Pattern Recognition

A.A. 2009/2010

Contents

1 Definizioni preliminari

Definizione 1.1. Algoritmo di classificazione

Detto F lo spazio delle osservazioni e detto C l'insieme delle classi, un *algoritmo di classificazione* è una funzione

$$f : F \to C$$

Definizione 1.2. Matrice di confusione

E' una matrice $k \times k$, dove m_{ij} indica il numero di elementi della classe $i-esima$ classificati nella classe $j-esima$.

Osservazione 1.1. a questo concetto si lega il discorso su falsi positivi e falsi negativi.

Definizione 1.3. Il caso ideale è una matrice del tipo

$$\begin{bmatrix} 1/k & & 0 \\ & \ddots & \\ 0 & & 1/k \end{bmatrix}$$

Definizione 1.4. Overfitting

E' un fenomeno che accade quando un classificatore non commette errori su un particolare dataset, ma non è generalizzabile, dove con *generalizzabilità* intendiamo la correttezza rispetto alla totalità dei campioni.

Definizione 1.5. Control set

Insieme finito di osservazioni per le quali è nota la classificazione.

Definizione 1.6. Training set

Insieme finito utilizzato per calcolare le regole di classificazione, cioè per "allenare" la macchina a classificare.

Definizione 1.7. Strategia *leave-one-out*

Dato un CS C, lo suddivido in due sottoinsiemi random C_1, C_2 tali che

$$C \begin{cases} C_1 = C - \{x\} \\ C_2 = \{x\} \end{cases}$$

dove C_1 sarà utilizzato come TS e C_2 sarà utilizzato come CS.

Ripetendo per ogni $x \in C$, otteniamo una stima accettabile della generalizzabilità dello schema di classificazione.

Distribuzione delle frequenze

E' interessante misurare come si distribuiscono le feature sulla popolazione, valutandone la frequenza.

$$frequenza = \frac{\#occorrenze}{totale}$$

Questo approccio ci consente di lavorare anche su popolazioni numericamente differenti.

Data una popolazione S divisa in k classi $C_1, \ldots, C_k$ avremo che

$$\forall C_i \quad \Rightarrow f_i = \frac{|C_i|}{|S|} \qquad 0 \leq f_i \leq 1$$

$$\sum_{i=1}^{k} f_i = 1$$

Date due classi $C_i, C_j \ : \ C_i \cap C_j = \emptyset$, definiamo $C_{ij} = C_i \cup C_j$.

Poichè l'intersezione è vuota $\Rightarrow f_{ij} = f_i + f_j$.

In generale dati $S_i, S_j \subseteq S \ \Rightarrow f_{ij} \leq f_i + f_j$.

Questa è detta *proprietà di sub-additività*.

Se conosciamo $|S_i \cap S_j| = s, \ f_{ij} = f_i + f_j - \frac{s}{|S|}$.

2 Probabilità

Definizione 2.1. PDF - Probability Density Function

E' una funzione di densità continua (e quindi integrabile) che gode di due proprietà:

1. $f(x) \geq 0 \quad \forall x \in Dom(f)$;
2. $\int_{-\infty}^{+\infty} f(x)\,dx = 1$.

Osservazione 2.1. si ottiene un paradosso matematico, visto che l'integrale su un valore fissato vale 0.

Definizione 2.2. Funzione probabilità

Diremo *probabilità* su uno spazio di eventi S la funzione

$$P : \mathcal{P}(S) \to [0,1]$$

dove con $\mathcal{P}(S)$ indichiamo l'insieme delle parti di S.

Questa funzione gode di alcune proprietà:

- $0 \leq P(X) \leq 1, \qquad \forall X \subseteq S$;
- $P(S) = 1$;
- $P(\emptyset) = 0$;
- $P(X \cup Y) = P(X) + P(Y) - P(X \cap Y)$.

Definizione 2.3. Probabilità a priori

Assegnata una feature o una classe C, diremo *probabilità a priori* di C dentro la popolazione il rapporto tra la cardinalità di C e la cardinalità di S e scriveremo

$$P(C) = \frac{|C|}{|S|}$$

Definizione 2.4. JPF - Joint Probability Function

La funzione di *distribuzione di probabilità congiunta (JPF)* definisce la probabilità che, dati due eventi X e Y, essi si verifichino contemporaneamente. Nell'ambito della PR, ci si riferisce all'osservazione congiunta di due o più feature.

Date due feature A e B, la JPF si esprime come

$$P(A,B) = P(A|B)\,P(A) = P(B|A)\,P(B)$$

La conoscenza della JPF di un fenomeno ci consente di descrivere completamente le probabilità a priori e congiunte per qualunque sottoinsieme di feature.

Definizione 2.5. Marginalizzazione

Sia S un insieme di osservazioni, ciascuna rappresentata da un vettore $(x_1, \ldots, x_k)$ e sia $f(x_1, \ldots, x_k)$ la JPF del fenomeno S.

Allora

$$P(x_j = \overline{x_j}) = \sum_{\forall x_1}\sum_{\forall x_2} \cdots \sum_{\forall x_{j-1}}\sum_{\forall x_{j+1}} \cdots \sum_{\forall x_k} f(x_1 \ldots \overline{x_j} \ldots x_k)$$

In sostanza ci stiamo concentrando sulla feature $\overline{x_j}$, mettendo **al margine** i valori delle feature $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots x_k$.

Questa operazione è detta *marginalizzazione rispetto a* x_j e le variabili marginalizzate si dicono **saturate**.

Definizione 2.6. Probabilità condizionale

Dati due eventi A e B, la probabilità condizionata di A dato B è la probabilità che si verifichi l'evento A assunto che si è verificato l'evento B. Nel nostro campo, date delle feature $x_1, \ldots, x_k$, possiamo esprimerla come

$$P\left(x_j = \overline{x_j} | x_1 = \overline{x_1}, \ldots, x_k = \overline{x_k}\right)$$

cioè la probabilità di osservare $\overline{x_j}$ dato che valgono le condizioni $\overline{x_1}, \ldots, \overline{x_k}$.

In generale date delle feature $x_a, x_b, x_e, x_m, x_n, x_s$

$$P\left(x_a x_b x_e | x_m x_n x_s\right) = \frac{\sum\limits_{\substack{\text{sulle } x \\ \text{fissate}}} P\left(x_1 \ldots x_k\right)}{P\left(x_m x_n x_s\right)}$$

Definizione 2.7. JPF continua

Caso unidimensionale

Consideriamo un set di osservazioni S composto da N elementi $x_1, \ldots, x_N$.

In questo caso

$$JPF = P\left(x\right) = f\left(x\right) = \int_x^x f(x)dx = 0$$

Consideriamo quindi un ϵ e valutiamo la JPF nell'intervallo $[x - \epsilon, x + \epsilon]$. L'integrale sarà quindi

$$\int_{x-\epsilon}^{x+\epsilon} f(x)dx$$

Caso multidimensionale

Supponiamo di avere un set di osservazioni S composto da N coppie $\left(x_1, y_1\right), \ldots, \left(x_N, y_N\right)$. Anche in questo caso, ovviamente, l'integrale sarà uguale a 0.

Consideriamo quindi ϵ e δ e valutiamo la JPF nell'intervallo bidimensionale $\left[x - \epsilon, x + \epsilon\right], \left[y - \delta, y + \delta\right]$ ottenendo

$$P\left(\left[x - \epsilon, x + \epsilon\right], \left[y - \delta, y + \delta\right]\right) = \int_{x-\epsilon}^{x+\epsilon} \int_{y-\delta}^{y+\delta} f\left(x, y\right) dx$$

Definizione 2.8. Fattorizzazione della probabilità congiunta

Possiamo quindi dare un'altra definizione di probabilità condizionale

$$P\left(w | x\right) = \frac{P\left(w, x\right)}{P\left(x\right)}$$

da cui otteniamo la formula di fattorizzazione della probabilità congiunta

$$P\left(w, x\right) = P\left(w | x\right) P\left(x\right)$$

Definizione 2.9. Indipendenza

Proviamo a rappresentare in un grafico due feature qualunque, cercando di ottenere delle informazioni sulla correlazione tra di esse.

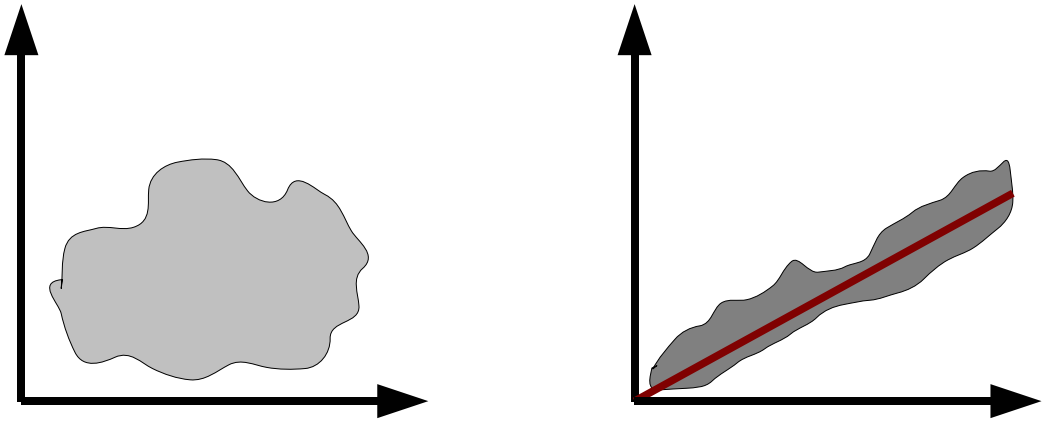


Table 1: Dipendenza e indipendenza

Nella prima figura, notiamo che non c'è un "ordine", ma le feature sono distribuite in maniera più o meno uniforme nel piano. Nella seconda invece notiamo che i valori si dispongono secondo un criterio abbastanza regolare. La dipendenza tra variabili consente di eliminarne una, riducendo la complessità del problema.

