, una probabilità di essere selezionato come nuovo centroide (e naturalmente tale probabilità dipenderà dalla distanza dal centroide più vicino) ed infine andiamo a selezionare un nuovo centroide (in maniera casuale) sulla base della probabilità, che dipende dalla distanza dal centroide corrente diviso (/) la distanza da tutti gli altri centroidi. Possiamo riassumere questi passi con il seguente algoritmo:

1. Select an initial point at random to be the first centroid
2. **for** i = 1 to *number of trials* do
3. for each of the N points, x_i, $1 \leq i \leq N$, find the minimum squared distance to the currently selected centroids, $C_1, \ldots, C_j$, $1 \leq j < k$, i.e., $\min_j d^2(C_j, x_i)$ → compute the distance of each point to its closest centroid
4. Randomly select a new centroid by choosing a point with probability proportional to $\frac{\min_j d^2(C_j, x_i)}{\sum_i \min_j d^2(C_j, x_i)}$ → assign to each point a probability proportional to its distance and pick up a centroid from the remaining points using the weighted probabilities.
5. **end for**

I **problemi** di questa tecnica sono

- **richiede di stabilire qual è il valore K**;
- nonostante cerchiamo di impostare il valore ottimale di K, **possiamo ritrovarci nella situazione in cui alcuni cluster rimangono vuoti** → una possibile soluzione a questo problema consiste nel manipolare la scelta dei centroidi oppure nell'andare a suddividere un cluster (molto grande) che è stato trovato in due cluster.
- nell'andare a selezionare il secondo oppure il terzo o il quarto centroide più distanze possible, potremmo avere un outlier e in più potrebbe essere complicato prendere sempre il centroide più distanze, in quanto ogni volta dobbiamo calcolare la distanza. Quindi, **l'algormitmo K-means è molto suscettibile alla presenza di outliners** → per risolvere questo problema, quindi, il nostro obiettivo è di individuare gli outliners ed eliminarli, prima di iniziare con l'esecuzione dell'algoritmo K-means.

Un'altra possibile soluzione consiste nell'applicare un **Post-processing**, ovverosia: una volta che abbiamo trovato la nostra clusterizzazione, andiamo ad analizzare effettivamente cosa abbiamo ottenuto ed eventualmente andremo a modificare e ripetere parte dell'algoritmo solamente su determinate zone. Il Post-processing, quindi, si applica per tentare di ridurre ulteriormente il valore dell'SSE, in quanto immaginiamo di aver eseguito l'algoritmo K-means e otteniamo una clusterizzazione, che ha un valore di SSE che per noi è troppo elevato. Possiamo, allora, tentare di ridurre tale valore di SSE andando a:

- **incrementare il numero di cluster**, attraverso una **tecnica di splitting**, che consiste in: prendere un cluster che ha un valore di SSE molto elevato e di conseguenza, che contribuisce in maniera sostanziosa al calcolo dell'SSE totale e andiamo a suddividerlo in due cluster;
- oppure possiamo tentare di **introdurre un nuovo centroide** e ri-eseguire il clustering tenendo in considerazione questo nuovo centroide.

- oppure andiamo a **ridurre il numero di cluster**, in quanto vi è la possibilità che abbiamo utilizzato un valore di K eccessivamente grande durante l'esecuzione dell'algoritm K-means. Per ridurre il numero di cluster svolgiamo un'**operazione di Merging**, ovverosia prendiamo due cluster molto vicini tra di loro e tentiamo di combinarli;
- oppure andiamo a **disperdere un cluster**, andando a rimuovere un centroide, al fine di ri-assegnare i punti ad un nuovo cluster.

L'ultima possibile soluzione consiste nell'applicare il **Bisecting K-means**, che consiste in un'ulteriore variazione dell'algoritmo K-means e consiste nell'idea che: per ottenere K cluster andiamo a suddividere i punti inizialmente in due cluster e successivamente andiamo a valutare i due cluster ottenuti, al fine valutare se uno dei due cluster deve essere ulteriormente suddiviso. Abbiamo, quindi, una suddivisione incrementale in cui analizziamo i cluster che abbiamo ottenuto e ogni volta facciamo una bi-sezione (ovvero dividiamo in due un cluster) e analizziamo se è necessario dividere ulteriormente un cluster. Con questa soluzione, chiaramente il fulcro centrale consiste nel capire se un cluster deve essere suddivido oppure no e questo lo fa attraverso diverse tecniche, che sono:

- **suddividere il cluster più grande** → se a fronte di una bi-sezione, abbiamo un cluster molto grande ed un cluster più piccolo, andiamo a suddividere ulteriormente il cluster più grande;
- **suddividere il cluster con il valore SSE più alto**;
- utilizzare un criterio, che è un'unione tra i due criteri appena sopra citati.

