Roberto Esposito
21 December 2020

Conoscenza incerta e ragionamento probabilistico

Gli agenti non hanno mai la completa conoscenza dell'ambiente e quindi devono agire in condizioni di incertezza.

Se si utilizza la logica di primo ordine per gestire un dominio si fallisce per tre motivi principali:

1. **Pigrizia**: elencare l'insieme completo di azioni permesse che rendono una regola vera può essere faticoso perché ne sono veramente tante;
2. **Ignoranza teorica**: non si hanno tutte le conoscenze in quell'ambito perché non si ha la conoscenza completa del dominio di interesse;
3. **Ignoranza pratica**: anche se si conoscono tutte le regole può essere che si presenta una mancanza di dati;

La conoscenza dell'agente si può limitare ad un grado di credenza nelle formule, vi è la possibilità che queste formule siano vere.

Attraverso la teoria della probabilità si può assegnare un valore compreso tra 0 e 1 che esprime un **grado di credenza** di una espressione.

Di base una formula è sempre vera oppure falsa, ma questo valore a priori è ignoto e a tal punto affido a questa formula un grado di credenza che sia vera. Il grado di credenza indica l'aspettativa che la formula possa essere vera.

Le credenze in un evento dipendono dalle percezioni ricevute dall'agente. Le sue percezioni sono le prove su cui si basano le asserzioni sulle probabilità.

Una probabilità nota in assenza di prove si dice a **priori (o non condizionata)** mentre dopo avere delle prove si parla di **probabilità a posteriori (o condizionata)** quest'ultima dipende da eventi noti.

I gradi di credenza sono sempre applicati alle proposizioni che possono essere visti come enunciati che affermano il verificarsi di qualcosa.

L'elemento base del linguaggio è la **variabile casuale**.

Ogni variabile casuale ha un dominio di valori che essa può assumere ovvero lo spazio campionario.

Le variabili possono essere di 3 categorie:

1. **Variabili booleane**: il cui dominio è {true, false};
2. **Variabili discrete**: includono le booleane e hanno un dominio enumerabile;
3. **Variabili continue**: non si possono enumerare perché il dominio è infinito e solitamente si utilizza il sottoinsieme [0,1] (che è ugualmente infinito);

Un **evento atomico** è un elemento fondamentale della teoria della probabilità.

Un evento atomico è una specifica completa dello stato del mondo di cui l'agente è incerto e consiste nell'assegnare a tutte le variabili casuali un valore del loro dominio (combinazioni degli eventi).

Gli eventi atomici devono avere 4 proprietà:

1. **Mutuamente esclusivi**: in ogni dato momento si può verificare solo un evento atomico e non si possono verificare insieme;
2. **Esaustività**: l'insieme di tutti i possibili eventi atomici è tale per cui uno di essi sarà sempre vero;
3. **Verificabilità**: l'insieme degli eventi atomici permette di derivare la verità o falsità di qualsiasi proposizione, se fossero noti tutti gli eventi atomici allora si può esprimere qualsiasi proposizione;
4. Ogni proposizione è logicamente equivalente alla **disgiunzione** di tutti gli eventi atomici che implicano la verità della proposizione stessa;

Definiamo la probabilità a priori P(a) di una proposizione a p il grado di credenza che le viene associato in assenza di altre informazioni.

Se una variabile aleatoria può avere un dominio di n valori distinti allora avremo n valori di probabilità a priori che possono essere incluse in un **vettore**. Ad esempio

P(A=1)=0.1;

P(A=2)=0.2; —> **P**(A) = <0.1, 0.2, 0.7>

P(A=3)=0.7;

L'insieme dei valori presenti all'interno del vettore è 1.

Questo vettore definisce la **distribuzione di probabilità a priori** e nel caso in cui la variabile anziché essere discreta è continua allora si chiamata **funzione di densità di probabilità**.

La distribuzione di **probabilità congiunta** indica la congiunzione di due o più variabili casuali e consiste nella probabilità di tutte le combinazioni possibili di valori assunti dalle variabili casuali. E nel caso di variabili discrete allora si rappresenta sottoforma di matrice.

Una distribuzione di probabilità congiunte si dice **completa** quando comprende tutte le variabili casuali che regolano l'ambiente considerato.

Le variabili continue ovviamente non possono essere rappresentate come forma matriciale, ma come **curva di funzione**.

La **probabilità a posteriori** (o condizionata) è il grado di credenza che si ha basandosi sulla conoscenza del mondo, si basa su una conoscenza pregressa. La probabilità di P(a|b) è la probabilità che si verifica a sapendo che si è verificato b.

La probabilità a posteriori si può calcolare da quella a priori come: $P(a|b) = \frac{P(a \wedge b)}{P(b)}$

Questa formula è vera ogni volta che P(b) > 0. Dalla formula della probabilità a posteriori si può ricavare la **regola del prodotto**: $P(a|b) = \frac{P(a \wedge b)}{P(b)} \iff P(a|b)P(b) = P(a \wedge b)$

Ovviamente **non** bisogna fraintendere P(a|b) come ogni volta che si verifica P(b) allora P(a) = P(a|b) perché a e b possono essere indipendenti o meno fra di loro.

Gli assiomi di base della teoria della probabilità sono gli assiomi base che definiscono la scala della probabilità e i loro estremi prendono il nome di **assiomi di Kolmogorov**:

1. Tutte le probabilità sono comprese tra 0 e 1;
2. Le proposizione vere hanno sempre valore 1 e quelle false hanno valore 0. P(true)=1 e P(false)=0;
3. La probabilità di una disgiunzione è data da: $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$;

Dagli assiomi di Kolmogorov si possono estrarre delle nuove regole avendo così le inferenze sugli assiomi, come la **regola della negazione**: $P(a) = 1 - P(\neg a)$ basta sostituire alla 3 di Kolmogorov $\neg a$ al posto di b e da lì il resto è banale.

In generale una certa variabile casuale discreta D di dominio <d1,d2,…,dn> ha come somma dei valori delle probabilità 1.

L'**inferenza probabilistica** è il calcolo delle probabilità a posteriori partendo dalle prove osservate; studiamo il problema e abbiamo necessariamente una conoscenza pregressa (probabilità a priori), basandoci su questa, possiamo determinare la base di conoscenza del nostro problema.

La **base di conoscenza** del problema è rappresentata dalla distribuzione di probabilità congiunta completa da cui è possibile derivare le risposte a tutte le domande.

La **marginalizzazione** consiste nell'estrazione della distribuzione di probabilità di un sottoinsieme di variabili, o anche una sola.

La somma delle probabilità di una riga corrisponde a calcolare la **probabilità marginale**.

Detto in modo più formale: $P(Y) = \sum P(Y \wedge z)$.

Utilizzando la regola del prodotto si ha $P(Y) = \sum P(Y|z)P(z)$ e questa prende il nome di **regola di condizionamento**.

Due eventi si definiscono **indipendenti** se il verificarsi di uno non ha alcuna implicazione sul verificarsi o meno dell'altro e quindi non viene influenzato.

In altri termini due eventi sono indipendenti se $P(a \wedge b) = P(a)P(b)$. Grazie all'indipendenza si può semplificare la rappresentazione di un problema.

La **regola di Bayes** si ricava dalla regola del prodotto e sfruttandone la sua commutatività ovvero: $P(a \wedge b) = P(b \mid a)P(a) = P(a \mid b)P(b)$ se si divide per P(a) secondo e terzo membro si ottiene: $P(b \mid a) = \frac{P(a \mid b)P(b)}{P(a)}$.

P(a | b) —> verosimiglianza

P(b | a) —> probabilità a posteriori

P(a) —> evidenza

P(b) —> probabilità a priori

Ciò è anche conosciuto come **teorema delle cause** in quanto se un evento A dipende da una causa B, grazie al teorema di Bayes si può calcolare la probabilità della causa B sapendo che l'evento A si è verificato.

Regola di Bayes normalizzata: $P(b \mid a) = \alpha P(a \mid b)P(b)$ questo fattore alpha serve a garantire la proprietà che la probabilità è limitata inf. 0 e sup. 1.

Alpha è il **fattore di normalizzazione** affinché la probabilità congiunta rispetti l'assioma 1, ovvero tutto il vettore di probabilità somma a 1.

Reti Bayesiane

Si definisce **indipendenza condizionale** la relazione di indipendenza tra due variabili in presenza di altre prove. Espresso in modo più formale date due variabili X e Y esse sono condizionalmente indipendenti se per una terza variabile Z è verificato: $P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z)$. (Ci permette di "spezzare" le due variabili).

Se si utilizza l'indipendenza condizionale si può semplificare notevolmente la dimensione delle tabelle rappresentanti la probabilità congiunta in quanto si vengono a creare più tabelle di dimensioni molto ridotte.

In generale una singola causa influenza direttamente più effetti, tutti condizionalmente indipendenti data la causa e quindi esprimendolo in modo più formale si ottiene: $P(Causa, Effetto1,...,Effetton) = P(Causa)\prod P(Effettoi \mid Causa)$.

Tale modello prende il nome di modello di Bayes ingenuo anche noto come **classificatore bayesiano**. Viene definito ingenuo perché lo si utilizza anche quando gli effetti non sono condizionalmente indipendenti data la causa.

Se si hanno informazioni sulla probabilità congiunta totale si può inferire qualsiasi informazione nascosta nella rappresentazione del problema, ma ciò non sempre lo si ha e nel

caso in cui si ha questa conoscenza totale risulta essere di dimensioni elevate. A tal proposito si ricorre all'utilizzo della rete bayesiana.

Una **rete bayesiana** è un modello probabilistico di tipo grafico che fonda tutto il suo principio sulla regola di Bayes.

Una rete bayesiana è un grafo orientato in cui ogni nodo è etichettato con informazione probabilistica quantitativa.

Le **caratteristiche principali** di una rete bayesiana sono:

1. I nodi della rete sono costituiti da un insieme di variabili casuali che possono essere discrete o continue
2. Un insieme di archi collega coppie di nodi. Se c'è un arco da X verso Y allora X è genitore di Y.
3. Ogni nodo Xi ha una distribuzione di probabilità condizionata del tipo P(Xi|Genitori(Xi)) e quantifica l'effetto dei genitori sul nodo.
4. Il grafo è un DAG (directed acyclic graph).

(Se il grafo non è direzionato allora diventa una rete markoviana)

Quindi l'assenza di percorsi fra due nodi significa che non sono in nessuna maniera collegati e quindi sono indipendenti.

Inoltre due nodi se non sono collegati fra di loro e sono fratelli (quindi hanno lo stesso genitore) allora potremo dire che sono condizionalmente indipendenti dato il genitore.

Bisogna fare attenzione però in quanto dato un nodo G si potrebbe essere indotti a dire che esso è condizionalmente indipendente dagli altri nodi dati i nodi padre, ma questo è falso perché la probabilità del nodo G è condizionata anche dai discendenti. (Se si è verificato G sapendo che i suoi genitori si sono verificati e si è verificato anche un figlio allora questa probabilità sarà sicuramente diversa da quella nel caso in cui non sappiamo se un figlio si è verificato o meno).

Attenzione il condizionamento di qualsiasi altro antenato di G non influenza la probabilità del nodo in quanto già l'abbiamo considerata in modo indiretto.

Ogni nodo ha una distribuzione di probabilità condizionata che viene espressa sotto forma di tabella che prende il nome di **CPT**. Ogni riga di questa tabella indica la probabilità condizionata di tutti i valori del nodo per un singolo caso condizionante ovvero la combinazione di valori dei genitori.

Se un nodo ha k genitori allora la CPT di quel nodo avrà al più 2^k probabilità indipendenti.

Nel caso in cui invece quel nodo non ha genitori allora avrà una sola riga la sua CPT e corrisponderà alla probabilità a priori.

Se la rete bayesiana fosse costituita da variabili aleatorie continue allora non avremo la distribuzione di probabilità, ma la **funzione di densità di probabilità**.