Più formalmente, date due feature A e B, diremo che esse sono indipendenti se

$$P(A|B) = P(A)$$

Inoltre vediamo che

$$P(A,B) = P(A|B)\,P(B) = P(A)\,P(B)$$

Quest'ultima formula costituisce un'altra definizione di indipendenza condizionale.

Definizione 2.10. Fattorizzazione canonica della JPF

Supponiamo di analizzare un fenomeno descritto da 4 feature a, b, c, d. Allora

$$\begin{aligned} P(a,b,c,d) &= P(a|b,c,d)\,P(b,c,d) = \\ &= P(a|b,c,d)\,P(b|c,d)\,P(c,d) = \\ &= P(a|b,c,d)\,P(b|c,d)\,P(c|d)\,P(d) \end{aligned}$$

Definizione 2.11. Formula di Bayes

Consideriamo la JPF di due variabili

$$\begin{aligned} P(a,b) &= P(a|b)\,P(b) \\ P(a,b) &= P(b|a)\,P(a) \end{aligned}$$

da cui posso scrivere

$$P(a|b)\,P(b) = P(b|a)\,P(a)$$

e quindi

$$P(a|b) = \frac{P(b|a)\,P(a)}{P(b)}$$

In PR scriviamo

$$P(w|x) = \frac{P(w|x)\,P(x)}{P(w)}$$

dove w è la classe (non nota) e x è la feature osservata.

Diremo

- $P(w|x)$ **probabilità a posteriori;**
- $P(w)$ **probabilità a priori;**
- $P(x)$ **evidenza;**
- $P(x|w)$ **verisimiglianza o LIKELYHOOD.**

2.1 Algoritmo di classificazione MAP

Definizione 2.12. Algoritmo MAP

Sia data un'osservazione $\overrightarrow{x}$. Devo assegnare tale osservazione ad una delle classi $w_1, \ldots, w_k$.

Allora

1. Calcolo $P(w_i|\overrightarrow{x})$, $1 \leq i \leq k$;
2. Assegno $\overrightarrow{x}$ alla classe w_i che ha la massima probabilità a posteriori.

Ovviamente è sempre presente la possibilità di commettere degli errori.

Definizione 2.13. Matrice delle perdite

Matrice L in cui l'elemento l_{ij} è il costo in cui si incorre prendendo la decisione $i-esima$ su un elemento della classe $j-esima$. Generalmente #decisioni = #classi, quindi la perdita è dovuto alla **misclassification**.

Nel caso in cui #decisioni = #classi, parliamo di *matrice delle perdite uniforme*. In questo caso

$$l_{ij=}\begin{cases}1 & \text{se c'è un errore}\\ 0 & \text{se non c'è un errore}\end{cases}$$

Supponiamo di avere un'osservazione $\overrightarrow{x}$ e sia L la matrice delle perdite.

Chiamo *rischio della decisione* $i-esima$ il valore

$$\begin{aligned} R(i,\overrightarrow{x}) &= l_{i1}P(1|\overrightarrow{x}) + l_{i2}P(2|\overrightarrow{x}) + \ldots + l_{ij}P(j|\overrightarrow{x}) = \\ &= \sum_{j=1}^{k} l_{ij}P(j|\overrightarrow{x}) \end{aligned}$$

Il valore ottenuto con questa formula è il **costo medio atteso** della decisione (cost expectancy) e si indica con E.

Definizione 2.14. Classificatore MAP di minimo rischio

Vogliamo minimizzare R al variare dello schema decisionale del classificatore, cioè al variare di $F : \overrightarrow{x} \rightarrow decisione$, dove

$$R = \int_{\forall \overrightarrow{x}} \sum_{\forall i} R(i,\overrightarrow{x})\, P(\overrightarrow{x})\, dx$$

Poichè non possiamo "controllare" P, dobbiamo minimizzare il costo della decisione per ogni osservazione. Dobbiamo quindi scegliere l'azione h che minimizza $R(i,\overrightarrow{x})$.

3 Odds

Definizione 3.1. Odd

In statistica, con il termine *odd* si intende il rapporto tra la probabilità di un evento A e la probabilità dell'evento complementare $\neg A$

$$odd\,(A) = \frac{P\,(A)}{P\,(\neg A)} = \frac{P\,(A)}{1 - P\,(A)}$$

Il rapporto tra due odd è detto *odds ratio* ed entrambi i concetti vengono frequentemente usati nelle statistiche sanitarie.

Definizione 3.2. Logit

Il modello logistico descrive la crescita di una popolazione e si dimostra che è modellato dall'equazione

$$\frac{d}{dt}P\,(T) = P\,(T)\,(M - P\,(T))$$

detta *equazione logistica.*

Definiamo inoltre un'altra funzione, detta *log*it, che è il logaritmo naturale dell'odd

$$logit\,(A) = \ln\left(\frac{P\,(A)}{1 - P\,(A)}\right)$$

La funzione *logit* può essere scritta anche come combinazione lineare di variabili $\overrightarrow{x}$ e pesi $\overrightarrow{\beta}$

$$logit\,(A) = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + c$$

In questo caso la stima della probabilità avviene effettuando prima la stima dei parametri $\overrightarrow{\beta}$ con il metodo di massima verosimiglianza e quindi operando la trasformazione

$$P\,(A) = \frac{e^{x\beta}}{1 + e^{x\beta}}$$

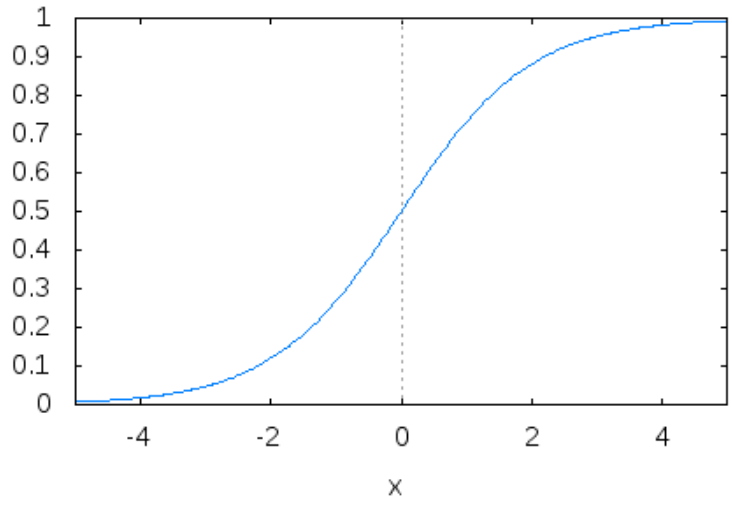
Definizione 3.3. Regressione logistica

Scegliendo $M = 1$ nell'equazione logistica e con la condizione che la variabile dipendente sia dicotomica (cioè binaria) si dimostra che l'equazione logistica ha come soluzione la funzione

$$f\,(x) = \frac{1}{1 - e^{-x}} = \frac{e^x}{1 + e^x}$$

Questa equazione modella un caso particolare di modello lineare, detto *regressione logistica.*

Il grafico della funzione $f(x)$ è una sigmoide definita in $]-\infty, +\infty[$ a valori in $[0, 1]$



Definizione 3.4. Generalizzazione del modello di regressione logistica

Per non avere limitazioni dovute alla linearità della funzione, possiamo sostituire la variabile indipendente x con una sua generica funzione $g(x)$; in questo caso scriveremo

$$F(x, g) = \frac{e^{g(x)}}{1 + e^{g(x)}}$$

E' interessante studiare il caso

$$g(x) = Ax + B$$

Al variare di B trasleremo la sigmoide lungo l'asse delle ascisse, mentre al variare di A varieremo lo "intervallo di transizione" intorno al valore medio.

La sigmoide è un'ottima approssimazione della funzione **soglia**, visto che pochi fenomeni presentano transizioni nette.

I parametri A e B ci consentono quindi di calibrare al meglio la curva per approssimare meglio la funzione soglia. Si tratta cioè di trovare A e B che minimizzano

$$\sum_i |odd(x_i - f_{A,B}(x_i))|$$

Proprio in questo consiste lo studio della regressione (lineare e non solo).

Definizione 3.5. Odds ratio

Consideriamo nuovamente l'odds ratio alla luce dell'equazione $g(x) = Ax + B$. Avremo

$$\begin{aligned} odd(x) &= e^{Ax+B} \\ odd(x+1) &= e^{A(x+1)+B} = e^A \cdot e^{Ax+B} \end{aligned}$$

Da cui l'odds ratio che vale

$$\frac{odd(x+1)}{odd(x)} = \frac{e^A \cdot e^{Ax+B}}{e^{Ax+B}} = e^A$$

Quindi tale rapporto **non dipende** da x.

Gli odds crescono quindi come una progressione geometrica di ragione e^A.