> 💡 Chiaramente, a seconda dei tre criteri che utilizziamo, otterremo un risultato diverso in termini di cluster ottenuti.

Analizziamo, infine, quali sono gli svantaggi e gli svantaggi dell'algoritmo K-means. I **vantaggi** sono:

- l'algoritmo K-means è sicuramente molto semplice ed ha anche il vantaggio di poter essere applicato ad un'ampia varietà di dati;
- è abbastanza efficiente, in quanto non è computazionalmente oneroso.

Gli **svantaggi**, invece, sono:

- essendo basato sulla distanza, la presenza di outliners e rumore può andare a modificare la struttura dei cluster;
- abbiamo la necessità di definire, per ognuno dei nostri dati, una nozione di centro;
- l'algoritmo K-means ha difficoltà a trovare i cluster, quando quest'ultimi non hanno una forma sferica (o comunque **globulare**);
- l'algoritmo K-means fa fatica a definire i cluster, quando quest'ultimi hanno delle **dimensioni** e **densità** molto diverse.

Hierarchical Clustering

Invece di costruire una clusterizzazione piatta (ovverosia una clusterizzazione ad un livello), un'altra possibilità è quella di costruire una **clusterizzazione gerarchica**. In questo caso, quindi, abbiamo:

- una **serie di cluster innestati tra di loro**;
- **non abbiamo la necessità di stabilire un numero di cluster**, dato che partiamo da un cluster iniziale che contiene tutti i dati e successivamente **continuiamo a suddividere i dati in cluster, fino a quando non raggiungiamo dei cluster che contengono un'unica istanza**.

> 💡 Una clusterizzazione gerarchica la possiamo riportare ad una clusterizzazione piatta, selezionando il livello dell'albero che ci interessa.

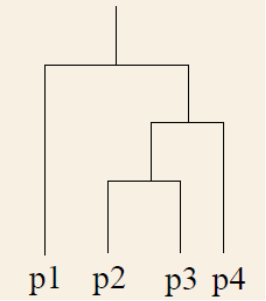
La clusterizzazione gerarchica è particolarmente utile per rappresentare le **tassonomie** (in quanto con le tassonomie si parte da un concetto generale e via via si va a specializzare i concetti, fino a quando non si arriva ad una definizione, che descrive in maniera univoca ciascuna istanza). Notiamo, inoltre, che la clusterizzazione gerarchica può essere di tue tipi:

1. **Approccio agglomerativo** (**bottom-up**) → approccio maggiormente utilizzato e consiste nel partire inizialmente dalle singole istanze (che sono dei cluster individuali) e ad ogni step, vado ad unire i due cluster più vicini → chiaramente, quindi, anche con questo approccio abbiamo bisogno di una nozione di prossimità;

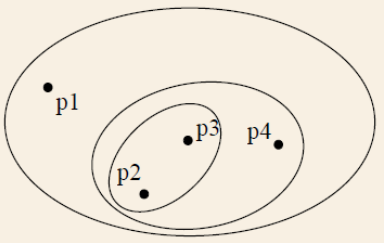
2. **Approccio divisivo** (**top-down**) → attraverso questo approccio, partiamo da un unico cluster, il quale rappresenta tutte le istanze e mano a mano andiamo a dividerle, fino a quando non otteniamo un cluster che contiene un'unica istanza → con questo approccio, quindi, abbiamo bisogno di un criterio per eseguire lo split.

Solitamente i cluster gerarchici si rappresentano attraverso una forma ad albero (che per esattezza non sono degli alberi), che prende il nome di **dendrogrammi**, i quali rappresentano:

- la **suddivisione** dei cluster;
- le **relazioni** che vi sono tra i cluster;
- l'**ordine** in cui vengono eseguite le operazioni di Merge e di Split.