Ci sono due modi per comprendere la semantica di una rete bayesiana:

1. Tipo numerico: la rete è una rappresentazione di una distribuzione congiunta di probabilità;
2. Tipo topologico: la rete è una codifica di una collezione di asserzioni di indipendenza condizionale;

La rete permette di avere una descrizione completa del dominio di interesse.

Ogni elemento della distribuzione di probabilità congiunta può essere calcolato dall'informazione contenuta nella rete.

Ad esempio sia X1=x1, X2=x2, X3=x3, il valore di probabilità si ottiene dalla topologia della rete come: $P(x1,x2,...,xn) = \prod P(xi|Genitori(Xi))$. Ogni elemento della distribuzione congiunta è rappresentato dal prodotto degli elementi appropriati nelle CPT della rete bayesiana.

La **regola della catena** è una generalizzazione della regola del prodotto in quanto avendo $P(x1,x2,...,xn) = P(xn|xn-1,...,x1)P(x1,...,xn-1)$ se si applica questo ragionamento n volte possiamo arrivare ad ottenere $P(x1,x2,...,xn) = \prod P(xi|xi-1,...,x1)$.

C'è una stretta equivalenza fra ciò e la definizione data di rete bayesiana in quanto abbiamo definito P(Xi|Genitori(Xi)), ma questo è uguale a P(Xi|Xi-1,…,X1). Per soddisfare tale uguaglianza si devono rispettare i criteri di costruzione di una rete bayesiana ovvero partire dalle cause con probabilità a priori per poi sviluppare "dall'alto verso il basso" la rete, in pratica si deve seguire un ordinamento.

La rete bayesiana utilizza quindi la regola della catena.

Si definisce **sistema localmente strutturato** ogni sotto-componente che interagisce direttamente solo con un numero ristretto di altri componenti, indipendentemente dal numero totale di essi.

La compattezza delle reti bayesiane è un esempio di sistemi localmente strutturati o anche noti come **sistemi sparsi**.

Per capire meglio questo concetto basti considerare il fatto che ogni nodo avrà al più k nodi genitore e la propria tabella avrà al più 2^k elementi, se la rete ha n nodi si avranno quindi al più $n2^k$ elementi. Contrariamente alla probabilità congiunta totale che avrà 2^n elementi, con

k < n. Ad esempio sia n = 30 e k = 5, la rete bayesiana avrà 960 elementi, la distribuzione congiunta totale più di un miliardo di elementi. In altri termini, connessioni "deboli" o nulle vanno trascurate perché danno informazioni maggiori, ma ne aumenta notevolmente la complessità.

Fino ad ora abbiamo rappresentato la semantica di una rete bayesiana in termini numerici attraverso la rappresentazione della distribuzione congiunta.

Ma si può anche individuare una **semantica di tipo topologico** che descrive le dipendenze codificate nella struttura albero e poi da lì derivarne la semantica numerica. A tal proposito si può definire la semantica da un punto di vista topologico di una rete come:

1. Un nodo è condizionalmente indipendente da tutti i suoi non-discendenti, dati i suoi genitori;
2. Un nodo è condizionalmente indipendente da tutti gli altri nodi nella rete, dati i suoi genitori, i suoi figli e i genitori dei suoi figli, prende il nome coperta di Markov);

Nonostante si cerchi di ottimizzare la complessità della rete, abbiamo visto che nel caso migliore ciascun nodo richiederà come complessità $O(2^k)$ elementi per la sua CPT ed è un valore elevato.

Si può cercare di migliorarne la complessità utilizzando le **distribuzioni canoniche** le quali si utilizzano per rappresentare le CPT di un nodo, sono distribuzioni noti a cui è possibile dimostrare l'aderenza del nostro problema ad esse e quindi un determinato nodo del sistema segue una distribuzione canonica la quale è codificata e quindi non è necessario calcolarla. Questa tecnica la si può utilizzare anche se risulta essere un'approssimazione del problema e quindi non è sempre esatta al 100%.

Un'altra tecnica, osservando i nodi e le relazioni con i propri genitori, è quella di verificare se un determinato **nodo** è **deterministico** oppure no. Un nodo si definisce deterministico se il suo valore di probabilità dipende esattamente da quello dei suoi genitori.

Non sempre è possibile individuare nodi deterministici, ma le relazioni di incertezza possono essere caratterizzate da relazioni logiche definite "rumorose".

Definiamo **OR rumoroso** (Noisy-OR) l'interazione tra n cause Xi = {x1,x2,…,xn} e il loro effetto comune y. Si assume che le cause siano ciascuna sufficiente per determinare y indipendentemente dalle altre. Quindi ogni variabile xi potrà causare y con una certa probabilità pi. In termini formali lo si può esprimere come: $pi = P(y|\bar{x1}, \bar{x2}, \ldots, xi, \ldots, \bar{xn})$.

Conseguentemente la probabilità che y sia causata da un sottoinsieme di cause Xp degli stati Xi che sono vere è pari a: $P(y|Xp) = 1 - \prod_{Xi \in Xp} (1 - pi)$. Per applicare questa tecnica dovremmo elencare tutte le possibili cause che hanno causato l'evento y, ma in alcuni casi non è possibile elencarle tutte e quindi si ricorre ad un **nodo pozzo**. Dopo un certo numero di cause, quelle non considerate finiscono nel nodo pozzo ovviamente ve ne faranno parte le cause che non danno un grande contributo.

In generale le relazioni logiche rumorose in cui una variabile dipende da k genitori sarà O(k) anziché $O(2^k)$.

Reti Bayesiane con Variabili Continue

La maggior parte dei problemi probabilistici è costituito da variabili continue. Una variabile continua ha un numero infinito di valori, quindi una CPT sarebbe impossibile da rappresentarla.

Per risolvere tale problema si può utilizzare la **discretizzazione** non efficiente in quanto non può essere particolarmente accurata.

Un'altra soluzione comune è quella di fare ricorso a delle funzioni di densità di probabilità standard il cui andamento dipende da un certo numero di **parametri**. Solitamente una funzione di densità che si utilizza è la distribuzione gaussiana (o normale) la quale è definita da due parametri:

1. Il valore medio (mu), dove si raggiunge il picco della gaussiana;
2. La varianza (sigma quadro), che determina l'ampiezza della funzione;

Reti Bayesiane Ibride

Una situazione che si può verificare spesso è la presenza all'interno della stessa rete bayesiana sia variabili continue che variabili discrete. In tal caso bisognerà stabilire due cose:

1. Nodo discreto con genitori continui;
2. Nodo continuo con genitori discreti;

CASO 1: figlio continuo

Se un figlio è una variabile continua ed ha genitori in parte discreti ed in parte continui dovremo considerare le possibili probabilità condizionate dei casi discreti. Ad esempio sia X discreta e Y continua ed hanno un figlio Z continuo. A tal punto dovremo calcolare

P(Z|X, Y) e P(Z|notX,Y), la scelta più comune è quella di una gaussiana lineare.

Inoltre le operazioni fra le gaussiane come ad esempio la somma dà origine ancora ad una gaussiana che prende il nome di gaussiana multivariata.

CASO 2: figlio discreto con genitore continuo

In questo caso si utilizza una funzione soglia morbida oltre la quale ad esempio il figlio avrà un certo valore e sotto il quale ne avrà un altro.

Una forma comune di funzione soglia è data dall'integrale della distribuzione normale standard, di media 0 e varianza 1: $\phi(x) = \int_{-\infty}^{x} N(0,1)(x)dx.$

Per calcolare la probabilità condizionata del nodo figlio discreto dato il genitore continuo allora si possono utilizzare due distribuzioni: la **distribuzione probit** e la **distribuzione logit**. Queste due distribuzioni permettono di definire una funzione soglia morbida che indica il punto di transizione. In modo particolare inoltre la logit è un elemento cardine nelle reti neurali (percettrone).

Costruzione della Rete Bayesiana

Una rete bayesiana ovviamente deve essere sviluppata da un esperto che conosce il problema e sa come affrontarlo, ma è una rete del tutto soggettiva quindi vi possono essere vari punti di vista del problema.

La procedura oggettiva che si può utilizzare per la costruzione di una rete bayesiana (non significa che è meglio di quella soggettiva) è quella di partire con l'analisi dei dati in nostro possesso. Questa procedura prevede l'utilizzo dell'**algoritmo Spanning Tree** che prevede di partire da una collezione di nodi e aggiungere archi finché tutti i nodi sono raggiunti da una connessione. I nodi connessi da un arco saranno dipendenti mentre quelli non connessi da un arco saranno al più condizionalmente indipendenti.

La strategia è quella di aggiungere archi tra i nodi che sono maggiormente dipendenti, a tal proposito è necessario definire una giusta misura di dipendenza.

La **dipendenza (metrica L1)** fra due variabili A e B viene espressa nel seguente modo: $Dep(A,B) = |P(A \wedge B) - P(A)P(B)| = \sum_{AxB} |P(ai \wedge bj) - P(ai)P(bj)|$

I valori $P(ai \wedge bj)$ vengono presi dalla **matrice delle co-occorrenze** (una matrice che indica quante volte due elementi occorrono in maniera congiunta) questo valore viene poi diviso per il numero totale delle osservazioni.

Se Dep(A,B) ha un valore molto alto allora metteremo un arco tra i nodi A e B. Si può notare che se Dep(A, B)=0 allora le variabili sono indipendenti.

Algoritmo:

1. Per ogni coppia di variabili A e B si calcola Dep(A,B)
2. Si collegano le coppie di nodi in ordine decrescente di dipendenza finché tutti i nodi sono raggiunti e non ci sono cicli

Un'altra metrica che si utilizza in questo caso è anche la **metrica L1 pesata** che ha il seguente valore: $Dep(A,B) = \sum_{AxB} (P(ai \wedge bj)x\,|P(ai \wedge bj) - P(ai)P(bj)|$

Da quest'ultima si può ricavare anche la **metrica L2** e la sua versione **pesata,** ovvero $L2 = Dep(A,B) = \sum_{AxB} (P(ai \wedge bj) - P(ai)P(bj))^2$ la versione pesata è praticamente identica ma viene moltiplicata semplicemente per $P(ai \wedge bj)$.

Un'altra metrica di ampio utilizzo è la **mutua informazione** nota anche con il nome di **co-entropia** oppure **divergenza di Kullback-Leibler**:

Mutual information = $Dep(A,B) = \sum_{AxB} P(ai \wedge bj) log_2(\frac{P(ai \wedge bj)}{P(ai)P(bj)})$

La mutua informazione ha 3 particolari proprietà:

1. Vale zero quando le variabili sono totalmente indipendenti
2. È positiva e cresce al crescere della dipendenza tra variabili
3. Non dipende dai singoli valori di probabilità

La **correlazione** è un'altra metrica che volendo si può utilizzare per individuare il grado di dipendenza fra due variabili.

-**Parameter learning**: dato un dataset e un dag che cattura le dipendenze tra le variabili l'obiettivo è stimare la distribuzione di probabilità condizionata delle singole variabili;

-**Structure learning**: dato un dataset, l'obiettivo è stimare un dag che cattura le dipendenze tra le variabili;

Modelli Markoviani

Nel caso in cui si considerano tecniche di ragionamento probabilistico in un contesto di **modelli dinamici** bisogna tenere conto che i valori osservati possono variare nel tempo e quindi una decisione può dipendere dal momento in cui i dati sono osservati.

Per ragionare probabilisticamente in presenza del fattore tempo bisogna modellare il cambiamento del mondo attraverso una **time slice**.

Definiamo:

-X_t per indicare le variabili ignote al tempo t;

-E_t per indicare le variabili di prova osservabili al tempo t, osservazione;

I modelli markoviani sono di solito identificarti da ciò:

Definiamo **processo stazionario** quando i cambiamenti nel mondo sono regolati da leggi che non mutano nel tempo, le leggi che valgono al tempo t0 varranno anche a t1, t2,…,tn.

Definiamo **ipotesi di Markov** il fatto che lo stato corrente dipende solo da una storia finita di stati precedenti.

Quando l'ipotesi di Markov è legata solo all'ipotesi dello stato immediatamente precedente allora si parla di processo **markoviano del primo ordine**, più in generale si parla di **catena di Markov (o processi di Markov)**.