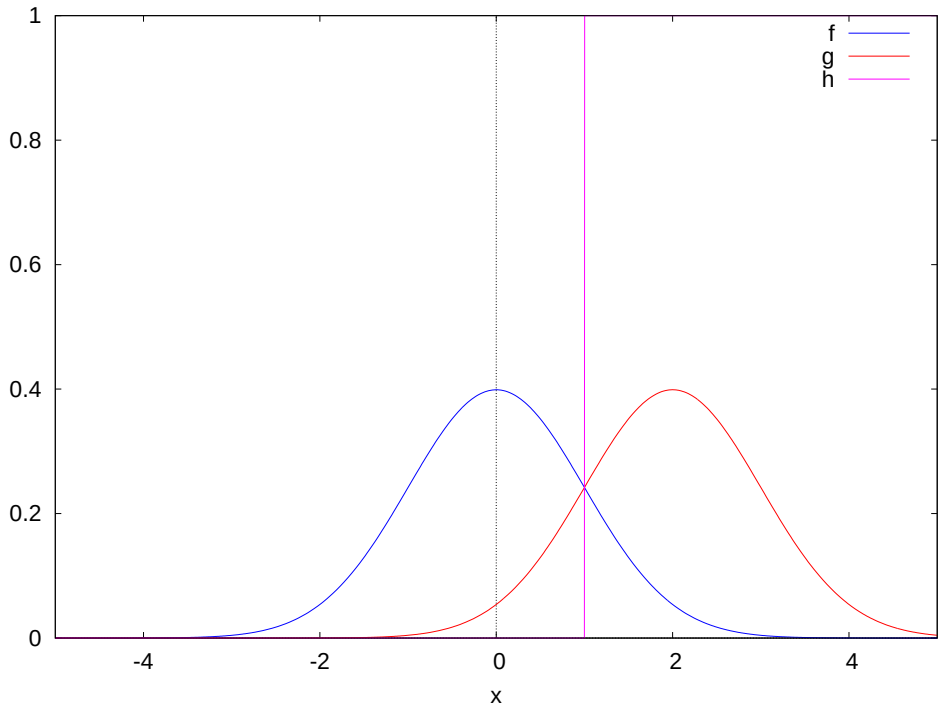
4 Curve ROC

Consideriamo un problema di classificazione binaria, dove cioè bisogna determinare se un elemento appartiene o meno ad una classe A. Consideriamo un'unica feature x, al cui variare assegnamo l'osservazione alla classe, tipicamente attraverso un schema di **soglia**.

Fissato cioè un valore s, ad esempio stabiliamo che

$$\begin{aligned} x < s &\Rightarrow A \\ x \geq s &\Rightarrow \neg A \end{aligned}$$

Tale classificazione può portare ad una indeterminatezza ineliminabile se le distribuzioni di probabilità che modellano le classi A e $\neg A$ si sovrappongono. Consideriamo ad esempio due distribuzioni gaussiane con media rispettivamente $\mu_1 = 0$ e $\mu_2 = 0$ e $\sigma_1 = \sigma_2 = 1$ (vedi figura sotto)



L'intersezione tra le due PDF è detta area dell'errore ineliminabile (AEI), e la retta verticale passante per $x = 1$ è il valore di soglia s. Al variare della soglia varia il numero di falsi, sia positivi che negativi. Il valore

$$\frac{|\mu_1 - \mu_2|}{2}$$

è detto **discriminabilità del sistema**.

Ovviamente l'obiettivo è minimizzare l'AEI, o, equivalentemente, massimizzare la discriminabilità.

Possiamo riassumere il tutto in una tabella simile alla *matrice di confusione*

	$-$	$+$
$-$	TN	FN
$+$	FP	TP

dove

- $+$ indica l'appartenenza alla classe A;

- $-$ indica la non appartenenza alla classe A;
- le righe sono le decisioni prese dal classificatore;
- le colonne sono le reali appartenenze delle osservazioni alla classe A;
- TN è il numero di "veri negativi";
- FN è il numero di "falsi negativi";
- TP è il numero di "veri positivi";
- FP è il numero di "falsi positivi".

Indichiamo con

- $N = TN + FP$ il numero di negativi $(-)$ nella realtà;
- $P = TP + FN$ il numero di positivi $(+)$ nella realtà.

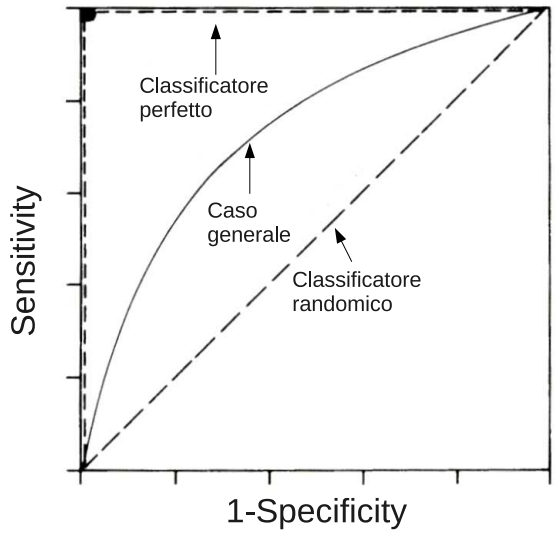
Generalmente, la matrice viene **normalizzata per colonne**.

Definiamo adesso alcune quantità notevoli:

- `precision` $= \frac{TP}{TP+FP}$;
- `sensitivity` (`recall`) $= \frac{TP}{TP+FN} = \frac{TP}{P}$ nella matrice normalizzata;
- `accuracy` $= \frac{TN+TP}{P+N}$;
- `specificity` $= \frac{TN}{TN+FP} = \frac{TN}{N}$ nella matrice normalizzata.

Le curve ROC (Receiver Operator Characteristic) consentono di interpretare graficamente il rapporto tra `hit` e falsi allarmi.

Lungo i due assi si rappresentano la `sensitivity` e la quantità $1 -$ `specificity`.



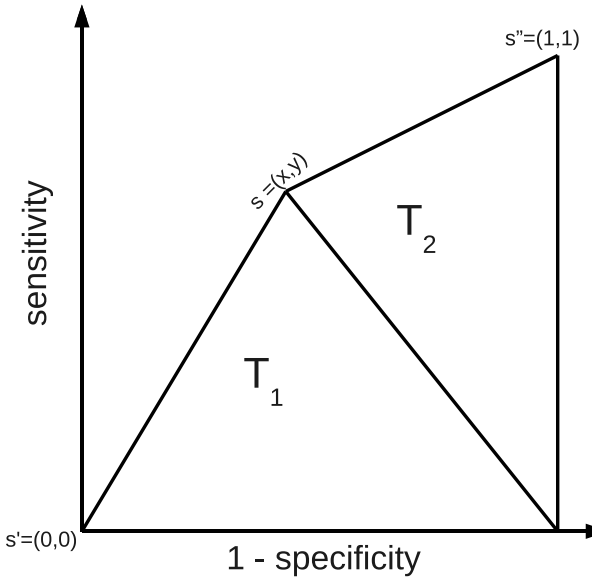
Il test consente di discernere tra A e $\neg A$ semplicemente analizzando l'area sottesa dalla curva, che fornisce la probabilità che il risultato di un test effettuato su un campione estratto a caso dal gruppo $\neg A$ sia superiore al risultato del test su un campione estratto a caso dal gruppo A.

Generalmente, le curve ROC passano per i punti $(0,0)$ e $(1,1)$ avendo come casi limite le curve che nel grafico sono indicate come classificatore perfetto e classificatore randomico.

Nel primo caso, la curva è in realtà una spezzata che passa per i punti $(0,0)$, $(0,1)$ e $(1,1)$, e l'area sottesa da essa vale quindi 1. Ciò significa che il classificatore riconosce correttamente tutti i campioni, ed è quindi perfetto.

Nel secondo caso, la curva ha un'inclinazione di $45°$ sull'asse orizzontale, e l'area sottesa vale $\frac{1}{2}$. Questo significa che la probabilità di un corretto riconoscimento è totalmente casuale (come lanciare una monetina) e che le distribuzioni di probabilità delle classi A e $\neg A$ si sovrappongono perfettamente.

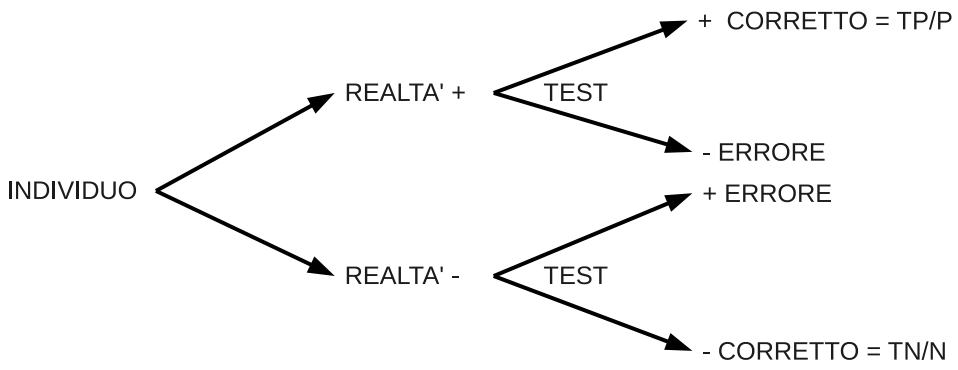
Nel caso generale avremo



Ci interessa l'area del quadrilatero, che è uguale alla somma delle aree dei triangoli T_1 e T_2

$$A = \frac{1}{2}\left(\frac{TP}{TP+FN}\right) + \frac{1}{2}\left(\frac{TN}{TN+FP}\right)$$

L'interpretazione di queste quantità nella nostra popolazione normalizzata è la seguente: consideriamo un individuo ed effettuiamo il test:



Per cui l'area A è uguale alla probabilità che il classificatore effettui una corretta classificazione.