Una rappresentazione alternativa, che viene utilizzata solamente quando i punti (ovvero le nostre istanze) sono bidimensionali, è una rappresentazione tramite un diagramma a cerchi .



Sorge a questo punto spontanea la domanda: "**Come si fa a costruire un clustering gerarchico tramite la tecnica agglomerativa?**" Abbiamo già detto, che tramite la tecnica agglomerativa, partiamo dai singoli punti come cluster e uniamo i due cluster più vicini fino a quando non rimane un solo cluster. L'algoritmo per riuscire a fare ciò è il seguente:

1. Calcoliamo la **matrice di prossimità**;

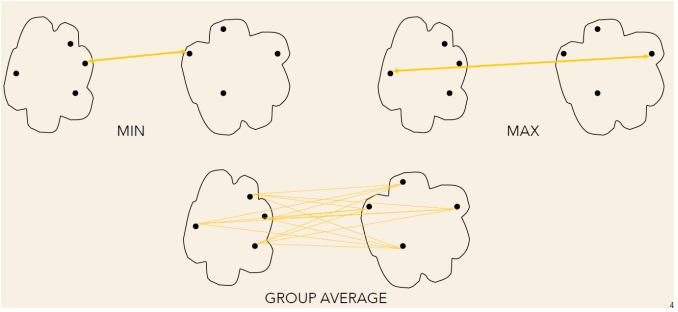
2. Ogni punto dati è un cluster;
3. Ripetere fino a quando non rimane un solo cluster:
    a. Unire i due cluster più vicini;
    b. Aggiornare la matrice di prossimità.

> 💡 Quindi, **attraverso la tecnica agglomerativa per costruire un clustering gerarchico, l'operazione fondamentale è di calcolare la prossimità tra due cluster**.

Anche in questo caso, la nozione di prossimità può essere diversa e a seconda della nozione che utilizziamo, vi sono diverse tecniche di agglomerazione gerarchica. In particolare, possiamo avere che:

- la prossimità tra cluster viene definita utilizzando una **rappresentazione a grafo del cluster** e in questo caso, abbiamo tre possibilità:
    - tecnica **MIN** (detta anche **Single Link**) → definisce la prossimità tra cluster, come la prossimità tra i due punti più vicini, che sono in due cluster differenti. La tecnica MIN, quindi, va a calcolare la distanza tra i punti più vicini nei due cluster e una volta ottenuta la minima distanza tra tutte le coppie di cluster, verrà eseguito il merge i due cluster, che si trovano alla distanza minima. Il punto di forma di questa tecnica, è che permette di gestire forme non ellittiche, mentre il suo punto di debolezza è che è molto sensibile alla presenza di rumore e/o outliers;
    - tecnica **MAX** (detta anche **Complete Link**) → definisce la prossimità come la prossimità definita tra i punti più distanza tra i due cluster. Quindi, dati i due cluster la tecnica MAX prende la distanza tra i punti più lontani dei due cluster. Il punto di forza di questa tecnica è che è poco suscettibile alla presenza di rumore e/o outliers (dato che va a prendere la distanza tra i punti più lontani), mentre il punto di debolezza è che tende a rompere gli ammassi di grandi dimensioni ed è orientato verso gli ammassi globulari;
    - tecnica **Group Average** → definisce la prossimità come la prossimità media, calcolata per coppie, tra tutti i punti del cluster. Quindi, prendiamo tutti i punti del primo cluster e calcoliamo la distanza rispetto a tutti i punti del secondo cluster ed infine calcolo la distanza media. Questa tecnica è un **compromesso** tra la tecnica MIN e la tecnica MAX.

In particolare, il punto di forza di questa tecnica è che è poco suscettibile alla presenza di rumore e/o outliers (proprio come la tecnica MAX), mentre il punto di debolezza è che è orientata verso gli ammassi globulari (come la tecnica MAX).



- utilizzando, invece, una **rappresentazione a prototipo dei cluster** (come quella del K-means), allora, otterremo una prossimità che è definita rispetto ai centroidi (e non rispetto alle istanze) e quindi possiamo utilizzare l'SSE e prende il nome di **Ward's Method**. Questa tecnica ha lo stesso punto di forza e lo stesso punto di debolezza della tecnica Group Average.