Per costruire un modello markoviano bisogna costruire due modelli:

- **modello di transizione**: lo stato corrente dipende solo dallo stato precedente e da nessun altro e quindi $P(X_t | X_{0:t-1)} = P(X_t | X_{t-1})$.

- **modello sensoriale**: lo stato corrente è determinato solo dagli eventi correnti e quindi $P(E_t | X_{0:t-1}, E_{0:t-1}) = P(E_t | X_t)$.

Ovviamente devo conoscere anche le **probabilità a priori** degli stati al tempo iniziale, se non li conosco allora ne ipotizziamo i valori.

È possibile fare 4 azioni sui modelli temporali, si possono inferire:

1. **Filtraggio o monitoring**: consiste nel calcolare lo stato-credenza, ovvero la distribuzione a posteriori date tutte le prove osservate $P(X_t | e_{1:t})$
2. **Predizione**: consiste nel calcolare la distribuzione a posteriori dello stato futuro date tutte le prove raccolte $P(X_{t+k} | e_{1:t})$ con 0 <= k < t;
3. **Smoothing o regolarizzazione**: calcolo della distribuzione a posteriori di uno stato passato date tutte le prove raccolte $P(X_k | e_{1:t})$ con 0 <= k < t;
4. **Spiegazione più probabile**: data una sequenza di osservazioni potremmo desiderare di trovare una sequenza di stati che più probabilmente ha generato tali osservazioni $argmax_{x_{1:t}} P(x_{1:t} | e_{1:t})$;

-**Filtraggio**: operazione che consiste nel calcolare lo stato-credenza di una variabile non nota conoscendo tutte le prove osservate dallo stato iniziale fino allo stato t.

Tale tecnica viene utilizzata grazie alla presenza di una funzione f che prende il nome di **stima ricorsiva** perché permette di esprimere tale valore in modo ricorsivo: $P(X_{t+1} | e_{1:t+1}) = f(e_{1:t+1}, P(X_t | e_{1:t}))$.

Il filtraggio è costituito da due fasi:

1. Si proietta in avanti la distribuzione di probabilità di stato da t a t+1;
2. Si aggiorna in base all'osservazione corrente e_{t+1};

$$P(X_{t+1} | e_{t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1}) = \alpha P(e_{t+1} | X_{t+1}, e_{1:t}) P(X_{t+1} | e_{1:t}) = \alpha P(e_{t+1} | X_{t+1}) P(X_{t+1} | e_{1:t})$$

La seconda uguaglianza applica semplicemente la regola di Bayes, quella successiva utilizza la proprietà di Markov del modello sensoriale.

Nel risultato finale si ha quindi il primo termine che si può calcolare dal modello sensoriale ed il secondo termine che costituisce la predizione.

-La **predizione** la si può vedere come un filtraggio senza conoscere le osservazioni del tempo t+1.

La quantità $P(X_t | e_{1:t})$ corrisponde al messaggio della funzione $f_{1:t}$ il quale viene propagato in avanti , modificato ad ogni transizione ed aggiornato ad ogni nuova osservazione, $f_{1:t+1} = \alpha FORWARD(f_{1:t}, e_{t+1})$.

-**Smoothing**: è il processo di calcolo della distribuzione di stati passati date le prove osservate fino al presente cioè al tempo t e permette di correggere tali distribuzioni al tempo k, con k<t.

$$P(X_k | e_{1:t}) = P(X_k | e_{1:k}, e_{k+1:t}) = \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k, e_{1:k}) = \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k) = \alpha f_{1:k} b_{k+1:t}$$

La variabile $b_{k+1:t}$ rappresenta il messaggio all'indietro e corrisponde al backward infatti si ha $b_{k+1:t} = BACKWARD(b_{k+2:t}, e_{k+1:t})$.

L'algoritmo di **FORWARD-BACKWARD** è un algoritmo inferenziale per modelli di Markov che calcola la probabilità a posteriori marginale di tutte le variabili di stato data una successione di osservazioni $e_{1:t} = e_1, \ldots, e_t$ cioè esso calcola per tutte le variabili di stato nascoste $X_k \in X_1, \ldots, X_t$, la distribuzione $P(X_k | e_{1:t})$, messaggi avanti, indietro, osservazioni e probabilità corrette\.

I **modelli di Markov nascosti** (Hidden Markov Model) sono molti utili per poter rappresentare un modello markoviano sottoforma di matrice. Ogni stato del processo markoviano è descritto da una singola variabile casuale discreta. È possibile così rappresentare il modello di transizione, sensoriale e dei messaggi in avanti e all'indietro.

Supponiamo che una variabile X_t possa assumere uno dei seguenti stati {1,2,...,S}.

Il modello di transizione diventa una matrice T_{SxS} tale che: $T_{ij} = P(X_t = j | X_{t-1} = i)$, ovvero la probabilità di una transizione dallo stato i al tempo t-1 allo stato j al tempo t.

Il modello sensoriale diventa una matrice diagonale O_t i cui valori sulla diagonale sono $P(e_t | X_t = i)$.

I messaggi in avanti e all'indietro si ottengono grazie a semplici operazioni tra matrici.

La probabilità in avanti $f_{1:t+1} = \alpha O_{t+1} T^T f_{1:t}$.

La probabilità all'indietro $b_{k+1:t} = \alpha O_{k+1} b_{k+2:t}$.

Apprendimento

Quando si parla di apprendimento bisogna fare una distinzione fra classificazione e regressione.

-**Regressione**: si costruisce un modello continuo per predire il valore di risposta di un predittore in funzione dei dati

-**Classificazione**: dà una risposta discreta sui dati effettuati indipendentemente dal fatto che i dati siano continui o discreti

La risposta di un regressore è continua mentre quella di un classificatore è discreta, il regressore stima l'andamento di un certo fenomeno indipendentemente da quanto esso complesso sia, la classificazione permette di attribuire ad ogni campione del modello un'etichetta che esso rappresenta.

Sia la regressione che la classificazione hanno un errore, l'errore della regressione è, data un retta di regressione quanto i punti siano distanti da tale retta, l'errore della classificazione è dato da, tracciata una retta di separazione, se ci sono dei campioni di un etichetta A che appartengono all'etichetta B allora questo è un errore.

Infine abbiamo il **clustering** il quale è diverso dalla classificazione poiché nella classificazione si conosce a priori ogni singolo campione a quale etichetta appartiene, nel clustering si raggruppano campioni con caratteristiche simili, ma non si conosce la loro etichetta.

L'**overfitting** è il sovraddatamento di un modello ai dati, il modello si adatta in modo particolare ai dati che gli vengono forniti.

Si può parlare di apprendimento sotto due diversi punti di vista:

-**Apprendimento induttivo**: apprendimento che nasce dalle osservazioni degli eventi nel fenomeno in esame, (alberi di decisione);

-**Apprendimento statistico**: apprendimento basato sulla formulazione di ipotesi probabilistiche che governano il fenomeno in esame, (apprendimento bayesiano);

Le forme più comuni di apprendimento sono:

1. **Apprendimento parametrico**: si suppone di conoscere le distribuzioni che regolano il fenomeno studiato, di tali distribuzioni se ne individuano i parametri, ad esempio notiamo che i nostri dati hanno un andamento gaussiano, ma non ne conosciamo né la media né la varianza;
2. **Apprendimento supervisionato**: apprendere una funzione che descrive il fenomeno studiato partendo da input e output noti. Si conoscono i dati e si conosce precisamente cosa essi rappresentano, praticamente è un problema di classificazione;
3. **Apprendimento non supervisionato**: si impara a riconoscere i pattern o i schemi presenti nell'input senza conoscerne i valori di uscita, problema di clustering;
4. **Apprendimento con rinforzo**: è la forma più generale di apprendimento. L'idea che sta alla base è quella di imparare ad apprendere basandosi su un giudizio delle sue decisioni. Le decisioni che portano ad un successo portano ad una ricompensa e quelle che risultano errate comportano una penalizzazione, in questo caso i dati di partenza possono anche essere molto pochi;

Un algoritmo di apprendimento deve disporre di due cose in particolare:

-**Rappresentazione dell'informazione appresa**: devono essere in quantità, in forma e in distribuzione quanto più eterogenea, poi questi possono essere rappresentati in diversi modi (grafo, piano cartesiano, etc);

-**Conoscenza pregressa**: è necessario dare delle informazioni pregresse per poter apprendere, ad esempio in passato si pensava che un processo di apprendimento dovesse partire da nessuna conoscenza pregressa, ma in realtà non è così in quanto nell'apprendimento umano anche un neonato nasce con un baglio di conoscenze apprese durante la vita in utero;

Apprendimento Induttivo

Un algoritmo di apprendimento induttivo dove, data una coppia (x, f(x)) in cui x è l'input e f(x) è l'output della funzione applicata ad x, deve essere in grado di costruire una **funzione di approssimazione** h di una serie di campioni provenienti da una funzione f. Ovviamente la funzione di approssimazione h deve essere una buona generalizzazione di f, ossia predire correttamente esempi che non ha ancora incontrato.

Il rasoio di Occam sostiene che si deve sempre preferire l'ipotesi più semplice consistente con i dati. Una ipotesi (la funzione di approssimazione) non dovrebbe mai essere più complessa dei dati stessi.

Un problema di apprendimento è **realizzabile** se lo spazio delle ipotesi contiene la funzione reale da apprendere, altrimenti è **irrealizzabile**. Purtroppo non è possibile determinare quanto un problema di apprendimento possa essere realizzabile o meno perché la funzione originale non è nota, ma in nostro aiuto viene la conoscenza pregressa che permette di fare delle ipotesi sui meccanismi che regolano il fenomeno in esame.

Apprendimento Statistico

L'apprendimento si basa sui dati e sulle ipotesi. I **dati** costituiscono le prove, ovvero le istanze di alcune o tutte le variabili casuali che descrivono il dominio. Le **ipotesi** sono le teorie probabilistiche sul funzionamento del dominio.

L'**apprendimento Bayesiano** si basa sulla regola di Bayes e calcola la probabilità di ogni ipotesi condizionandola ai dati osservati. Le predizioni sono basate su tutte le ipotesi.

Indichiamo con D tutto il set di dati e con d le singole osservazioni, la probabilità di una ipotesi si ottiene dalla regola di Bayes: $P(h_i \mid d) = \alpha P(d \mid h_i)P(h_i)$.

Per calcolare la probabilità non nota di una proprietà X di una certa osservazione d è necessario fare: $P(X \mid d) = \sum P(X \mid d, h_i)P(h_i \mid d) = \sum P(X \mid h_i)P(h_i \mid d)$, la d la si toglie in quanto ormai ha dato il suo contributo.

Ipotizziamo che si abbiano una serie di ipotesi tutte indipendenti ed identicamente distribuite si indica la **verosimiglianza** come $P(d \mid h_i) = \prod_j P(d_j \mid h_i)$.

Nell'apprendimento bayesiano l'ipotesi vera prima o poi domina la predizione.

Nell'apprendimento bayesiano spesso succede che lo spazio delle ipotesi è molto grande, ma nella maggior parte dei casi è necessario ricorrere a delle approssimazioni.

Un'approssimazione molto comune è la **MAP (Massimo a posteriori)** che basa la propria predizione sulla ipotesi più probabile h_{MAP}, ossia quella con probabilità massima $P(h_i \mid d)$.

Sia nell'apprendimento bayesiano che nella MAP la distribuzione a priori $P(h_i)$ ha un ruolo fondamentale, spesso causa del sovraddattamento (overfitting). Se le probabilità a priori non sono note allora si introduce la MLE.

Si parla della **stima a massima verosimiglianza (Maximum Likelihoods Estimation)** quando la distribuzione sullo stato delle ipotesi è uniforme. L'apprendimento MLE è molto diffuso nonostante il fatto che presenti delle debolezze in caso di dataset piccoli.

La **differenza tra probabilità e verosimiglianza** è che la probabilità indica l'area sottostante ad una data distribuzione mentre la verosimiglianza indica il valore dell'ordinata in corrispondenza di una certa ascissa noto l'andamento dei dati.