5 Analisi delle componenti

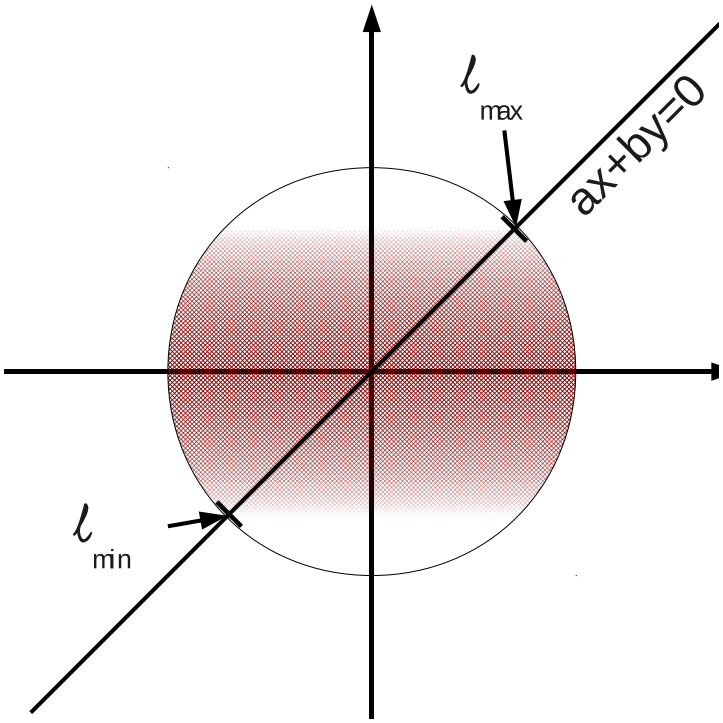
5.1 PCA - Principal Component Analysis

PCA è un metodo per (ri)scrivere i dati, non è un vero e proprio metodo di PR, in quanto non effettua classificazione. E' un metodo non supervisionato (cioè non conosciamo/analizziamo le classi dei dati in input, che vengono solo riscritti); sostanzialmente, è un *cambiamento della base nello spazio vettoriale nel quale sono stati raccolti i dati.*

Un'operazione preliminare molto utile è sottrarre la media dei valori dai dati in modo da centrarli nell'origine del sistema di riferimento; in questo modo spostiamo il baricentro delle distribuzioni per studiare la dispersione dei dati lasciandoli però "puliti".

L'obiettivo che ci poniamo è quello di trovare la direzione di massima dispersione dei dati, cioè la retta sulla quale la proiezione dei dati ha minore densità.

Poniamoci nel caso bidimensionale, come rappresentato in figura;



possiamo tracciare una retta passante per l'origine $ax + by = 0$; introducendo il versore $(u, v) : u^2 + v^2 = 1$ possiamo riscrivere l'equazione della retta $ax + by = 0$ come segue:

$$(x, y) = \lambda\,(u, v) = \lambda\,(b, -a) \qquad \Longrightarrow \qquad vx - uy = 0$$

Adesso proiettiamo i dati sulla retta e valutiamo la distanza di tale proiezione dall'origine:

$$d\,(\text{proiezione di } \ (x, y) \ \text{ sulla retta di versore } \ (u, v)\,,\ \text{origine}) = ux + vy$$

In questo modo stiamo riducendo la dimensione del problema da

$$\begin{array}{rcl} \mathbb{R}^2 & \longrightarrow & \mathbb{R} \\ (x, y) & \longrightarrow & ux + vy = l \end{array}$$

Il valore l apparterrà all'intervallo $[l_{min}, l_{max}]$ che rappresenta il segmento della retta che interseca la distribuzione di dati. Ovviamente, avremo che $l_{min} \leq 0 \leq l_{max}$; in ogni caso non è detto che (come nel caso della figura) $|l_{min}|$ e $|l_{max}|$ siano uguali. Al variare del coefficiente angolare (o equivalentemente dell'angolo θ, se usiamo le coordinate polari) varierà l'ampiezza dell'intervallo. Proprio questo ci consente di individuare la direzione di massima dispersione; infatti, se $\forall\theta \ |l_{min}| = |l_{max}|$ la lunghezza del segmento sarebbe costante, e quindi tutte le direzioni sarebbero equivalenti.

Per trovare quindi la direzione di massima dispersione cominciamo con il calcolare la matrice di covarianza per la nostra popolazione

$$\Sigma = \begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix}$$

dove

$$c_{ij} = \frac{1}{N} \sum_{l=1}^{N} (x_{il} - x_{iMEDIO}) (x_{jl} - x_{jMEDIO})$$

Però, poichè abbiamo sottratto la media, avremo che $x_{iMEDIO} = 0 \, \forall i$ e quindi avremo

$$c_{ij} = \frac{1}{N} \sum_{l=1}^{N} (x_{il} x_{jl})$$

Noi vogliamo calcolare la direzione individuata dal vettore $(u_1, \ldots, u_n)$ che garantisce la massima variazione dei valori proiettati

$$y_i = u_1 x_{i1} + \ldots + u_n x_{in}$$

Si può dimostrare che - avendo sottratto la media - la dispersione si può calcolare come:

$$\underset{\{y_i\}_{i=1\ldots n}}{DISPERSIONE} = \frac{1}{N} \sum_i y_i^2$$

In definitiva, la quantità da massimizzare è

$$\frac{1}{N} \sum_i (u_1 x_{i1} + \ldots + u_n x_{in})^2$$

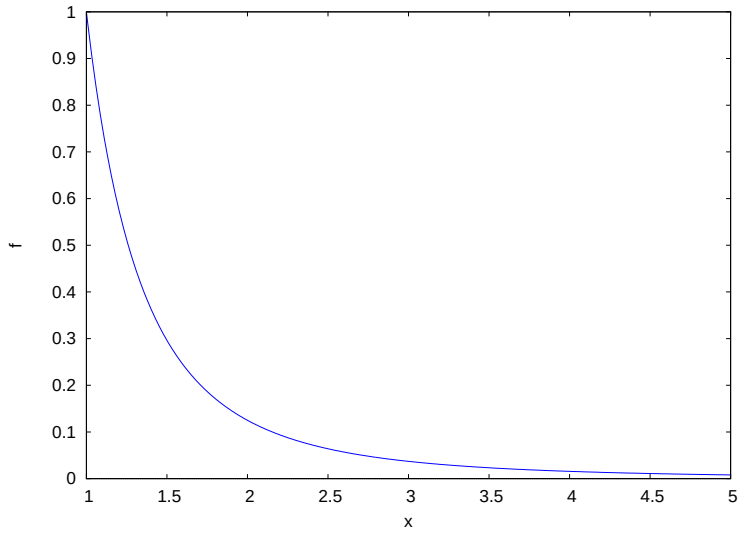
Questo è un problema di ottimizzazione risolvibile con il metodo dei moltiplicatori di Lagrange, ma siccome abbiamo anche computabilità e algoritmi trrre da fare saltiamo i passaggi ed andiamo alla soluzione, arrivando quindi alla seguente relazione:

$$\begin{aligned} \frac{2}{N} (u, v) \, \Sigma &= -2\lambda (u, v) \\ &\Downarrow \\ (u, v) \, \Sigma &= -2\lambda' (u, v) \\ &\Downarrow \text{ riscritto come} \\ \overrightarrow{v} \Sigma &= \lambda \overrightarrow{v} \end{aligned}$$

In sostanza, il problema è trovare gli autovettori della matrice di covarianza, in quanto essi rappresentano le direzioni (u, v); gli autovalori λ rappresentano invece l'ampiezza della dispersione della proiezione dei dati. Quindi, basterà trovare gli autovettori ed i corrispondenti autovalori di Σ ed ordinarli in maniera decrescente per trovare la direzione di massima dispersione dei dati.

5.1.1 Legge di potenza

Gli autovettori $|\lambda_i|$ ordinati in maniera decrescente seguono un profilo del tipo $x^{-\alpha}$.



Avremo anche che $\sum_i |\lambda_i| \simeq 1$. Possiamo dedurre che i primi autovettori, associati agli autovalori di maggior peso, descrivono la maggior parte del fenomeno, mentre gli ultimi, associati agli autovalori di minor peso, descrivono i dettagli.

5.1.2 PCA ed elaborazione di immagini

Le immagini possono essere rappresentate come matrici di valori. Detti p_i i pixel, possiamo descrivere un'immagine di dimensione $h \times k$ come

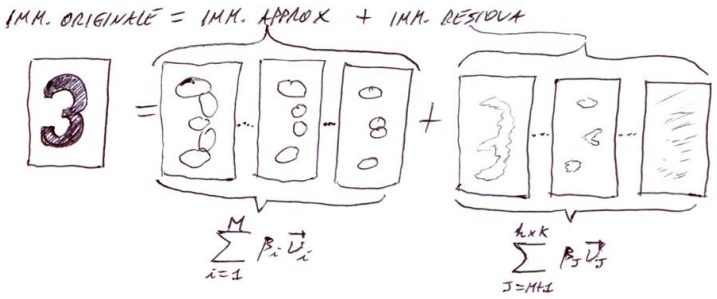
$$imm = \sum_{h \times k} \overrightarrow{\alpha_i} \overrightarrow{p_i}$$

Da qui possiamo ricavare gli autovettori $\overrightarrow{v_1}, \ldots, \overrightarrow{v_{h \times k}}$ e scrivere

$$imm = \Sigma \overrightarrow{\beta_i} \overrightarrow{v_i}$$

ossia possiamo scomporre le immagini in combinazioni di immagini che non sono "immagini canoniche" (nel senso di immagini derivanti dalla combinazione lineare delle basi canoniche dello spazio vettoriale di dimensione $h \times k$) ma sono comunque delle basi.