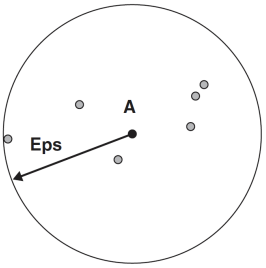
Indichiamo, infine, gli ultimi problemi e limitazioni che vi sono con la tecniche di agglomerazione gerarchica:

- Una volta presa la decisione di unire due cluster, non è possibile annullarla;
- Nessuna funzione obiettivo globale viene minimizzata direttamente;
- I diversi schemi hanno problemi con uno o più dei seguenti aspetti:
  - sensibilità al rumore;
  - rottura di cluster di grandi dimensioni;
  - difficoltà a gestire cluster di dimensioni diverse e di forma non globulare.

DBSCAN

Al contrario del K-means che era una tecnica prototype-based, il DBSCAN è una tecnica **density-based**. In questo caso, il numero di cluster non siamo noi a deciderlo a priori, bensì viene determinato in maniera automatico dall'algoritmo → l'idea del DBSCAN è di andare a **suddividere lo spazio delle nostre istanze rispetto alla densità** e di conseguenza, avremo che ogni cluster rappresenta una regione ad alta densità, la quale è separata da altre regioni che hanno invece una densità minore. In questo caso, quindi, non abbiamo bisogno di una

definizione di similarità o di prossimità, bensì abbiamo bisogno di definire una nozione di densità. Per essere precisi, anche nel DBSCAN abbiamo un concetto simile a quello del centroide, perchè la densità di un punto (all'interno del Data set) viene stimata come: il numero di punti che si trovano un certo raggio prestabilito (che prende il nome di **Eps**) rispetto al punto di cui calcoliamo la densità. Questo concetto è più chiaro con la seguente immagine:



> 💡 Questo ci fa capire, che **con il DBSCAN non abbiamo il parametro K da decidere, bensì abbiamo da decidere il parametro Eps. Inoltre, una volta determinato l'Eps, siccome dobbiamo calcolare la densità, dovremmo definire anche un treasure**.

Tramite, quindi, la nozione di densità definita tramite un approccio center-based rispetto al punto, noi potremmo classificare un punto in tre tipologie:

- **Center points** → punti all'interno di un'area ad alta densità e diciamo che A è un **Core point** si trova all'interno di un cluster density-based, ovverosia: un punto P è un Core point se vi sono almeno **MinPts** punti (da notare che MinPts sarà un parametro che dovremmo definire), che si trovano ad una distanza Eps da A (P incluso) → per dare la definizione di Core point e quindi di cluster, abbiamo bisogno di due parametri:
  - il parametro **Eps**, che ci dà il raggio che consideriamo;
  - il parametro **MinPts**, che ci dà il numero di punti che dobbiamo trovare all'interno della circonferenza, per affermare che l'area è ad alta densità.
- **Border points** → punti che si trovano al bordo di una regione ad alta densità. Esso non è un Core point, ma si trova nelle vicinanze di un Core point, ovverosia il Border point è un punto che si trova all'interno della circonferenza di raggio Eps, ma a sua volta non è un Core point;

- **Noise points** → punti che si trovano in maniera sparsa all'interno della regione. Quindi, il Noise point non è né un Core point né un Border point.

L'idea generale dell'algoritmo DBSCAN è: dati due **Core point** abbastanza vicini, ovverosia che sono all'interno di una distanza Eps l'uno dall'altro, questi vengono messi nello stesso cluster. I **Border point** che sono abbastanza vicini ad un Core point, sono messi nello stesso cluster del Core point (se un Border point si trova sul tra due Core point, dobbiamo decidere a quale Core point assegnarlo). I **Noise point**, invece, vengono scartati. L'algoritmo di DBSCAN, quindi, è il seguente:

1. Etichettare tutti i punti come punti Core point, Border point oppure Noise point;
2. Eliminare i Noise points;
3. Inserire un bordo tra tutti i Core point entro una distanza Eps l'uno dall'altro;
4. Fare di ogni gruppo di Core point collegati un cluster separato;
5. Assegnare ogni Border point ad uno dei cluster dei Core point associati.

> 💡 Questo algoritmo ha complessità temporale pari a O(m * tempo per trovare i punti entro il raggio Eps), mentre la complessità spaziale è pari a O(m), dove 'm' rappresenta il numero di punti.