Per utilizzare la MLE si raccolgono dei dati e da tali dati si studia per trovare i corretti parametri che rappresentano la distribuzione di probabilità del fenomeno studiato.

Spesso accade che le distribuzioni delle probabilità hanno un andamento gaussiano per il teorema centrale del limite ed in questi casi il compito da svolgere è quello di trovare i parametri del valore medio e della deviazione standard che rappresentano la gaussiana. La distribuzione deve essere simmetrica rispetto ad una zona molto densa di campioni, punto di accumulazione.

La stima a massima verosimiglianza cerca di trovare i parametri della guassiana, per massimizzare la verosimiglianza si fa "scorrere" la gaussiana sui dati e laddove essa raccoglie la maggior parte di osservazioni avremo un valore di likelihood più alto.

Alberi di Decisione e Random Forest

Un **albero di decisione** è una struttura dati che è ramificata che a partire da una radice fino ad arrivare alle foglie ed i percorsi che partono dalla radice fino alle foglie permettono di classificare o anche fare la regressione anche se solitamente effettuano la classificazione dell'osservazione all'interno di una certa classe. Data un'osservazione questa la si confronta con la radice a seguito di questo confronto verrà fatta scendere nell'apposito figlio fino ad arrivare ad una foglia la quale classificherà tale osservazione.

Leo Breiman fu un ricercatore che ha studiato gli alberi in tutte le loro forme ed è considerato il padre fondatore della teoria relativa a tutti gli alberi di decisione e in particolare modo la Random Forest.

Random Forest è uno dei classificatori più potenti sui dati al giorno d'oggi e si basa su un approccio molto banale e molti degli algoritmi di apprendimento odierni si basano su delle assunzioni molto semplici.

La Random Forest si basa sull'albero di decisione, in modo molto intuitivo e naturale classifica dei pattern attraverso una sequenza di domande e la sequenza domande-risposte è il percorso che mi permette di raggiungere una foglia partendo da una radice. Inoltre le classi possono ripetersi, non è detto che tutte le foglie rappresentano distintamente ognuna una classe diversa, vi possono essere più foglie che rappresentano la stessa classe.

Una foglia si definisce **pura** se tutti i campione all'intero di essa sono realmente la classe che rappresentano, una foglia si definisce **impura** se i campioni all'interno di essa sono campioni diversi.

Pretendere che tutte le foglie di un albero siano pure comporterebbe una complessità dell'albero enorme in quanto dovremmo considerare tutte le possibili combinazioni e questo potrebbe essere visto come un problema di overfitting.

Ovviamente le classi devono essere note, quindi tutte le foglie si conoscono.

CART

(Classification and Regression Trees)

È un algoritmo che consiste in:

1. Nell'assegnare tutti i nodi con un'etichetta al nodo radice
2. Sia N il nodo corrente
   2.1. Trova la caratteristica F e una soglia T tale che
      2.1.1. Sia possibile splittare N in due sottoinsieme (due figli) $S_{sinistro}$ e S_{destro}
      2.1.2. Sia tale split quello che massimizza la purità dei sottoinsiemi (i nodi che si creano devono avere una purità maggiore del padre)
   2.2. Se $S_{sinistro}$ e S_{destro} sono troppo piccoli per essere divisi ci si ferma
   2.3. Altrimenti
   2.4. Crea due nodi figli $N_{sinistro}$ e N_{destro} e procedi sui due nuovi nodi singolarmente come al punto 2

Con lo sviluppo di un albero di decisione bisogna considerare vari fattori ad esempio:

1. È sufficiente che le regole siano binarie e multicolore ?
2. Quale proprietà dovrebbe essere testata (che mi permette di andare verso la massima purità dei nodi)? E in quale ordine ?
3. Quando un nodo è da intendere come nodo-foglia ?
4. Se l'albero si espande in modo eccessivo come se ne può effettuare il pruning ?
5. Se il nodo-foglia è impuro quale etichetta assegnare alla classe che esso rappresenta ?
6. Come dovrebbero essere gestiti i dati mancanti ?

1 - Binario o multivalore ?

Qualsiasi albero di decisione con numero arbitrario di discendenti può essere convertito in albero binario a meno di un incremento dell'altezza dell'albero.

2 - Impurità del nodo

Un nodo si definisce **puro** quando i campioni in esso contenuti hanno la stessa etichetta.

Definiamo **l'impurità** di un nodo N come: **0 <= i(N) <= 1**

- i(N)=0 quando il nodo è puro;

- i(N)=1 quando l'impurità è massima, ossia campioni equiprobabili nel nodo;

Se si tratta di un problema di regressione si usa di solito lo **scarto quadratico medio**

$RSS_i = \sum_{Left} (y_i - < y_L >)^2 + \sum_{Right} (y_i - < y_R >)^2$**.**

(In generale ω_j indica la classe j**).**

Se si tratta di problemi di classificazione allora si calcola l'**Entropy Impurity**

$i(N) = -\sum_j P(\omega_j) log P(\omega_j)$**.**

L'impurità di un nodo la si può calcolare anche con la **Variance Impurity** nel caso in cui ci troviamo di un problema a due classi: $i(N) = P(\omega_i)P(\omega_j)$**.**

Se si generalizza a più classi si ottiene invece la **Gini Impurity (indice Gini)** calcolabile come $i(N) = \sum_{i!=j} P(\omega_i)P(\omega_j) = 1/2[1 - \sum_j P^2(\omega_j)]$**.**

Un ultimo valore di impurità è dato dalla **misclassification impurity** che si indica come $i(N) = 1 - max_j P(\omega_j)$ che altro non è che la probabilità di mis-classificare un nodo di training sul nodo N.

Scelta la misura di impurità, i**n funzione di quale caratteristica e soglia suddividere il nodo corrente N ?** È necessario studiare i dati ed individuare la caratteristica F e la soglia T che minimizzano la loro impurità: $max_{(F,T)} \triangle i(N) = i(N) - P_l i(N_l) - (1 - P_l) i(N_r)$,

$N_{l/r}$ indica o il figlio sinistro o il figlio destro, $i(N_{l/r})$ indica le impurità dei nodi figli sinistro/destro, $P_{l/r}$ sono le frazioni di nodi del nodo N che finiscono nel nodo figlio sinistro/destro.

3 - Quando interrompere la suddivisione

La scelta ideale è quella di arrivare alla minima impurità possibile, ma questo comporta un prezzo da pagare ovvero la complessità.

Una soluzione è quella di fissare una certa **soglia di minima riduzione di impurità** β (valore molto piccolo) nel momento in cui si ottiene $max \triangle i(N) <= \beta$ allora ci si ferma.

Il vantaggio è quello che si utilizza tutto il dataset, ma lo svantaggio è che l'albero potrebbe risultare non-bilanciato. Ciò significa che nel momento in cui dobbiamo classificare un dato

nel caso in cui l'albero non è perfettamente bilanciato potrebbe richiedere tempo O(n), a differenza del caso in cui è perfettamente bilanciato e quindi richiederebbe tempo O(logn).

Per alcune classi saprò classificare, ma per altre ci metterò molto tempo.

Un'altra soluzione potrebbe essere quella di individuare una **soglia sul numero di campioni** in un nodo, se un nodo contiene meno di x elementi allora non lo splitto più altrimenti continuo fin quando non raggiungo la soglia prefissata. Questa scelta non è molto saggia in quanto non ci garantisce che quegli elementi siano ben classificati.

In modo più generico si può definire un'ulteriore **criterio di arresto** espresso nel seguente modo**:** $\alpha * size + \sum_{N \in foglie} i(N)$, si tengono correlati insieme una serie di parametri come la dimensione dell'albero e la loro impurità.

Laddove size può indicare il numero di nodi o di collegamenti mentre α una costante positiva.

Come ulteriore approccio alternativo, si può usare un test statistico per misurare la significatività della riduzione di impurità, ovvero se splittando si ottiene un vantaggio o meno, come T-test, χ-squared, Mann-Withney, ANOVA, etc.

4 - Pruning

Nel momento in cui ho un certo albero che soddisfa i nostri requisiti di impurità, si può decidere che alcuni nodi non avrebbero più ragione di esistere e quindi è preferibile cancellarli.

A tal proposito si introduce il concetto di pruning il quale risolve il problema secondo il quale si slitta sempre un nodo finché è possibile minimizzare l'impurità; inoltre viene risolto un altro problema ovvero che due nodi "vicini", con lo stesso antenato vengono fusi in uno se l'incremento di impurità derivante dall'unione è inferiore ad una soglia. In questo modo l'antenato diventa una foglia. Praticamente se due nodi rispettano la soglia, ma in particolare modo il loro antenato rispetta questa soglia, ad esempio β oppure un'altra soglia che magari possiamo definire noi (soglia di pruning) allora i due nodi li posso unire in modo tale che continuano a rispettare il valore soglia (solitamente si fa negli alberi multivalore).

5 - Assegnare l'etichetta ad una foglia

Se il nodo è puro non si crea alcuna ambiguità.

Quando un nodo è impuro, la scelta di etichettare il nodo cade sulla classe che è maggiormente rappresentata da quel nodo.

Spesso potrebbe capitare che un albero di decisione non si adatta alla forma dei dati di training, avendo una forma particolarmente complessa. A tal proposito si può effettuare un

lavoro di pre-processing il quale può aiutare ad esempio si potrebbe utilizzare la PCA, analisi delle componenti principali.

6- Dati Mancanti

In presenza di **deficient pattern** (attributi mancanti) è possibile scegliere di effettuare più strategie in base alle ragioni di mancanza del dato, in quanto un dato mancante non sempre è un'assenza di informazione.

Soluzione naive: elimino tutti i campioni con attributi mancanti (tecnica molto semplice, ma grande spreco di informazione).

Surrogate split:

- ogni nodo non foglia viene annotato con una **regola di splitting "primaria"** (quella che massimizza la riduzione di impurità);

- in aggiunta a questa vengono create delle **regole surrogate** (attributo, regola) le quali massimizzano la "predictive association" con la regola primaria;

- le regole surrogate sono ordinate in ordine decrescente di correlazione con la regola primaria;

- se la regola primaria ha degli attributi mancanti allora si passa alla prima delle regole surrogate in funzione della quale non c'è nessun deficient pattern;

Pro vs. Contro degli Alberi di Decisione

PRO	CONTRO
Sia per regressione che classificazione	Accuratezza: SVM ha un media di errore minore del 30% rispetto a CART
Computazionalmente semplici, rapida adattabilità ai dati	Instabilità: un cambiamento dei dati di training anche lieve può dare origine a rappresentazioni dell'albero molto diverse
Non parametrico (non ci preoccupiamo di calcolare i parametri della distribuzione dei dati)	
Completamente automatizzatile	
Permette di gestire gli attributi mancanti	
Molto semplice da interpretare quando l'albero non è eccessivamente esteso	

La soluzione per i contro però è la Random Forest.

Come funziona la costruzione di un albero di decisione ?

Per ogni feature calcolo il valore di impurità, ad esempio la Gini impurity. Nella radice inserisco la feature che minimizza questo valore e quindi la più pura, itero questo procedimento per tutti i nodi a seguire.

Cosa succede se gli attributi non sono del tipo si/no ?

Bisogna sempre trovare una soglia di separazione del nodo in sottonodi. Per fare quest'operazione si ordinano i valori e si individuano delle "probabili soglie" per ognuna di esse se ne calcola l'indice Gini e si sceglie come valore soglia, quella che minimizza il valore Gini, ed è possibile continuare effettuando delle successive diramazioni.

Cosa succede se gli attributi non sono di tipo numerico ?

In tal caso si associa ad ogni valore un codice numerico, si effettua una codifica. Oppure faccio delle proposizione logiche del tipo (A or B) quindi se non è nessuna va in false altrimenti in true e poi faccio A che può essere true o false.

Random Forest

Per parlare delle Random Forest bisogna parlare di **Bagging** (Bootstrap AGGregatING algorithm) il quale si basa sul bootstrap.