Il vantaggio è che anche in questo caso vale la legge di potenza: infatti si osserva che buona parte dell'immagine può essere ricostruita usando solo gli autovettori con autovalori di peso più elevato, che identificano le "regolarità" dell'immagine, mentre gli altri autovettori identificano le "variabilità" (rumore e dettagli) e possono anche essere scartati (ad esempio per la compressione dell'immagine).



5.2 Discriminant Analysis

La *Discriminant Analysis (DA)* è un tipo di analisi che aiuta a ridurre la dimensione dei dati in modo supervisionato.

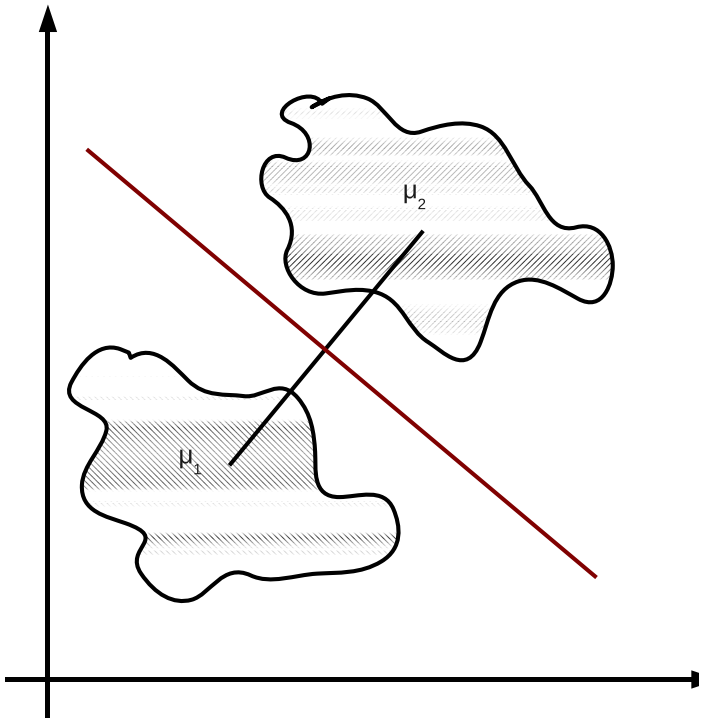
Punto centrale, comune a tutti i metodi di DA, è la **funzione lineare di score** (o funzione discriminante lineare), ossia una funzione in grado di assegnare un punteggio (z nella formula seguente) ad un oggetto in base ad una valutazione delle caratteristiche dell'oggetto.

$$(x_1, \ldots, x_n) \longrightarrow z = \alpha_1 x_1 + \ldots + \alpha_n x_n$$

Il problema è, quindi, trovare una "buona" funzione di score.

5.2.1 Analisi discriminante di Fisher

Date due distribuzioni gaussiane con σ comparabili, prendiamo i punti medi delle popolazioni, tracciamo una retta (il vettore delle differenze dei punti medi μ_1 e μ_2), prendiamo il punto medio di tale vettore e tracciamo la retta ortogonale passante per tale punto. Otteniamo così la retta discriminante, cioè la retta di separazione tra le due popolazioni.



Noi stiamo trattando la DA di Fisher con $k = 2$ popolazioni, ma ovviamente il discorso è generalizzabile, a patto che le distribuzioni siano

- approssimativamente normali (gaussiane);
- con σ simili tra loro.

La formula di Fisher nasce da due esigenze:

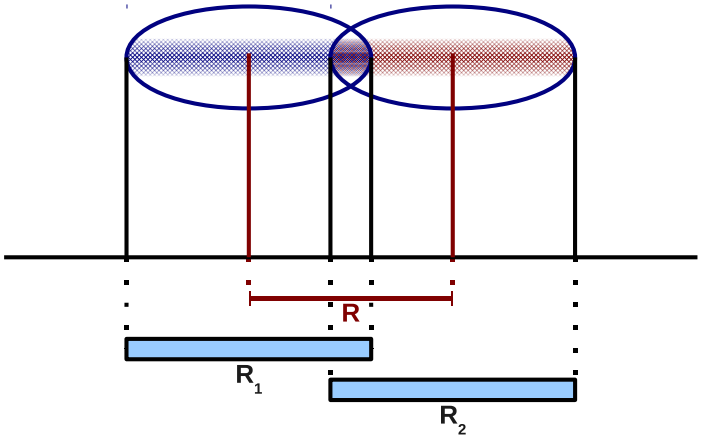
1. ESIGENZA DI AGGREGAZIONE (nella stessa classe):
   garantire che $\overrightarrow{x}$ appartenenti alla stessa classe abbiano punteggi (*score*) $z(\overrightarrow{x})$ simili tra loro.

2. ESIGENZA DI SEPARAZIONE (tra classi diverse):
   garantire che gli score $z(\overrightarrow{x})$ di elementi appartenenti a classi diverse siano molto diversi tra loro.

Partendo dalle due popolazioni A e B ($A \cup B = U$, $A \cap B = \emptyset$), possiamo identificare delle **matrici di dispersione *(scatter matrix)*** che rappresentano le due esigenze su citate:

- $D_{BETWEEN} \Longrightarrow$ rappresenta la distanza tra le medie proiettate delle classi, e ovviamente vogliamo che sia grande.
- $D_{WITHIN} \Longrightarrow$ è una misura della densità/dispersione di ciascuna classe sulla retta.

Graficamente:



$$\begin{aligned} s_B &= R \\ s_W &\propto R_1 + R_2 \end{aligned}$$

Se a queste relazioni aggiungiamo anche il vettore dei coefficienti $\overrightarrow{\alpha}$ che ci servono per determinare il nostro *score*:

$$z(\overrightarrow{x}) = \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n$$

allora possiamo introdurre le quantità

$$\begin{aligned} s_W &= \overrightarrow{\alpha} \cdot D_W \\ s_B &= \overrightarrow{\alpha} \cdot D_B \end{aligned}$$

che dipendono quindi dalla scelta di α.

Il problema della scelta di α è dunque un problema di ottimizzazione: vogliamo minimizzare $J(\alpha)$, definito come

$$J(\overrightarrow{\alpha}) = \frac{s_B(\overrightarrow{\alpha})}{s_W(\overrightarrow{\alpha})}$$

Questo problema è "simile" ad un problema della fisica noto come **potenziale generalizzato di Rayleigh**. Fisher formulò quindi un teorema che afferma che

Teorema 5.1. Teorema di Fisher
Dato $J(\overrightarrow{\alpha})$ definito come sopra, il vettore di coefficienti $\overrightarrow{\alpha}$ che minimizza tale quantità è dato da

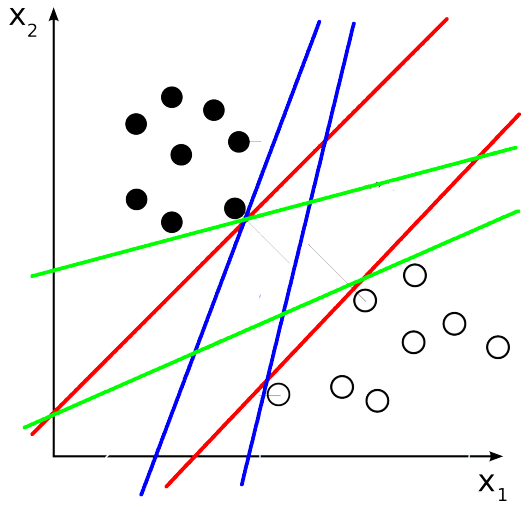
$$\overrightarrow{\alpha} = (\overrightarrow{m_A} - \overrightarrow{m_B}) \cdot D_W^{-1}$$

6 Support Vector Machine

Le macchine a vettori di supporto (SVM, dall'inglese Support Vector Machines), o macchine kernel, sono un insieme di metodi di apprendimento supervisionato per la regressione e la classificazione di pattern, sviluppati negli anni '90 da Vladimir Vapnik ed il suo team presso i laboratori Bell AT&T.

Appartengono alla famiglia dei classificatori lineari generalizzati e sono anche note come classificatori a massimo margine, poiché allo stesso tempo minimizzano l'errore empirico di classificazione e massimizzano il margine geometrico.

Supponiamo di avere due classi linearmente separabili

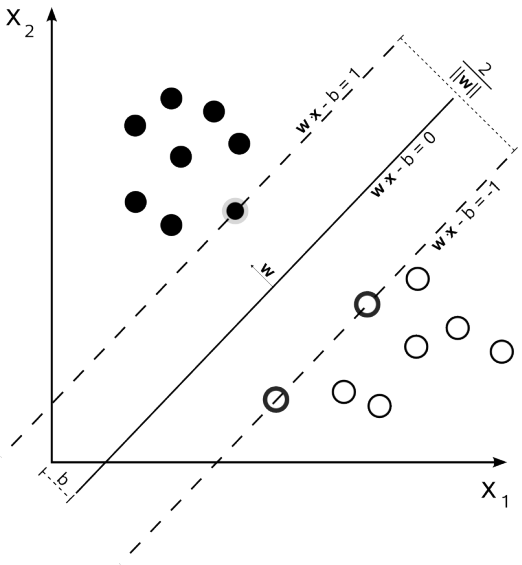


Vogliamo trovare un sistema di pesi $\overrightarrow{w}$ in grado di separare le due classi, cioè vogliamo trovare il corridoio più largo possibile tra tutti i corridoi che separano le classi.

Formalizziamo:

- classe 1: n_1 campioni $(x_{i1}, \ldots, x_{in_1})$, $i = 1 \ldots n_1$ etichettati con $l_1 = +1$;
- classe 2: n_2 campioni $(x_{j1}, \ldots, x_{jn_2})$, $j = 1 \ldots n_2$ etichettati con $l_2 = -1$.