I **problemi** del DBSCAN sono che abbiamo due parametri da determinare, ovvero l'EPS e il MinPts, i quali hanno una rilevanza sui cluster che vengono determinati (nel senso che andremo a definire i cluster sulla base dei due parametri). Sorge allora spontanea la domanda: "**Come possiamo determinare i migliori valori di Eps e MinPts?**" Per determinare i migliori valori di Eps e di MinPts, l'idea è di guardare alla distanza tra un punto e il suo K-esimo punto più vicino e tale distanza prende il nome di **K-dist** → da notare, che per **tutti i punti che stanno nello stesso cluster, il valore di K-dist dovrebbe essere lo stesso, mentre i punti che non stanno nel cluster** (come ad esempio i Noise point) **avranno un valore di K-dist molto elevato**. Per selezionare i parametri, quindi, dovremmo eseguire i seguenti passi:

1. Calcolare il valore di k-dist per tutti i punti del Data set, per un qualche valore di K;
2. Ordiniamo i valori di K-dist in ordine crescente;

3. Tracciare i valori ordinati;
4. Si noterà una brusca variazione in corrispondenza del valore di k-dist che corrisponde a un valore adeguato per Eps. Quindi, il nostro Eps diventerà la K-dist, che corrisponde alla variazione repentina, mentre MinPts diventerà K;
5. I punti per i quali la k-dist è inferiore a Eps sono etichettati come Core point, mentre tutti gli altri punti verranno etichettati come Border point oppure come Noise point.

Il DBSCAN, quindi:

- ha il vantaggio di essere **molto più resistente al rumore** rispetto alle tecniche precedenti e allo stesso tempo, **ci permette anche di gestire dei cluster che hanno delle forme arbitrarie o delle forme arbitrarie** (quindi il DBSCAN è meno suscettibile al fatto di avere cluster di dimensioni diverse);
- il problema del DBSCAN, allora, non riguarda la dimensione dei cluster, bensì i**l problema riguarda la densità dei cluster**. Se abbiamo dei cluster più densi e dei cluster meno densi, il DBSCAN non funziona molto bene;
- inoltre, il DBSCAN ha **problemi nel gestire dati di grandi dimensioni**, in quanto la densità è più difficile da definire;
- inoltre, siccome ci richiede di calcolare la k-dist, abbiamo un **problema di complessità nell'eseguire tutti i calcoli di prossimità**.

Cluster Evaluation

Adesso che abbiamo visto tutti gli algoritmi di clusterizzazione, sorge spontanea la domanda: "**La clusterizzazione che abbiamo fatto, è buona oppure no?**" Nel caso della classificazione, era facile capire se avevamo fatto una buona classificazione o meno, in quanto avevamo le etichette reali e di conseguenza, andavamo a confrontare le etichette predette con le etichette reali e avevamo una serie di misure (come ad esempio: la precisione e l'accuratezza) che ci consentivano di verificare la bontà del nostro modello. Nel caso del clustering, invece, siamo in un caso **unsupervised** e nella maggior parte dei casi, non sappiamo cosa aspettarci nel momento in cui facciamo la clusterizzazione. Quindi, ognuno di noi, dato un cluster può vedere i raggruppamenti che vuole (nel senso, che i cluster sono negli occhi di chi guarda) e se applichiamo un qualsiasi algoritmo di clustering, quest'ultimo

fornisce sempre un risultato. I problemi della validazione della clusterizzazione sono:

- Abbiamo la necessità di definire delle tecniche che ci permettono di validare la clusterizzazione che abbiamo fatto ed in particolare, al fine di validare la clusterizzazione, dobbiamo essere innanzitutto in grado di determinare se esiste una **tendenza alla clusterizzazione dei dati**, ovverosia se esiste effettivamente una suddivisione dei dati o se quest'ultimi sono distribuiti in maniera uniforme;
- Dobbiamo determinare il numero corretto di cluster;
- Valutare quanto i risultati di un'analisi cluster si adattino ai dati senza fare riferimento a informazioni esterne;
- Confrontare i risultati di un'analisi cluster con risultati noti all'esterno (ad esempio, etichette fornite dall'esterno);
- Confronto tra due gruppi di cluster per determinare quale sia migliore.