Il **bootstrap** è un metodo statistico di resampling molto utile quando le osservazioni sono limitate e non c'è possibilità di ottenerne delle nuove. Tale tecnica consiste, partendo da un dataset originale D si estrae un numero K di campioni in maniera random con reinserimento producendo B sottoinsiemi di D. Su ciascuno dei B sottoinsiemi si produce la misura statistica di interesse e si media su tutte le misure ottenute sui dataset di bootstrap.

L'accuratezza del bagging è tanto maggiore quanto più è alta la non-correlazione tra i sottoinsiemi di bootstrap, ovvero sono sottoinsieme disgiunti.

La Random Forest costruisce gli alberi di decisione per ogni sottoinsieme B, quando devo classificare un campione verrà mandato a tutti questi alberi di decisione e la classe più votata sarà quella relativa al campione analizzato. Nella Random Forest il rischio di overfitting è veramente molto basso, si potrebbe verificare solamente nel caso in cui il numero degli alberi è pari al numero degli elementi del dataset.

Una foresta è un insieme di molti alberi

L'algoritmo consiste in:

1. A partire dal training set D si costruiscono B sottoinsiemi di boostrap;
2. Per ciascuno dei B sottoinsiemi si costruisce un albero di decisione;
3. Ad ogni nodo:
    A. Seleziono m variabili in maniera random tra le M possibili e lo si fa in modo indipendente per ogni nodo;

B. Trovo il miglior criterio di split per tale sottoinsieme di m variabili;

4. Fa cresce ciascun albero fino alla massima altezza in relazione al criterio di splitting adottato senza effettuare il pruning;

Quando un campione viene dato in input alla foresta, esso passa attraverso ciascun albero separatamente.

La classe del nuovo campione verrà scelta per mezzo di un processo di voto o di media delle risposte di tutti gli alberi, si preferisce a tal proposito avere un numero dispari di alberi.

Pro vs. Contro:

-Pro: gli stessi di CART, ma si gestisce in modo più immediato la presenza di attributi mancanti;

-Contro: si perde l'intuitività del CART;

SVM & PCA

La qualità del dataset è importante per evitare che un dataset non overfitti, risulta essere importante quindi scegliere in modo accurato le caratteristiche del dataset da analizzare, ma molto spesso tali combinazioni di caratteristiche così rappresentate nella loro forma originale non sono semplice da rappresentare per vari motivi (ambiguità, non è possibile trovare l'iperpiano/curva per separare i dati).

Per risolvere questo problema l'utilizzo di SVM in questo caso oppure PCA permette di modificare lo spazio all'interno del quale si trovano le caratteristiche, creandone uno nuovo che permette di individuare facilmente l'iperpiano di rappresentazione.

I due approcci PCA e SVM sono agli antipodi in quanto lo scopo di PCA è quello di ridurre il set di caratteristiche e SVM ha lo scopo di aumentare il set di caratteristiche. Queste due tecniche affrontano lo stesso problema, ma con approcci diversi.

SVM - Support Vector Machine

Quando si utilizza la SVM lo si fa perché si ritiene che i dati di partenza non siano adeguati al problema e per tal motivo si aggiungono nuove caratteristiche, ma non nel senso che effettuo una nuova operazione di raccolta dei dati. Dai dati originali SVM crea dei nuovi dati i quali si sono ottenuti tramite combinazioni lineari dei dati precedenti aggiungendo nuove caratteristiche in modo tale da poter separare facilmente i dati, l'esatto contrario di PCA.

Un **classificatore SVM** cerca la separazione lineare che meglio classifica i dati, massimizzando il margine (d1=d2) di separazione tra le classi. Immaginiamo un problema binario con due classi, l'obiettivo di SVM è quello di trovare la retta di separazione che taglia

il dataset in due e massimizza il margine del campione più vicino di una classe alla retta a sinistra e massimizza il margine del campione dell'altra classe più vicino a destra.

Tale classificatore è banale in quando i dati sono linearmente separabili, ma nel caso in cui non lo sono è tutto molto più complesso.

Definendo la SVM in modo più formale: si supponga di avere il training set L=$\{x_1, \ldots, x_n\}$ in cui ciascun x_i è rappresentato da D caratteristiche e i campioni possono appartenere a due sole possibili classi $y_1 = -1$ e $y_2 = +1$.

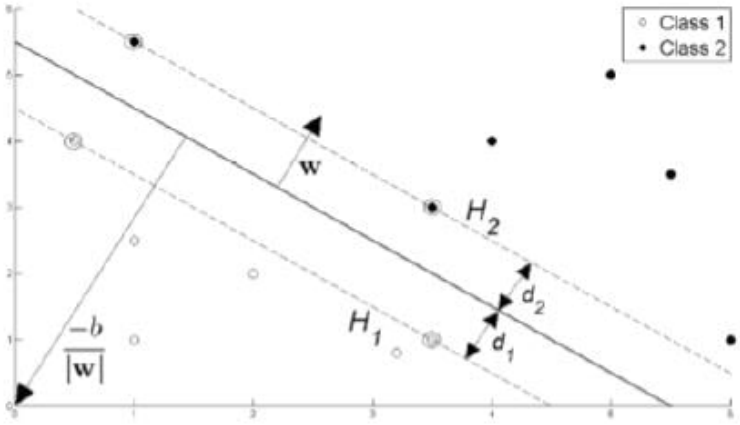
$$\{x_i, y_i\} \text{ con i=1,..., L, } y_i \in \{-1,1\}, \text{ x } \in R^D$$

Adesso assumiamo che il dataset sia linearmente separabile e quindi esiste un iperpiano che separa le due classi.

L'iperpiano è definito dalla seguente equazione $w^T x + b = 0$, dove

- w indica la normale all'iperpiano;
- $\frac{b}{||w||}$ è la distanza perpendicolare dell'iperpiano dall'origine degli assi (b sulla norma w)

Queste due informazioni permettono di capire a che punto del piano si trova l'iperpiano e quale è la sua inclinazione.



Come mostrato in figura sopra il classificatore SVM permette di selezionare i valori di w e b tali che i dati di training possano essere descritti per mezzo di un **classificatore lineare** $g(x) = w^T x + b$ tale che:

- $w^T x_i + b >= +1 \forall y_i = +1$
- $w^T x_i + b <= -1 \forall y_i = -1$

Volendo è possibile combinare i due vincoli in una sola disuguaglianza del tipo:

$$y_i(w^T x_i + b) >= 1 \forall y_i$$

Inoltre per semplicità viene mostrato in un piano a due dimensioni, ma tale concetto lo si applica in qualsiasi piano n-dimensionale laddove i dati sono però linearmente separabili, praticamente l'iperpiano avrà la stessa dimensione dei dati.

Si effettua la **trasposta di w** in quanto essendo w e x_i grazie alla trasposta di w si può effettuare l'operazione di prodotto righe per colonna.

Bisogna affermare che SVM si formalizza a due classi, ma lo si può generalizzare anche ad n classi.

Definiamo **vettori di supporto** i vettori che sono più vicini tra di loro che appartengono all'una e all'altra classe e all'interno di questa area tracciata dai due vettori di supporto verrà individuato l'iperpiano.

Se consideriamo i soli campioni più vicini all'iperpiano di separazione, i vettori di supporto, i due iperpiani H_1 e H_2 su cui giacciono tali punti sono tali che:

- $w^T x_i + b = -1$ per H_1
- $w^T x_i + b = +1$ per H_2

L'obiettivo di SVM è quello di trovare l'orientazione dell'iperpiano che separa al meglio le due classi con il requisito che il margine tra i vettori di supporto delle due classi da esso costituito sia massimo.

Tale margine lo si può esprimere come $\frac{1}{||w||}$ e quindi massimizzare il margine significa $min||w||$ tale che $y_i(w^T x_i + b) >= 1 \forall i$.

Tutto ciò di cui abbiamo parlato era verificato nel caso in cui i dati erano linearmente separabili, ma nel caso in cui non lo sono cosa bisogna fare ? È possibile aggiungere una nuova caratteristica che combina due o più caratteristiche (combinazione che può essere lineare, quadratica, etc), non faccio altro che acquisire nuove caratteristiche solo da dati che io già conosco, non modifico nient'altro. A tal punto il piano diventa linearmente separabile se considero il piano di cui una dimensione corrisponde alla nuova dimensione appena creata.

Come è stato detto il nostro compito è quello di ottenere min||w|| il quale si può anche modificare nel seguente modo: $min||w|| \equiv min\frac{1}{2}||w||^2$. Lo si trasforma in questo modo in quanto permette di rappresentare un problema di ottimizzazione che si risolve con la programmazione quadratica attraverso l'introduzione dei moltiplicatori di Lagrange i quali danno origine alla formulazione del problema di minimizzazione primale.

$$L(w, b, \alpha)_P = \frac{1}{2}||w||^2 + \sum_{i=1}^{L} \alpha_i(1 - y_i(w^T x_i + b))$$

Sviluppando la formula sopra riportata e per trovarne il minimo bisognerà derivare rispetto a w e rispetto a b. Gli $\alpha_i > 0$ corrisponderanno ai vettori di supporto, i punti dove passano i punti di supporto.

Nel caso in cui lo spazio non sia linearmente separabile si può utilizzare la SVM lineare introducendo però il concetto della **soft margin** secondo il quale i punti mal classificati dovranno avere una penalità che è proporzionale alla distanza dall'iperpiano di separazione, quanto si discosta il miglior vettore di supporto dai campioni che pur appartenendo alla stessa classe si trova nella zona dell'altra classe.

Quando la separazione lineare non è garantita allora si utilizza tale espediente.

La SVM non lineare quando viene applicata a dati linearmente separabili crea una matrice H dal prodotto interno dei campioni di training del tipo:

$$H_{i,j} = y_i y_j k(x_i, x_j) \text{ con } k(x_i, x_j) = x_i \cdot x_j = x_i^t x_j$$

$k(x_i, x_j)$ prende il nome di kernel function e nel caso in cui $k(x_i, x_j) = x_i^T x_j$ prende il nome di kernel lineare.

Se ad esempio $k(x_i, x_j) = (x_i \cdot x_j + \alpha)^d$, con d >= 2 allora parleremo di kernel polinomiali.

Generalmente quando si rimpiazza il prodotto interno dei vettori con una kernel function allora si parla di kernel trick.

Come kernel function si può trovare anche una gaussiana.

Pro vs. Contro

È basato su un'ottimizzazione standard per il calcolo dei suoi parametri.	La complessità è proporzionale al numero dei campioni per cui i solverà QP generali spesso falliscono con numeri > 1000 campioni
Confini non lineari possono essere individuati senza sovraccarico computazionale	È intrinsecamente un classificatore binario. Per avere un classificatore SVM su K(>2) classi bisogna costruire K classificatori SVM distinti per separare una classe dalle K-1 restanti introducendo il vincolo di oneclass-against-the-rest nel problema di ottimizzazione.
Computazionalmente competitivo	

Le SVM sono molto utilizzate nelle face detection, classificazione di immagini, bioinformatic.

PCA - Principal Component Analysis

Quando si parla di PCA non si affronta un reale problema di classificazione, ma un problema di pre-processing sui dati. Mentre nel SVM si prendono i dati così come sono e se ne effettuano le combinazioni, venendo a creare però delle nuove feature. Se le feature non sono molto rappresentative del problema molto probabilmente non lo saranno neanche le feature generate da esse.

L'analisi delle componenti principali è una tecnica utile per la compressione e classificazione dei dati. Lo scopo è quello di ridurre le dimensioni del dataset, cercando un nuovo insieme di variabili con dimensione minore dell'insieme originario e rappresentativo del contenuto informativo del dataset.

Se studio la distribuzione delle feature se individuo che due o più feature sono l'una la combinazione lineare dell'altra, volendo posso anche considerare solo una di esse. Se riesco ad individuare solo le **feature linearmente indipendenti** posso considerare solamente queste andando a scremare la rappresentazione del dataset diventando più compatta senza perdere informazioni.