Stabiliamo un sistema di pesi $\overrightarrow{w}$; nel piano otterremo



Il nostro obiettivo è quello di massimizzare la distanza tra le rette

$$\begin{aligned} \overrightarrow{w} \cdot \overrightarrow{x} - b &= +1 \\ \overrightarrow{w} \cdot \overrightarrow{x} - b &= 1 \end{aligned}$$

La distanza tra le due rette è $d = \frac{2}{|\overrightarrow{w}|}$. Il problema si riduce quindi a minimizzare il modulo del vettore $|\overrightarrow{w}|$.

Possiamo accorpare le equazioni delle due rette usando i coefficienti l_i e l_j: infatti, possiamo dire che

1. $\forall x_i \in \mathcal{C}_1 \implies \overrightarrow{w} \cdot \overrightarrow{x_i} - b \geq 1$;
2. $\forall x_j \in \mathcal{C}_2 \implies \overrightarrow{w} \cdot \overrightarrow{x_j} - b \leq -1$.

Da cui possiamo scrivere

$$l_i \left(\overrightarrow{w} \cdot \overrightarrow{x_i} - b\right) \geq 1 \qquad \forall x_i$$

Il problema è quindi un problema di ottimizzazione quadratica definito dalla formula

$$\min\left(\mathbf{w}, b\right) \; : \; \left[\frac{1}{2}\left\|\mathbf{w}\right\|^2 + \sum_{i=1}^{n_1+n_2} \alpha_i \left(l_i \left(\mathbf{w} \cdot \mathbf{x}_i - b\right) - 1\right)\right]$$

che ha come soluzione

$$\mathbf{w} = \sum_{i=1}^{n_1+n_2} \alpha_i l_i \mathbf{x}_i$$

I coefficienti α_i sono quasi tutti uguali a 0. Gli $\alpha_i : \alpha_i \neq 0$ sono detti *moltiplicatori di supporto* ed i punti x_i a cui sono associati sono detti punti di supporto, e giacciono sulla frontiera della striscia di piano. Quindi avremo

- $\overrightarrow{w} \cdot \overrightarrow{x_i} = b + 1$ per i punti di supporto della classe $\mathcal{C}_1$;
- $\overrightarrow{w} \cdot \overrightarrow{x_i} = b - 1$ per i punti di supporto della classe $\mathcal{C}_2$.

Dato un punto di supporto $\overrightarrow{x_i}$, possiamo calcolare $b = \overrightarrow{w} \cdot \overrightarrow{x_i} - 1$.

A causa dell'errore numerico introdotto dall'algoritmo, è più robusto effettuare una normalizzazione rispetto al numero totale dei vettori di supporto. Detta N_{SV} tale quantità, calcoliamo quindi

$$b = \frac{1}{N_{SV}} \sum_i \left(\overrightarrow{w} \cdot \overrightarrow{x_i} - 1\right)$$

6.1 SVM-soft

Il metodo **SVM** è vincolato alla lineare separabilità delle classi. Per classi *quasi* linearmente separabili (presenza di rumore) esiste una variante chiamata **SVM-soft**, nella quale si accetta di compiere qualche errore di classificazione.

Il formalismo è identico a quello definito dal metodo SVM, cioè il problema è sempre un problema di ottimizzazione quadratica basato sulla minimalizzazione dei pesi $\overrightarrow{w}$; i vincoli sono però rilassati con l'introduzione di variabili *dummy* dette **imprecisione**:

$$\begin{aligned} \forall \overrightarrow{x_i} \quad &\Rightarrow \quad \exists \xi_i : \overrightarrow{w} \cdot \overrightarrow{x_i} - b = 1 - \xi_i \\ \forall \overrightarrow{x_j} \quad &\Rightarrow \quad \exists \xi_j : \overrightarrow{w} \cdot \overrightarrow{x_j} - b = -1 + \xi_j \end{aligned}$$

Si introduce cioè una sorta di "soglia di tolleranza" sui vettori di supporto, che quindi non saranno più i punti esattamente sulla frontiera, ma i punti appartenenti ad un intorno della frontiera.

La quantità da minimalizzare diventa quindi

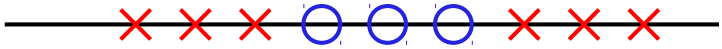
$$\frac{1}{2}\left\|\mathbf{w}\right\|^2 + \sum \xi_i$$

6.2 SVM su classi non linearmente separabili

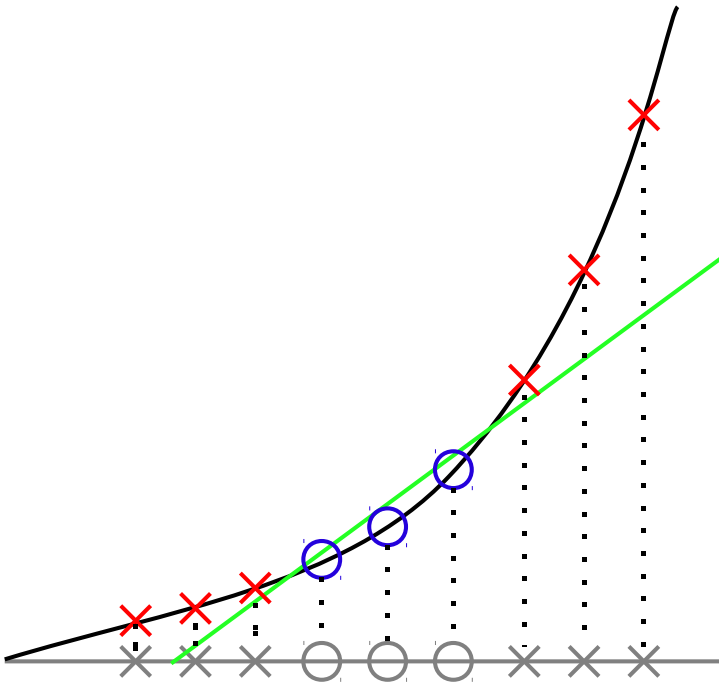
Spesso le classi non sono linearmente separabili, neanche con l'applicazione di trasformazioni lineari (traslazioni, rotazioni, cambiamenti di scala). L'idea quindi è quella di aumentare le dimensioni dei dati, facendoli "esplodere" su una dimensione più ampia.

Caso unidimensionale

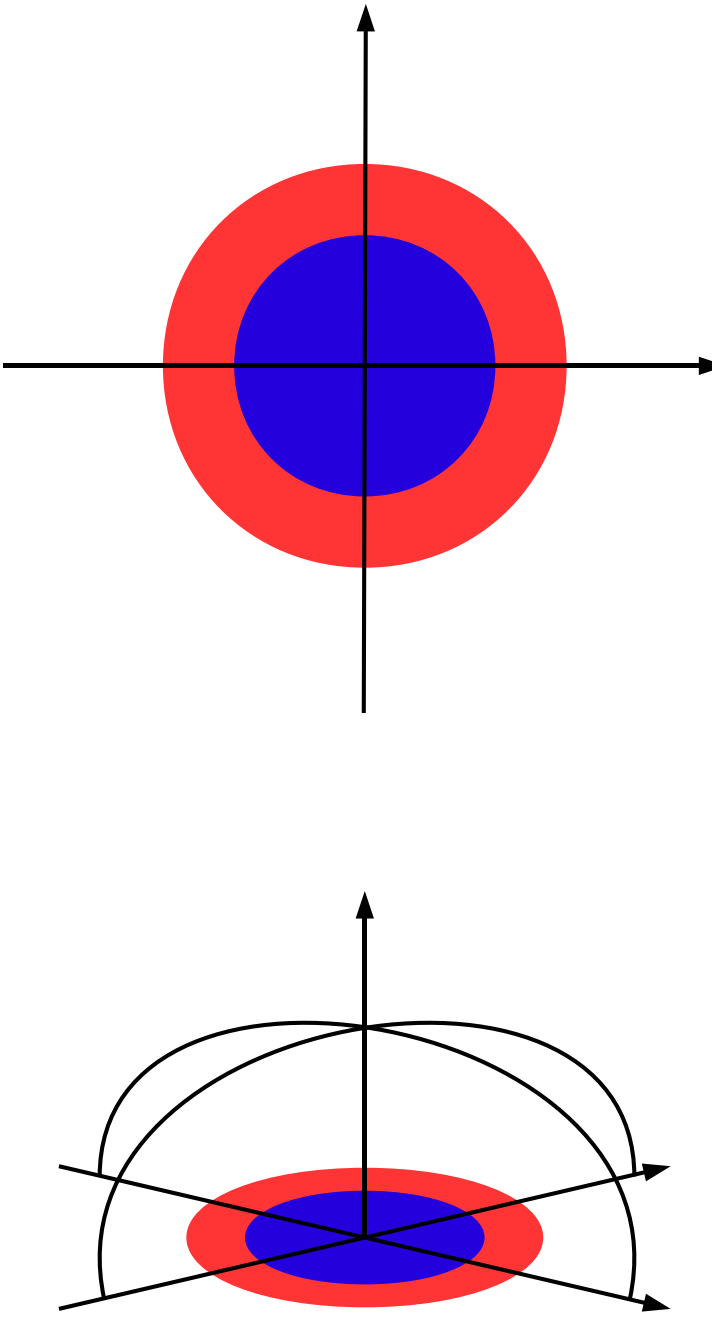
Supponiamo di avere un TS del tipo rappresentato nella figura sottostante:



Come risulta evidente, non ci sono trasformazioni lineari che rendano le classi separabili. Operiamo quindi la seguente trasformazione, $x \rightarrow (x, x^2)$ ottenendo



Caso bidimensionale



Il problema ovviamente si complica, a causa dell'introduzione di nuove variabili. Nel caso bidimensionale, ad esempio possiamo applicare le seguenti trasformazioni:

- $(x, y) \longrightarrow \left(x, y, e^{x^2+y^2}\right)$;
- $(x, y) \longrightarrow \left(x, y, e^{-\left(x^2+y^2\right)}\right)$ - trasformazione gaussiana.