Quindi, **le misure che ci permettono di determinare la validità di un cluster variano a seconda se ci troviamo in un caso:**

- **supervised →** in questo caso abbiamo a disposizione delle informazioni esterne per valutare la bontà di una clusterizzazione ed in questo caso, parliamo di **indici esterni**. In particolare, se abbiamo delle etichette da associare agli oggetti di uno stesso cluster, possiamo ritornare al concetto di entropia;
- **unsupervised →** in questo caso, dovremmo valutare la bontà di una clusterizzazione, senza aver bisogno di informazioni esterne. Queste misure prendono il nome di **indici interni** (dato che dipendono solamente dai dati che abbiamo a disposizione) e sono principalmente due misure:
  - **Cluster cohesion** → determina quanto sono vicini gli oggetti all'interno di uno stesso cluster;
  - **Cluster separation** → determina quanto siano diversi tra di loro i cluster.
- **oppure relative →** vanno a confrontare i diversi clustering che abbiamo eseguito (per esempio, se due clustering fatti con il K-means potrebbero essere confrontati utilizzando l'SSE o l'entropia).

A questo punto, quindi, analizziamo in maniera più approfondita questi tre concetti e partiamo, allora, dalla **unsupervised evaluation** → il nostro obiettivo è quello di determinare, senza avere altre informazioni esterne, se la nostra

clusterizzazione (la quale è formata da K cluster) complessivamente è buona oppure no. Questo si ottiene andando a svolgere una somma pesata della **validità** di ciascun cluster, dove la validità è un termine generale, che può indicare:

- coesione;
- separazione;
- oppure una combinazione tra coesione e separazione.

$$overall\ validity = \sum_{i=1}^{K} w_i\ validity(C_i)$$

💡 Quindi, per determinare la validità complessiva della nostra clusterizzazione, consideriamo come fattore fondamentale il livello di coesione, di separazione oppure un livello che è una combinazione tra le due.

"**Come sono definite la coesione e la separazione?**" La coesione è definita, per un certo cluster Ci, come la sommatoria delle prossimità dei suoi elementi; mentre nel caso della separazione, andiamo a confrontare la separazione tra un cluster Ci ed un cluster Cj e diciamo che la separazione è la sommatoria delle prossimità di due elementi (X ed Y) che provengono rispettivamente dal primo e dal secondo cluster. In particolare, le formule relative alla coesione e alla separazione sono le seguenti:

$$cohesion(C_i) = \sum_{\substack{x \in C_i \\ y \in C_i}} proximity(x, y)$$

$$separation(C_i, C_j) = \sum_{\substack{x \in C_i \\ y \in C_j}} proximity(x, y)$$

💡 Naturalmente, nel caso della coesione vogliamo **massimizzare** la prossimità, mentre nel caso della separazione vogliamo **minimizzare** la prossimità.

In realtà, possiamo definire la coesione non come la sommatoria delle prossimità di tutti i possibili punti del cluster Ci, bensì possiamo definire la coesione come la sommatoria delle prossimità tra un punto del cluster Ci e il

suo centroide → notiamo, allora, che questa definizione è molto simile a quella dell'SSE:

$$cohesion(C_i) = \sum_{x \in C_i} proximity(x, c_i)$$

e allora otteniamo, che la **coesione totale** è data dalla sommatoria totale (su tutti i cluster) della sommatoria (su tutti gli elementi di ciascun cluster) della distanza al quadrato tra X e il suo centroide:

$$total\ cohesion = SSE = \sum_i \sum_{x \in C_i} (x - c_i)^2$$

Invece, nel caso della separazione, possiamo rappresentare quest'ultima come la prossimità rispetto ai punti che appartengono a due cluster, ma in realtà possiamo rappresentare la separazione come la prossimità rispetto ad un centroide complessivo costruito su tutti i dati, ovverosia costruito su tutto il nostro Data set:

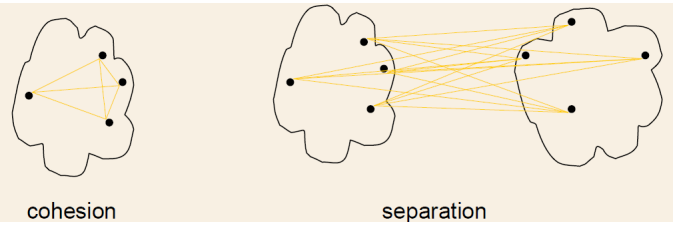
$$separation(C_i) = proximity(c_i, c)$$

In questo caso, la **separazione complessiva** (se utilizziamo l'SSE come funzione di prossimità) equivale alla sommatoria su tutti i cluster, della cardinalità del nostro cluster moltiplicata per la differenza al quadrato tra 'c' (ovvero il centroide complessivo) e il centroide del cluster singolo:

$$total\ separation = SSB = \sum_i |C_i|(c - c_i)^2$$

Se invece avessimo rappresentato il nostro cluster tramite una versione a grafo, allora avremmo ottenuto che:

- la **coesione** poteva essere considerata come la somma dei pesi delle connessioni all'interno delle stesso cluster → quindi, la coesione sarebbe gli archi che connettono i nodi presenti all'interno di uno stesso cluster;
- la **separazione** poteva essere considerata come la somma dei pesi tra i nodi che stanno all'interno del cluster e i nodi che stanno al di fuori del cluster.



L'idea è quella di riuscire a trovare una misura, che ci permetta di combinare la coesione e la separazione in un'unica misura. Questa misura prende il nome di **misura di Silhouette** e ci permette innanzitutto di confrontare in maniera rapida due cluster ed in particolare, rappresenta quanto bene gli oggetti 'cadono' all'interno del proprio cluster. La Silhouette è definita a livello di singolo punto (quindi ci dice quanto sta bene il punto nel cluster che gli è stato assegnato), ma partendo dalla Silhouette del singolo punto siamo in grado di costruire la Silhouette per un cluster e successivamente anche la Silhouette per l'intera clusterizzazione. Sorge a questo punto spontanea la domanda: "**Come si calcola la Silhouette?**" Preso un oggetto di distanza 'o', andiamo a calcolare la sua distanza media da tutti gli altri punti che si trovano nello stesso cluster (e tale misura la identifico come 'a(o)'). A questo punto, per tutti gli altri cluster diversi da quello in cui si trova il punto 'o', calcoliamo la distanza media tra il punto 'o' e gli oggetti negli altri cluster (quindi avremo N misure medie, dove N è il numero di cluster - 1) e poi trovo il valore minimo (il quale lo identifico come 'b(o)'). A questo punto la Silhouette è data da:

$$\text{Silhouette(o)} = \frac{b(o)-a(o)}{\max(a(o),b(o))}$$

💡 **La Silhouette, quindi, sarà un numero compreso tra -1 e +1**. Un valore negativo della Silhouette si ottiene quando a(o) > b(o) e sono valori che cerchiamo di non ottenere, in quanto significherebbe che la distanza media del nostro punto 'o', rispetto ai punti dello stesso cluster, è maggiore rispetto alla distanza media con i punti degli altri cluster. Invece, il valore massimo (ovvero 1) si ottiene quando a(o) = 0.

A questo punto, abbiamo ottenuto la Silhouette per tutti i singoli punti e di conseguenza, possiamo calcolare la Silhouette per un intero cluster, andando a calcolare la media delle Silhouette dei punti, che stanno all'interno dello stesso cluster. Successivamente, possiamo calcolare anche la Silhouette per un'intera

clusterizzazione, andando a fare la media delle Silhouette di tutti i punti del nostro Data set.

> 💡 Quindi, **la Silhouette è la tipica tecnica che viene utilizzata per confrontare le clusterizzazioni che abbiamo ottenuto**.

Infine, nel caso in cui siamo in una **supervised evaluation**, dove a fronte di una clusterizzazione abbiamo accesso ad informazioni esterne che ci danno (ad esempio) delle etichette per le nostre istanze, dobbiamo capire qual è il grado di corrispondenza tra le etichette dei cluster e le etichette delle classi. Nel caso della supervised evaluation, abbiamo tipicamente due approcci:

- approccio **Classification oriented** → in cui andiamo ad utilizzare le misure che abbiamo già visto per la classificazione, ovverosia: la **purità**, **l'entropia**, la **precision** e la **recall** ed infine valutiamo in che misura un cluster contiene oggetti della stessa classe;
- approccio **Similarity oriented** → in cui utilizziamo delle tecniche basate sulla similarità, che misurano sostanzialmente quanto due oggetti nella stessa classe sono anche nello stesso cluster e viceversa, andando a confrontare le etichette della classe con le etichette del cluster.