Immaginiamo di avere un dataset i cui dati sono disposti ad esempio orientativamente lungo la bisettrice del piano allora potremmo individuare una rette di regressione che li attraversi, e settandola come prima componente principale individuando una nuova combinazione delle componenti. Riuscendo ad ottenere così una distribuzione dei dati lungo una sola componente, ciò ci permette di utilizzare solo una componente per individuare quei dati anziché due tenendo sempre conto però dello scarto quadratico medio ovvero la distanza dai campioni dalla retta di regressione.

Il **metodo dei minimi quadrati** consente di determinare la **retta di regressione** o dei **minimi quadrati**. Siano $X=\{x_1, x_2, \ldots, x_n\}$ e $Y=\{y_1, y_2, \ldots, y_n\}$ i dati sperimentali di una popolazione che sono rappresentati sul piano cartesiano 2D come le coppie (x_i, y_i) per i=1,…,n

Si definisce **retta di regressione** o dei minimi quadrati, la retta di equazione $y = \alpha x + \beta$ per la quale risulta minima la somma degli scarti $E = \sum_i^n (\alpha x_i + \beta - y_i)$.

La distanza di un punto dalla retta di regressione è la distanza verticale del punto dalla retta. Inoltre si dimostra che $\alpha = \frac{cov(X,Y)}{\sigma_x^2}$ e $\beta = \mu_Y - \alpha\mu_X$.

L'analisi delle componenti principali produce la retta per la quale la somma degli scarti quadratici delle distanze perpendicolari dei campioni dalla retta è minima.

La **prima componente principale** è proprio tale retta, la **seconda componente principale** segue lo stesso principio con il vincolo che deve essere ortogonale alla prima componente principale. In un problema n-dimensionale lo stesso approccio si applica a tutte le altre successive componenti.

Inoltre la retta di regressione deve anche essere tale da conservare la **massima varianza** tra i dati continuando a distinguere un campione dall'altro, la seconda componente principale è la massima che conserva la massima varianza tra i dati dopo la prima, la terza a sua volta sarà la massima che conserva la massima varianza tra i dati dopo la prima e la seconda. Possiamo immaginare che le componenti sono ordinate dalla più conservatrice della varianza alla meno conservatrice della varianza.

Se vogliamo definire **algebricamente la PCA**, affermiamo che dato un campione di n osservazioni di un vettore a p variabili X=$\{x_1, x_2, \ldots, x_p\}$, si definisce prima componente principale dei campioni X, la trasformazione lineare $z_1 \equiv a_1^T x = \sum_{i=1}^{p} a_{i1} x_i$ laddove il vettore $a_1 = (a_{11}, a_{21}, \ldots, a_{p1})$ è scelto in maniera tale che $var[z_1]$ è massima.

In modo generico definiamo la k-esima componente principale dei campioni X, la trasformazione lineare $z_k \equiv a_k^T x$ con k=1,…,p laddove il vettore $a_k = (a_{1k}, a_{2k}, \ldots, a_{pk})$ è scelto in modo tale che $var[z_k]$ è massima e con il vincolo che $cov[z_k, z_l] = 0$ per 1<= l (elle)< k (praticamente la covarianza della componente k-esima e tutte le sue precedenti vale 0) ed inoltre $a_k^T a_k = 1$.

La PCA è stata ampiamente utilizzata per lo studio delle **eigen faces** (auto facce), partendo da un dataset di volti (ORL faces) la PCA è tale che per ciascun soggetto individua la sua auto faccia indipendentemente dalla sua orientazione soffermandosi solamente sui tratti somatici.

La PCA per la **compressione di immagini** è molto utile in quanto è intrinsecamente un algoritmo di compressione. La riduzione di dimensione dello spazio sarà tanto più efficace quanto più alta sarà la compressione dell'immagine preservandone il suo aspetto.

Per valutare la compressione si utilizza un **peak signal-to-noise ratio** (PSNR) ed è una misura adottata per valutare la qualità di un'immagine compressa rispetto all'originale. Maggiore è il valore del PSNR maggiore sarà la somiglianza con l'immagine originale. Livelli crescenti di compressione abbassano il PSNR, i valori compresi tra 20 e 40 sono considerati accettabili.

Inoltre la PCA la si può utilizzare come **filtro di soppressione di rumore in un'immagine**, la soppressione di pixel spuri all'interno di un'immagine. Un'immagine con rumore contiene una bassa varianza in quanto significa che i pixel sono disposti in modo omogeneo e quindi utilizzando la PCA si prenderanno solo le componenti con una varianza alta escludendo quei pixel spuri.

Misurazione dell'affidabilità dei sistemi

Affidabilità: capacità del test di offrire sempre lo stesso risultato nel corso di misurazioni ripetute;

Validità: capacità del test di distinguere in una popolazione elementi di classi differenti;

Cut-off ideale: indica il valore di taglio ideale che il classificatore individua per separare i dati appartenenti ad una determinata classe dai dati appartenenti all'altra, il valore di taglio in quel punto è netto e non dà origine ad equivoci.

Cut-off reale: indica il valore di taglio reale che il classificatore individua e bisogna fare attenzione perché ad esempio nel caso di malattie, un soggetto sano potrebbe essere un falso positivo e un soggetto malato potrebbe essere un falso negativo. I risultati che si possono ottenere a seconda della posizione del valore di cut-off sono 4:

1. Se il risultato della predizione è positivo ed il valore vero è anche esso positivo allora il soggetto viene chiamato vero positivo (TP)
2. Se invece il valore vero è negativo, il risultato viene chiamato falso positivo (FP)
3. Contrariamente si ha un vero negativo (TN) quando entrambi, sia la predizione che il valore vero sono negativi
4. Un falso negativo (FN) invece si ha quando il risultato è negativo e il valore vero è positivo

La tabella di contingenza mette in relazione tutte le possibili combinazioni che si possono avere fra TP, TN, FP, FN.

Dallo studio dei FP e FN nascono due nuove misure:

Alta Sensibilità: indica che vi è un'alta probabilità che un soggetto malato risulti positivo al test; bassa probabilità che un soggetto malato risulti negativo al test;

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Alta Specificità: indica che vi è un'alta probabilità che un soggetto sano risulti negativo al test; bassa probabilità che un soggetto sano risulti positivo al test;

Le due misure sono speculari l'una rispetto all'altra.

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP}$$

Un'ulteriore distinzione che dovremo fare sarà quella fra:

Precisione: indica il grado di "convergenza"/"dispersione" dei dati rilevati individualmente rispetto al valore medio della serie cui appartengono

Accuratezza: indica il grado di corrispondenza del dato teorico con il dato reale o di riferimento ovvero la differenza tra il valore medio campionario e valore vero o di riferimento

La curva ROC e EER:

ROC: indica la Receiver Operating Characteristic ed è la curva che mette in relazione gli allarmi veri dai falsi allarmi, ha come valori sull'ascissa la specificità e sull'ordinata la sensibilità, è ottima tale curva ROC quando ha un andamento perpendicolare ed è parallela alle ordinate

EER: indica l'Equal Error Rate costituisce il punto in cui il tasso dei falsi positivi eguaglia i falsi negativi, più basso è tale valore e maggiore risulta l'accuratezza del metodo

K-Fold Cross Validation

La convalida incrociata, cross validation, è una tecnica ampiamente utilizzata per stimare correttamente un classificatore, soprattutto quando i dati a disposizione sono pochi.

La tecnica del K-Fold Cross Validation consiste nell'estrazione dal dataset originale di training una parte dei dati questo viene ripetuto per k volte con porzioni diverse. Ad ogni passo la k-esima partizione del dataset viene usata in modo che il metodo venga addestrato sulla porzione di training e validata sulla parte di validazione estratta dal dataset.

La validazione consiste nel eseguire il classificatore addestrato su una parte dei dati del dataset e se mi mantengo al di sotto di una certa soglia di errore allora posso eseguire il test reale, altrimenti ri-addestro il classificatore.

Apprendimento Non Supervisionato

[Si parla di **punto Fazi** quando un dato può appartenere a più di una classe o cluster.]

Non sempre si ha un'identificazione dei campioni di training ad una particolare classe e quando ci troviamo in una situazione di questo tipo si parla di **apprendimento non supervisionato**.

L'apprendimento non supervisionato ha trovato grande applicazioni nell'ambito medico in quanto permette di separare e raggruppare in gruppi i pazienti che sono accomunati da determinati parametri non necessariamente con la stessa malattia, ma perché hanno caratteristiche simili.

Un **cluster** è formato da un insieme di oggetti simili raccolti e raggruppati assieme, in cui la similarità ha senso se rapportata al problema che sia sta esaminando. Si ha quindi un insieme di entità simili che costituiscono un'aggregazione di punti nello spazio tale che la distanza tra qualsiasi coppia di punti dello stesso cluster sia minore della distanza tra punti di cluster differenti.

È possibile immaginare i cluster come regioni connesse in uno spazio multi-dimensionale, contenenti una densità relativamente alta di punti, separati da altre regioni mediante una regione che contiene una bassa densità di punti.

Il problema del clustering non è un problema banale perché i dati si possono presentare in differenti forme e dimensioni, il numero di cluster dipende dalla "risoluzione"/"granularità" con cui si studiano i dati.

Inoltre la mancanza di etichette non mi permette neanche di capire se il raggruppamento da me calcolato può essere corretto o meno, a differenza della classificazione.

Gli algoritmi di clustering possono essere numerosi, si suddividono in varie tipologie:

- **Esclusivo**: ogni pattern appartiene ad un solo cluster;
- **Non esclusivo**: ogni pattern potrà essere assegnato a più di un cluster;
- **Gerarchico**: vi sono delle sequenze innestate di partizioni;
- **Partizionale**: vi è una singola partizione;

Un'ulteriore distinzione che si può eseguire sugli algoritmi di clustering è in:

- **Agglomerativi**: si parte dai singoli campioni e si formano dei cluster sempre più popolati, riducendo il numero di cluster e rendendoli molto densi;
- **Divisivi**: si parte dal dataset considerandolo come unico cluster e dividerlo poi di volta in volta;
- **Seriali**: elaborazione dei pattern (i campioni) uno ad uno;
- **Simultaneo**: elaborazione dei pattern (i campioni) in contemporanea;
- **Graph-theoretic**: si basa sulla collegabilità, basati sulla teoria dei grafi;
- **Algebrico**: si basa sui criteri di errore, (K-means);

La maggior parte degli algoritmi di clustering si basa su due tecniche molto note:

- Partizionamento iterativo ad errore quadratico
- Clustering gerarchico agglomerativo

Uno dei principali cambiamenti tra un algoritmo di clustering ed un altro è quello di selezionare un'**appropriata misura della similarità** per definire cluster che spesso sono sia dipendenti dai dati (forma del cluster) che dal contesto.

Un'altra caratteristica da tenere in considerazione negli algoritmi di clustering è misurare la **compattezza del cluster** e quindi la sua alta densità, ovvero il fatto che data la sua forma questa deve essere molto popolata.

La **scelta della misura di similarità** è un grande problema degli algoritmi di clustering, la misura di clustering più ovvia tra due pattern è la distanza tra essi, la distanza euclidea.

Se la distanza rappresenta una buona misura della diversità allora possiamo imporre che la distanza tra due pattern nello stesso cluster sia significativamente minore della distanza tra due pattern appartenenti a differenti cluster. Banalmente a questo punto, un modo per fare clustering potrebbe essere quello di scegliere la soglia sulla distanza e raggruppare i pattern che sono più vicini rispetto alla soglia fissata.

La **distanza metrica** si definisce sulla base di un dato insieme S di campioni, una distanza $d : SxS -> R$ è metrica se soddisfa tutte le seguenti proprietà:

1. **Identità**: $\forall x \in S, d(x, x) = 0$
2. **Positività**: $\forall x != y \in S, d(x, y) > 0$
3. **Simmetria**: $\forall x, y \in S, d(x, y) = d(y, x)$
4. **Disuguaglianza triangolare**: $\forall x, y, z \in S, d(x, z) <= d(x, y) + d(y, z)$

Una distanza si definisce **semi-metrica** se soddisfa solamente le prime 3 proprietà.

Una distanza si definisce **pseudo-simmetrica** quando solamente la 1,3 e 4 sono soddisfatte.