7 Reti bayesiane

7.1 Definizioni generali e proprietà

Le reti bayesiane ci consentono di modellare un fenomeno attraverso un DAG e delle funzioni di probabilità semplici, semplificando quindi la formulazione della JPF.

Se $G \equiv (V, E)$ con alcune funzioni di probabilità è una *rete bayesiana* allora la JPF$=P(x_1, \ldots, x_n)$ può essere espressa come il prodotto della probabilità di osservare x_i condizionata soltanto dai suoi progenitori, per $i = 1 \ldots n$, cioè

$$JPF = P(x_1, \ldots, x_n) = \prod_{i=1}^{n} (x_i | parent(x_i))$$

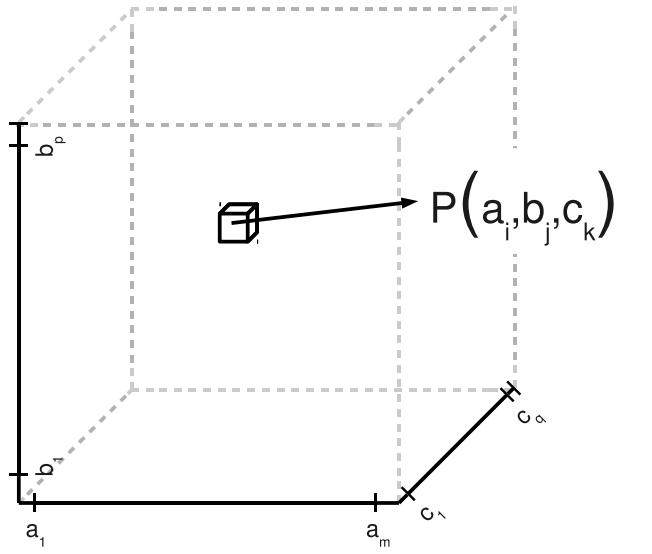
Una rete può essere usata per generare esempi con JPF prescritte, cioè una volta descritta la JPF attraverso il grafo, possiamo assegnare le probabilità a priori e calcolare le probabilità condizionate, ottenendo così una quantità di esempi che statisticamente rispettino le regole che abbiamo stabilito.

Supponiamo di avere solo tre features, e le chiamiamo `A`, `B`, `C`.

- `A` può assumere i valori $a_1, \ldots, a_m$;
- `B` può assumere i valori $b_1, \ldots, b_p$;
- `C` può assumere i valori $c_1, \ldots, c_q$.

Un modo grafico per immaginare la JPF di queste tre variabili è quello di pensare ad un parallelogramma in cui una direzione avrà etichetta `A` e valori da a_1 ad a_m, un'altra direzione avrà etichetta `B` e valori da b_1 a b_p e la terza direzione avrà etichetta `C` e valori da c_1 a c_q.

Se conosciamo i valori da assegnare a ciascuna delle $m \times p \times q$ caselle, questi $m \times p \times q$ valori definiscono la JPF.



In generale, l'unica relazione che posso dare su questa JPF è

$$\sum_i \sum_j \sum_k P(a_i, b_j, c_k) = 1$$

Sappiamo che possiamo *marginalizzare*, cioè non tenere conto di una delle tre variabili. Ad esempio, marginalizziamo la variabile A, quindi consideriamo solo la distribuzione di probabilità $P(\mathtt{B}, \mathtt{C})$.

Vogliamo sapere qual è la probabilità che si verifichino congiuntamente gli eventi b_j, c_k; vogliamo quindi calcolare

$$P(b_j, c_k) = \sum_{i=1}^{m} P(a_i, b_j, c_k)$$

vogliamo vedere se grazie al grafo riusciamo a trovare una "struttura" che descriva i valori di probabilità all'interno del cubo.

Considerando che con tre feature riusciamo già a creare un modello esasustivo, possiamo esaminare una serie di casi speciali che si possono produrre.

7.2 Correlazione tra feature

Nella trattazione seguente dei tre casi notevoli ci concentreremo sulle relazioni tra le feature A e B a seconda che il valore della feature C sia fissato (**osservato**) o meno.

I caso: catena di influenze (o di condizionamenti): $\mathtt{A} \longrightarrow \mathtt{C} \longrightarrow \mathtt{B}$

In questo caso A e B sono dipendenti visto che

$$P(\mathtt{A}, \mathtt{B}) \neq P(\mathtt{A})\, P(\mathtt{B})$$

Osservando C, però, si ha che

$$P(\mathtt{A}|\mathtt{C})\, P(\mathtt{B}|\mathtt{C}) = P(\mathtt{A}, \mathtt{B}|\mathtt{C})$$

che è proprio la definizione di indipendenza dato C.

Quindi **osservata la feature C, le feature A e B diventano indipendenti.**

II caso: concausa: $\mathtt{A} \longleftarrow \mathtt{C} \longrightarrow \mathtt{B}$

Ancora una volta notiamo che

$$P(\mathtt{A}, \mathtt{B}) \neq P(\mathtt{A})\, P(\mathtt{B})$$

Osservato C però si ha anche in questo caso

$$P(\mathtt{A}|\mathtt{C})\, P(\mathtt{B}|\mathtt{C}) = P(\mathtt{A}, \mathtt{B}|\mathtt{C})$$

Come nel I caso, **osservata la feature C, le feature A e B diventano indipendenti.**

III caso: concorrenza di cause: $\mathtt{A} \longrightarrow \mathtt{C} \longleftarrow \mathtt{B}$

Qui, A e B sono indipendenti. Infatti

$$P(\mathtt{A}, \mathtt{B}) = P(\mathtt{A})\, P(\mathtt{B})$$

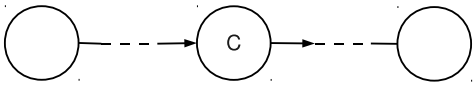
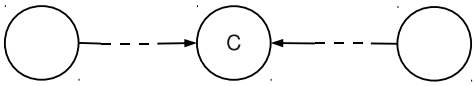
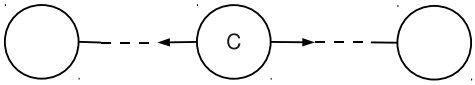
Se però fissiamo il valore di C (quindi del "sintomo"), stiamo in qualche modo condizionando le cause tra di loro. Matematicamente

$$P(\mathtt{A}, \mathtt{B}|\mathtt{C}) = \frac{P(\mathtt{A}, \mathtt{B}, \mathtt{C})}{P(\mathtt{C})} = \frac{P(\mathtt{A})\, P(\mathtt{B})\, P(\mathtt{C}|\mathtt{A}, \mathtt{B})}{P(\mathtt{C})}$$

Poichè il rapporto $\frac{P(\mathtt{C}|\mathtt{A},\mathtt{B})}{P(C)} \neq 1$, possiamo dire che **A e B sono dipendenti osservato C.**

Questa tecnica (fissare un valore per il "sintomo" e tralasciare una delle cause) è nota come **explaining away**.

NOTAZIONE:

Nodo **testa-coda**	
Nodo **testa-testa**	
Nodo **coda-coda**	

Vogliamo ora valutare come in un grafo le correlazioni tra nodi possono semplificare il nostro lavoro di valutazione della JPF.

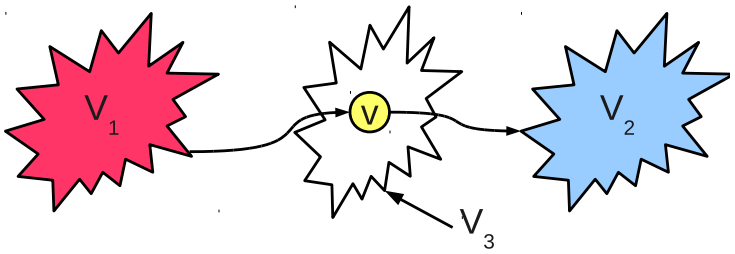
Definizione 7.1. Nodi bloccanti

Dato un grafo $G \equiv (V, E)$ consideriamo $V_1, V_2, V_3 \subseteq V$ tali che $V_1 \cap V_2 \cap V_3 = \emptyset$. Consideriamo un cammino π tra $v_1 \in V_1$ e $v_2 \in V_2$. Osservato un nodo v_3, un nodo v si dice bloccante per il cammino considerato se vale una delle due seguenti condizioni:

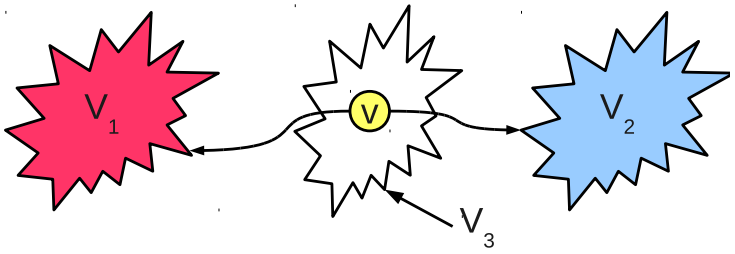
1. v è un discendente di v_3 e in esso gli archi del cammino formano un **testa-coda** o un **coda-coda**, cioè v_1 e v_2 sono indipendenti osservato v_3;
2. v non è un discendente di v_3 e non ha discendenti in comune con v_3 nei quali gli archi del cammino formino un **testa-testa**.