Esistono altri tipi di distanze: euclidee (cerchio), Manhattan (quadrato), Mahalanobis (ellisse).

Grazie alla misura di similarità si può decidere quale campione apparterrà a quale gruppo, però dopo aver effettuato tale operazione bisogna eseguire la scelta del **criterio da ottimizzare**, ovvero valutarne la compattezza, quindi se a valle di un processo di clustering quei cluster sono realmente corretti. Immaginiamo di avere un insieme D=$\{x_1, \ldots, x_n\}$ di n campioni che vogliamo partizionare in esattamente k sotto insiemi disgiunti $D_1, \ldots, D_k$. Ogni sottoinsieme rappresenta un cluster, con i campioni nello stesso cluster che sono per qualche motivo più simili l'un l'altro rispetto ai campioni negli altri cluster. Il criterio più semplice e più usato per fare clustering è il criterio della **somma degli errori quadrati**.

La **stima della somma dei quadrati (squared sum estimate, SSE)**: supponiamo che l'insieme dato di n pattern sia stato partizionato in qualche modo in k cluster, $D_1, \ldots, D_k$.

Supponiamo che n_i sia il numero di campioni in D_i e supponiamo che m_i sia la media di quei campioni e quindi varrà: $m_i = \frac{1}{n_i} \sum_{x \in D_i} x$

Allora avremo che la somma dei quadrati degli errori è: $j_e = \sum_{i=1}^{k} \sum_{x \in D_i} ||x - m_i||^2$

Per un cluster D_i il vettore delle medie m_i detti **centroidi** è la migliore rappresentazione dei campioni nel dataset.

Se ho un centroide di un cluster ed i dati di quel cluster sono vicini al centroide allora avrò un'area molto densa, viceversa se i dati distano molto dal centroide avrò una somma degli errori molto alta e mi indica che il cluster così formato non va bene e quindi devo ripetere il processo di clustering, ma cambiando il numero di cluster.

Un **algoritmo** di clustering per il partizionamento ad errore quadratico è:

1. Seleziona k centroidi;
2. Repeat
3. Generare un partizionamento assegnando ogni campione al centroide più vicino
4. Calcolare nuovi centroidi dei cluster
5. Until i centroidi non cambiano
6. Adattare il numero dei cluster unendo o dividendo i cluster esistenti o rimuovendone alcuni esclusi

Se prendiamo in considerazione questo algoritmo senza il passo 6 non abbiamo altro che il k-means.

Il **K-Means** parte con un grosso svantaggio ovvero il fatto che devo scegliere a priori il numero di cluster (centroidi), la scelta dei primi k centroidi può essere fatta in modo completamente random oppure mediante una conoscenza a priori dei dati.

Questo algoritmo dovrebbe convergere ad un valore e poi al passo 5 quindi uscire dal loop, ma questa situazione accade che spesso non si verifica mai e quindi bisogna individuare una **soglia di limite**, quando praticamente la variazione ad una iterazione e all'iterazione successiva non è più significativa.

L'algoritmo K-Means è efficiente a livello di calcolo e fornisce buoni risultati se i cluster sono compatti, ipersferici nella forma e ben separati nelle caratteristiche.

La **partizione iniziale** (suddivisione iniziale dei cluster) è spesso scelta generando k punti casuali uniformemente distribuiti con l'ampiezza dei dati, o selezionando a caso k punti dai dati in possesso.

Tale algoritmo quindi potrebbe **convergere** molto lentamente o rapidamente a seconda dei parametri scelti.

Un altro problema di questo algoritmo è che due **esecuzioni diverse** dello stesso algoritmo non ci forniscono gli stessi cluster, per tale motivo per valutare il k-means non bisogna valutarlo una sola volta, ma va eseguito un certo numero di volte.

Vi sono numerose **varianti** del k-means:

- Incorporare in esso un criterio fuzzy
- Utilizzare algoritmi genetici
- Utilizzare lo splitting iterativo per la ricerca della partizione iniziale

Inoltre se si osserva che le distribuzioni dei dati dei singoli cluster hanno una forma gaussiana allora modificheremo il k-means in modo tale da non calcolare più la distanza euclidea, ma le misture gaussiane e tale variante prende il nome di **clustering model-based**, in tale variante il valore di k corrisponde ai numeri delle componenti della mistura.

Clustering Gerarchico

L'algoritmo k-means è un clustering di tipo **flat** in quanto tutti i campioni appartengono o ad A o ad B, quindi i cluster sono disgiunti e allo stesso livello.

Nel clustering è possibile stabilire delle gerarchie di cluster ed in base al livello in cui studio il mio problema posso avere un numero più o meno variabile di cluster. Si definiscono delle gerarchie e quindi anziché essere di tipo flat, sarà di tipo **multilivello**.

Nel clustering gerarchico per un insieme di n campioni:

1. Il primo livello è fatto da n cluster (ogni cluster contiene esattamente un campione)
2. Il secondo livello contiene n-1 cluster
3. Il terzo livello ne contiene n-2
4. E così via fino all'ultimo (ennesimo) livello in cui tutti i campioni formano un singolo cluster

In questa tipologia di clustering non si valuta più solo la similarità tra due campioni, bensì anche la similarità tra cluster, possono valutare se due cluster sono simili o meno e quindi se fonderli o no.

La rappresentazione più ovvia del clustering gerarchico è quella di un albero, anche chiamato **dendrogramma**, che mostra come vengono raggruppati i campioni.

Se esiste un gap particolarmente grande tra valori simili per due livelli particolari, allora è probabile che il livello con un più basso numero di cluster rappresenti un raggruppamento più naturale.

L'**asse verticale** indica una misura generalizzata della similarità tra clusters.

L'**asse orizzontale** indica il numero di clusters ad un determinato livello.

L'algoritmo del **clustering gerarchico agglomerativo** consiste in:

1. Si specifica il numero dei clusters richiesti. Si pone ogni pattern in un unico cluster.
2. Si trovano i clusters più vicini rispetto ad una misura di distanza
3. Si uniscono questi due clusters
4. Si ottiene il cluster risultante

Le misure di distanza più note, fissati due clusters D_i e D_j sono:

1. $d_{min}(D_i, D_j) = min_{x \in D_i, x' \in D_j} ||x - x'||$
2. $d_{max}(D_i, D_j) = max_{x \in D_i, x' \in D_j} ||x - x'||$
3. $d_{avg}(D_i, D_j) = \frac{1}{D_i D_j} \sum_{x \in D_i} \sum_{x' \in D_j} ||x - x'||$
4. $d_{mean}(D_i, D_j) = ||m_i - m_j||$, laddove m_i e m_j rappresentano i centroidi

Quando si utilizza come misura di distanza d_{min} allora l'algoritmo prende il nome di: **clustering del nearest neighbor**.

Inoltre, se l'algoritmo termina quando la distanza tra i clusters più vicini eccede una soglia fissata, si parla di algoritmo a singolo collegamento (**single linking**) per il quale:

- I pattern rappresentano i nodi di un grafo
- I bordi che connettono i pattern appartengono allo stesso cluster (il cerchio)
- Unire due cluster corrisponde ad aggiungere un bordo tra due coppie più vicine di nodi in questi cluster

Quando si utilizza come misura di distanza d_{max} allora l'algoritmo prende il nome di:

clustering del farthest neighbor.

Inoltre, se l'algoritmo termina quando la distanza tra i cluster più vicini eccede una soglia, è detto algoritmo del **full linking**, per il quale:

- I pattern rappresentano i nodi dei grafi
- I bordi connettono tutti i pattern appartenenti allo stesso cluster
- Unire due cluster corrisponde ad aggiungere un bordo aggiuntivo tra ogni coppia di nodi in questi clusters

La scelta della misura di distanza da utilizzare dipende molto dai dati, se la forma dei dati segue una forma non compatta è il caso di considerare il single linking, invece se risultano avere una forma molto più compatta allora è il caso di utilizzare il full linking.

Pro vs. Contro del Single Linking

- PRO: può separare campioni a forma non ellittica fintanto che il confine tra i cluster non è troppo piccolo (banana set)
- CONTRO: in presenza di dati rumorosi, l'algoritmo fallisce (quando elementi di cluster diversi hanno una distanza minima)

Pro vs. Contro del Full Linking

- PRO: può separare campioni in presenza di rumore
- CONTRO: tende a spezzare cluster di grandi dimensioni, mostra debolezze nel caso in cui i dati raggruppati presentano forme ipersferiche di dimensioni diverse

DBSCAN (Density-Based Spatial Clustering of Application with Noise)

L'algoritmo DBSCAN risolve i problemi legati al k-means come ad esempio il fatto che il numero dei cluster viene stabili a priori. L'algoritmo appena presentato risolve i problemi del k-means basando il raggruppamento sul concetto di densità dei campioni.

Esso prende in input due parametri:

- **minPts**: il numero minimo di punti per considerare un intorno di un punto denso
- ϵ: la distanza che definisce un intorno circolare di ciascun punto dai suoi vicini

La scelta dei parametri è arbitraria, dipende sempre dalla distribuzione dei dati, in generale se un dataset presenta una distribuzione dei suoi campioni molto sparsa, ciò indica che non è una buona rappresentazione del problema in questione.

Inoltre a differenza del k-means dove tutti i campioni di un cluster sono vicini ad un centroide, quindi un certo punto, in questo caso, invece, non succede questo, ma tutti i punti che sono collegati l'uno all'altro lungo un percorso appartengono allo stesso cluster.

I punti isolati vengono riconosciuti correttamente come rumore in quanto nel loro intorno non vi è un minPts.

Come stimare la qualità dei Cluster ?

Nei problemi di apprendimento non supervisionato, l'assenza di etichette rende impossibile stimare l'accuratezza/precisione dei cluster che si sono formati.

Per misurare il potenziale del raggruppamento prodotto si può ricorre all'uso di:

- Punto a gomito (elbow point)
- Coefficiente di forma (silhouette coefficient)
- Adjusted Rand Index (ARI)

Elbow Point: è un tecnica che rappresenta su un grafico, sulle ascisse i possibili numeri di cluster e sulle ordinate la SSE, il punto dove si forma il "gomito", ovvero la variazione più brusca, indica il numero più corretto di cluster da utilizzare, la curva della SSE.

Coefficiente di forma: è una misura della coesione e separazione tra i dati. Quantifica quanto i dati sono ben disposti nei cluster assegnati in funzione di due parametri:

1. Quanto bene essi sono ammassati nel cluster assegnato

2. Quanto è distante ciascun campione da qualsiasi altro cluster

Il coefficiente di forma varia tra -1 e +1. Valori alti indicano condizioni maggiori di coesione e separazione dei dati.

Il coefficiente di forma presenta dei limiti in quanto ad esempio nel caso del banana set tale coefficiente raggrupperebbe in modo errato i campioni.

Adjusted Rand Index (ARI): se in un algoritmo di clustering, i dati che abbiamo hanno delle etichette allora possiamo indicare la bontà dei clusters analizzando la similarità tra le etichette reali e quelle predette dal metodo di clustering. Se si confronta il comportamento di K-means e di DBSCAN, sullo stesso banana-set si note che in termini di indice ARI c'è maggiore coerenza con il comportamento esibito dal metodo.

Con il training si può verificare se i cluster prodotti contengono i campioni con le stesse etichette. Per funzionare è quindi importante che ci siano le etichette.

Reti Neurali

Quando si parla di reti neurali in parte è sbagliato dire che sono un'emulazione del cervello umano in quanto ad oggi non si hanno particolari e approfondite conoscenze sul funzionamento del cervello umano.

I **neuroni** (biologici) possiedono strutture arboree chiamate **dendriti** che ricevono segnali da altri neuroni mediante giunzioni dette sinapsi. Alcuni neuroni comunicano mediante poche sinapsi, altri ne posseggono migliaia. Il collegamento tra il nucleo del neurone e il dendrite si chiama **assone**. Tutti i processi che apprendiamo nel tempo sono frutto dell'interconnessione dei neuroni.

Una rete neurale computazionale trae ispirazione da questa logica e tende a costruire una struttura creata da nodi (neuroni) i quali tendono ad interconnettersi mediante gli archi. Di fronte ad un determinato compito alcuni neuroni saranno attivati in quanto sono quelli con quel determinato compito (modelli di apprendimento supervisionato). Non per forza questa attivazione dei neuroni viene vista come un insieme disgiunto, la combinazione dei neuroni permette al modello di apprendere.

Le reti neurali sono "learning machine" generali, adattive, non lineari, distribuite, costituite da molti **processori elementari** (PE).

I segnali che fluiscono attraverso le connessioni sono scalati con opportuni parametri modificabili detti **pesi** (w_{ij}).

Il singolo PE somma tutti questi contributi e produce un'uscita che è una funzione non lineare di tale somma.

Un **neurone** ha una struttura costituita come segue, ovvero riceve in input una serie di parametri $X_1, \ldots, X_D$ queste informazioni confluiscono verso il neurone attraverso delle

connessioni pesate $W_1, \ldots, W_D$. L'informazione arriva in input in modo opportunamente pesato al neurone già filtrato.

Il neurone effettua una combinazione lineare di questi valori i quali vengono dati in input ad una funzione di attivazione la quale determina se a fronte di questo input il neurone si deve attivare o meno, idealmente se il neurone è bravo si deve attivare quando la classe del dato corrisponde alla classe sulla quale il neurone si è attivato.

Le reti neurali sono un **modello adattivo**, hanno una serie di parametri in input e a partire dei dati modificano questi parametri ovvero dopo aver elaborato la risposta stimano la percentuale di errore e si addestrano nuovamente affinché l'errore diminuisca o idealmente arrivi a 0.

Il primo modello matematico di neurone fu teorizzato da **McCulloch e Pitts**:

- Si ha un certo input $a_0, a_1, \ldots, a_j, \ldots$ ogni input ha ad esso assegnato un peso $W_{0,i}, \ldots, W_{j,i}$ il quale indica il passaggio dallo stato j verso lo stato i.
- Il neurone i riceve in input una somma pesata di tutti gli input più l'input a_0=-1 che si chiama **Bias Weight** il quale è responsabile dell'individuazione di quella soglia di separazione che fa sì che se il valore (la somma pesata) raggiunge un certo valore allora la funzione di attivazione si attiverà altrimenti no.

Se non considerassi a_0, ma solo quei parametri, mi mancherebbe il contributo per spostarmi in modo "netto" da una classe piuttosto che un'altra. Il Bias è quel parametro in più con un opportuno peso e sarà appreso e fa sì che la risposta del neurone si sposta da una parte all'altra.

$$g(in_i) = \begin{cases} 1 & in_i >= 0 \\ -1 & in_i < 0 \end{cases}$$

Laddove $a_i = g(in_i) = g(\sum_{j=0}^{n} W_{j,i} a_j)$

Questa funzione prende il nome di **funzione segno**, funzione lineare a tratti che ha valore 1 se l'input è >= 0, -1 altrimenti.

Questo modello matematico elementare può essere utilizzato per implementare porte logiche AND, OR e NOT.

Tuttavia non tutto si può fare con delle semplici funzioni segno, ma si possono utilizzare altre funzioni non lineari, come la **tangente iperbolica** (può assumere anche valori negativi), la **funzione logistica** (valori tra 0 e 1, chiamata anche Sigmund) e la **funzione ReLU** (Rectified Linear Unit) questa ultima è il max tra 0 e il valore dell'input.

L'aspetto negativo della tangente iperbolica è che se l'input è enormemente negativo il neurone diventerà talmente negativo che se anche gli mostro un numero grande di campioni quel neurone non si attiverà mai più, questo crollo/depodenziamento dei neuroni potrebbe portare il nostro modello a basarsi su quei pochi neuroni che sono rimasti. È da evitare che la funzione possa prendere in input valori enormemente positivi o negativi per evitare che si

specializzi in modo eccessivo, di conseguenza vengono perse anche le informazioni perché se si fa fluire l'informazione verso neuroni disattivati, non faccio altro che perdere quell'informazione e il neurone "bruciato" non propagherà in avanti la mia informazione.

La ReLU propaga l'informazione così com'è senza il contributo di una sfumatura cosa che fa invece la funzione logistica. Nell'interconnessione di migliaia di neuroni ha mostrato una maggiore efficacia.

Oggi la funzione più diffusa è la **leakyReLU** che assume un valore prossimo allo zero. Con questa funzione il neurone continua ad essere in vita dando un piccolo contributo anziché essere "morto".

Il **percettrone** è una rete neurale a singolo strato alimentata in avanti in cui tutti i nodi di input sono collegati a tutti i nodi di output (grafo bipartito completo).

Il suo output è calcolato semplicemente come funzione dei suoi input pesati.

È un classificatore binario ed è capace di apprendere funzioni semplici. Si fa la supposizione che esiste una separazione lineare dei campioni del dominio di interesse.

Con il percettrone si può realizzare la **funzione di maggioranza** la quale vale 1 se più della metà dei suoi input vale 1. Tutti i pesi $W_j = 1 \forall j = 1,...,n$ e il bias $W_0 = \frac{n}{2}$.

Un albero di decisione con la medesima funzione richiederebbe la memorizzazione di $O(2^n)$ nodi.

L'output di un percettrone a soglia restituisce 1 se e solo se la somma pesata dei suoi input (incluso il bias) è positiva: $\sum_{j=0}^{n} W_j x_j > 0$.

$W \cdot x > 0$ definisce **l'iperpiano** di separazione nello spazio degli input che fa in modo che il percettrone restituisca 0 se e solo se i campioni di input sono da una parte specifica di tale iperpiano.

Non è altro che un **separatore lineare**.

Un percettrone è in grado di risolvere problemi che sono linearmente separabili.

Lo studio delle reti neurali ebbe un enorme crollo quando ci si rese conto che il percettrone non fosse in grado di computare lo XOR e quindi che fosse solo un separatore lineare.

Il percettrone è in grado di apprendere solo se i suoi pesi sono scelti in modo tale che il campione dato in input venga proiettato dalla parte corretta dell'iperpiano di separazione. L'apprendimento dei pesi è formulato come una ricerca di ottimizzazione nello spazio dei pesi.

La misura dell'errore usata è la **somma dei quadrati degli errori**.

Dato un campione x per il quale il percettrone restituisce $h_W(x)$ e la cui vera etichetta è y, il quadrato dell'errore sarà: $E = \frac{1}{2} Err^2 = \frac{1}{2}(y - h_W(x))^2$.

Si può utilizzare il metodo della **discesa del gradiente** per ridurre tale quadrato dell'errore calcolando le derivate parziali di E rispetto ad ogni suo peso. Devo verificare quale peso è corretto e quale no e per ognuno di essi eseguo le opportune modifiche.

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}(y - g(\sum_{j=0}^{n} W_j x_j)) = -Err \times g'(in) \times x_j$$

Il concetto che sta alla base della discesa del gradiente, il gradiente in un punto di una funzione è quella retta tangente alla funzione in quel punto e seguire il verso discendente del gradiente significa andare nel verso in cui la funzione minimizza.

La scelta del punto di partenza determina il fatto che si può ricadere in un minimo locale.

Spesso accade che il **tasso di apprendimento**, ovvero il "salto" che faccio da un punto ad un alto, viene scelto in una quantità molto piccola diventando così controproducente. Se il tasso di apprendimento invece è troppo grande rischio di non riuscire a raggiungere il minimo. L'unico vantaggio di avere un learning rate alto mi permette di sfuggire a minimi locali "fuoriuscendo" dalla conca.

Le dimensioni del tasso di apprendimento sono difficili da scegliere, solitamente è preferibile che il tasso di apprendimento sia alto nelle fasi iniziali e vada via via sempre di più a diminuire.

Rosenblatt ha sviluppato l'algoritmo a discesa del gradiente, nel quale vogliamo ridurre E, si aggiornano i pesi nel seguente modo: $W_j = W_j + \alpha \times Err \times g'(in) \times x_j$.

α indica il tasso di apprendimento che determina se incrementare o decrementare il valore del peso. Se $Err = y - h_W(x)$ è positivo, l'output è troppo piccolo e quindi i pesi positivi vengono aumentati e quelli negativi diminuiti.

Una rete neurale viene addestrata fornendo un campione alla volta, (**iterazione**).

I pesi vengono aggiornati ad ogni iterazione modificando lievemente i pesi del modello per ridurre l'errore.

Quando tutto il dataset di addestramento è stato dato in input alla rete, si chiude **un'epoca**.

Il ciclo di addestramento non si chiude alla fine di una epoca, ma spesso i campioni vengono riproposti alla rete in un numero variabile di epoche, effetto collaterale se si eccede con le epoche è quello di overfitting.

Il processo termina al raggiungimento di un criterio di arresto, ovvero quando i pesi si modificano di poco.

Un altro condizione di terminazione è quella dell'**early stopping** che consiste nell'interruzione dell'addestramento prima che l'errore raggiunga un plateau, considerando così i pesi in modo tale che l'errore sia basso, ma evitando l'overfitting.

Anziché passare un campione per volta è preferibile addestrare il modello in **batch** (a lotti): invece di aggiornare i pesi ad ogni campioni si fanno fluire nella rete un numero di campioni e si aggiornano i pesi solo al termine del lotto, solitamente potenze del 2.

Infine vi è il **gradiente stocastico**: solo alcuni campioni del training vengono forniti alla rete presi in modo casuale.

Le **reti neurali multistrato** (**MLP**) hanno segnato il momento in cui nel 1985 si è pensato di considerare le reti neurali anche per problemi non linearmente separabili, basandosi sull'idea della combinazione di più funzioni linearmente separabili potesse produrre delle superfici/iperpiani enormemente complessi capaci di separare anche problemi non linearmente separabili.

Una MLP è una rete alimentata in avanti che non connette gli input direttamente ai suoi output, ma è dotata di un numero variabile di livelli intermedi, chiamati livelli nascosti che sono interconnessioni di percettroni. Gli algoritmi di apprendimento per le MLP sono del tutto simili a quelli del percettrone se non per un aspetto fondamentale ovvero l'apprendimento dei pesi dei livelli nascosti. La correzione dei valori dei pesi del livello di output è semplice perché conosciamo il valore reale y, per addestrare anche i pesi dei livelli nascosti si retropropaga l'errore con l'**algoritmo di backpropagation**.

La MLP inoltre è in grado di risolvere anche un problema multiclasse perché può avere un output layer molto grande dove ad esempio si attiva l'i-esimo neurone quando l'i-esima classe è vera, la cosiddetta one-hot encode (dove solo uno dei neuroni di output vale 1).

La funzione di minimizzazione dell'errore non dipenderà più solo dal peso che collega il layer di input con il layer di output come nel percettrone, ma dipenderà dall'hidden layer precedente fino ad arrivare all'input layer.

Indichiamo per prima cosa $\triangle_i = Err_i \times g'(in_i)$, dove Err_i indica la i-esima componente dell'errore $y - h_W(x)$ e si individua la funzione di errore $\triangle_i$.

La regola di aggiornamento dei pesi diventa: $W_{j,i} = W_{j,i} + \alpha \times a_j \times \triangle_i$

Si ritiene che ciascun nodo j del livello nascosto sia "responsabile" in parte dell'errore $\triangle_i$ in ognuno dei nodi di output a cui è collegato.

I valori $\triangle_i$ sono suddivisi in base alla forza delle connessioni tra nodo nascosto e nodo di output e passati all'indietro per fornire i valori $\triangle_j$ allo stato nascosto.

La regola di propagazione dei valori diventa: $\triangle_j = g'(in_j) \sum_i W_{j,i} \triangle_i$

L'aggiornamento dello strato di input è del tutto identico a quello di output: $W_{k,j} = W_{k,j} + \alpha \times a_k \times \triangle_j$

Se un nodo contribuisce molto all'errore allora il suo peso sarà abbassato, così sarà possibile attivare o disattivare i neuroni della rete che stanno lavorando bene oppure male.

Il valore di un output layer lo si può vedere come valore percentuale di un dato appartenente a quella classe.