Per comprendere meglio il significato di questa definizione, ecco una spiegazione con dei grafici.

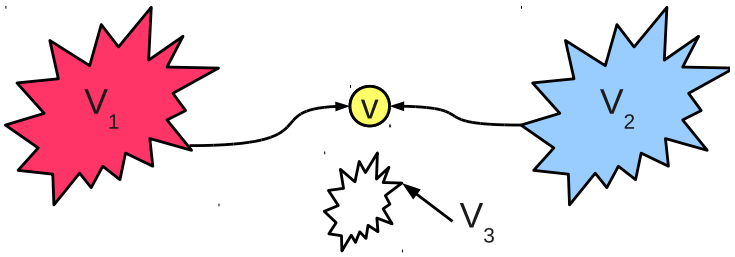
V_1, V_2 sono due sottoinsiemi di vertici; V_3 è l'insieme dei nodi osservati; v, u sono nodi.



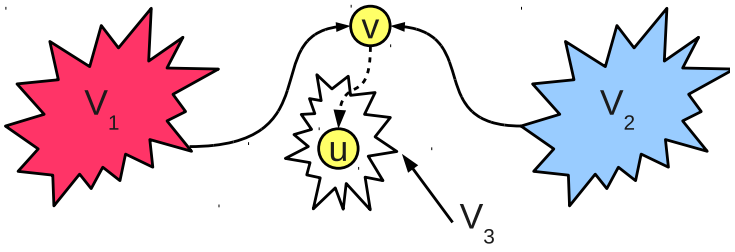
BLOCCANTE



BLOCCANTE



BLOCCANTE



PASSANTE

Definizione 7.2. D-Separation

Dati due insiemi di feature $\mathcal{A} = \{a_1, \ldots, a_n\}$ e $\mathcal{B} = \{b_1, \ldots, b_h\}$ diremo che essi sono *d-separati* se **ogni** cammino da a_i a b_j, $i = 1 \ldots n$, $j = 1 \ldots h$ è **bloccato.**

8 Markov

Definizione 8.1. Ipotesi di Markov

Una feature è influenzata direttamente solo da un numero finito (piccolo) di altre feature.

Definizione 8.2. Markov Blanket

Dato un nodo A in una rete di Bayes, chiameremo **Markov blanket** di A l'insieme più piccolo di variabili che la condizionano. In pratica

$$MB(A) = \{parent(A)\} \cup \{children(A)\} \cup \{parent(children(A))\}$$

Definizione 8.3. Naive-Bayes Rule

Assumiamo che tutte le feature siano indipendenti tra loro, trascurando quindi le (importanti) connessioni tra esse al fine di semplificare i calcoli. Sia $\{x_1, \ldots, x_n\}$ un insieme di feature e sia $\{w_1, \ldots, w_k\}$ un insieme di classi. Allora piuttosto che calcolare

$$P(w_i|x_1, \ldots, x_n)$$

calcolo

$$\frac{P(x_1, \ldots, x_n|w_i)\, P(w_i)}{P(x_1, \ldots, x_n)}$$

Grazie all'assunzione di indipendenza tra le feature, la formula precedente diventa

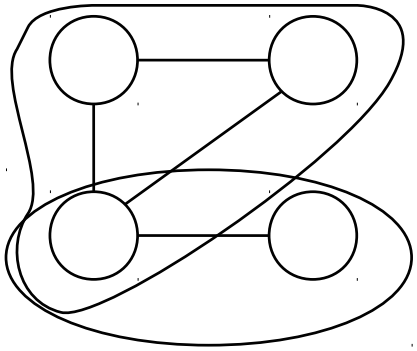
$$\frac{P(x_1|w_i) \cdots P(x_n|w_i) \cdot P(w_i)}{P(x_1) \cdot P(x_2) \cdots P(x_n)}$$

ossia una fattorizzazione di probabilità più semplici da calcolare rispetto alla formula corretta. Ovviamente c'è un errore dovuto al tralasciare le dipendenze tra le feature, ma paradossalmente, per $n \gg 0$, la regola naive di Bayes fornisce buoni risultati.

8.1 Campi di Markov

8.1.1 Fattorizzazione per clique

Rappresentando il set di feature tramite un grafo G dove gli archi indicano la dipendenza tra le feature, è possibile semplificare il calcolo della PDF fattorizzando, in accordo all'ipotesi di Markov, il grafo per clique massimali.



Quindi

$$P(G) = \prod_{\text{max clique}} P(clique_i)$$

La ripetizione di alcune feature non causa problemi, visto che comunque si ha un "risparmio" in termini di calcolo. Si deve in ogni caso normalizzare la produttoria, per riportarla ad un valore consistente con la definizione di probabilità, $0 \leq P(G) \leq 1$.

Definiamo quindi

$$\widetilde{P}(v_1, \ldots, v_n) = \frac{\prod_i P(clique_i)}{Z}, \qquad i = 1, \ldots, \#\text{clique massimali}$$

dove Z è detto *fattore di normalizzazione o di partizione.*

Si tratta quindi di scomporre il grafo in clique massimali, ciascuna delle quali corrisponde ad un fattore della produttoria

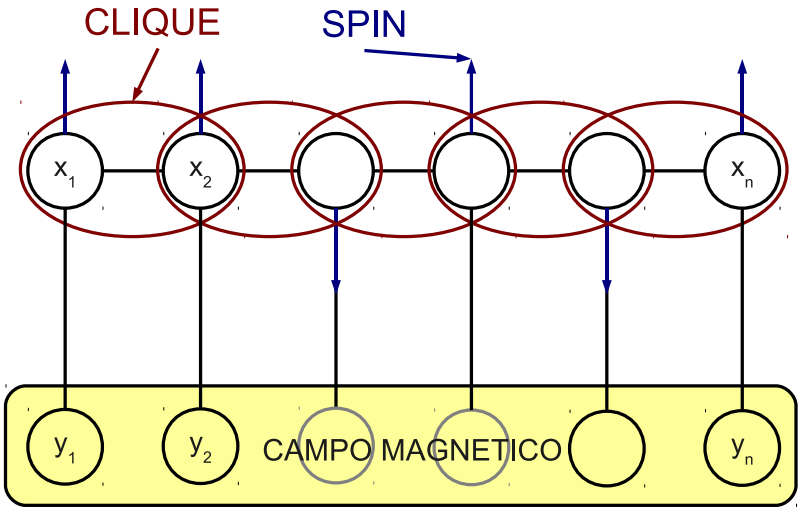
Per garantire che tutti i fattori siano non negativi (e quindi consistenti con la definizione di probabilità) possiamo utilizzare la funzione esponenziale e riscrivere quindi la formula precedente come

$$\widetilde{P}(v_1, \ldots, v_n) = \frac{1}{Z} \prod_i^k e^{f_i(v_{i1}, \ldots, v_{ik})}$$

dove Z è definito come sopra e $f_i(v_{i1}, \ldots, v_{ik})$ sono i fattori corrispondenti alle clique.

8.1.2 Simulated annealing

Partiamo da un noto esempio di fisica, il modello di Ising. Supponiamo di avere degli atomi disposti in linea retta come in figura e supponiamo che tali atomi siano immersi in un campo magnetico



Il campo magnetico orienterà gli spin degli atomi in maniera probabilistica.

Allora ogni atomo x_i è influenzato sia dal campo magnetico y_i che dai suoi vicini. Misuriamo l'**energia** del sistema, data dalla formula:

$$E = \underbrace{\sum_{i=1}^{n} k_1 e^{-x_i y_i}}_{\text{con il C.M.}} + \underbrace{\sum_{i=1}^{n-1} k_2 e^{x_i x_{i+1}}}_{\text{tra atomi}}$$

Algorithm 1 Ricerca della minima energia

1. Si sorteggi un indice $\bar{i}$ e si calcoli
$$D = E_0 - E_{flipped}$$
dove $E_{flipped}$ è l'energia del sistema dopo aver invertito la direzione degli spin degli atomi dipendenti da $\bar{i}$.
2. Si valuti D.
   (a) Se $D > 0 \Rightarrow$ l'energia è diminuita (si sta minimizzando l'energia). Si ponga $E_0 = E_{flipped}$ e si ritorni al passo 1.
   (b) Se $D \leq 0 \Rightarrow$ l'energia non è diminuita. Si torni al passo 1.

Vogliamo minimizzare questa energia, ma analiticamente è difficile; ricorriamo quindi a tecniche di approssimazione numerica.

Sia quindi E_0 =energia iniziale del sistema (dipende dalla configurazione iniziale degli x_i).

ricerca della minima energia

Questo algoritmo non è buono, perchè l'energia comincia a oscillare.

Allora introduciamo l'algoritmo noto come **simulated annealing.**

Algorithm 2 Simulated annealing

1. Si fissi E_0;
2. Si fissi $0 < \varepsilon < 1$ (inizialmente si scelga un valore ~ 1;
3. Si sorteggi un indice $\bar{i}$;
4. Si calcoli $D = E_0 - E_{flipped}$;
5. Si valuti D:
   (a) se $D > 0$: si generi un numero casuale $rand_1$: se $rand_1 < \varepsilon$
      - si ponga $E_0 = E_{flipped}$;
      - si diminuisca il valore di ε;
      - si torni al punto 3;
   (b) se $D \leq 0$: si generi un numero casuale $rand_2$:
      i. se $rand_2 > \varepsilon$ si torni al punto 3;
      ii. se $rand_2 \leq \varepsilon$:
         - si ponga $E_0 = E_{flipped}$;
         - si diminuisca il valore di ε;
         - e si torni al punto 3.

La diminuzione ad ogni passo del parametro ε consente all'algoritmo di stabilizzarsi e non oscillare molto.

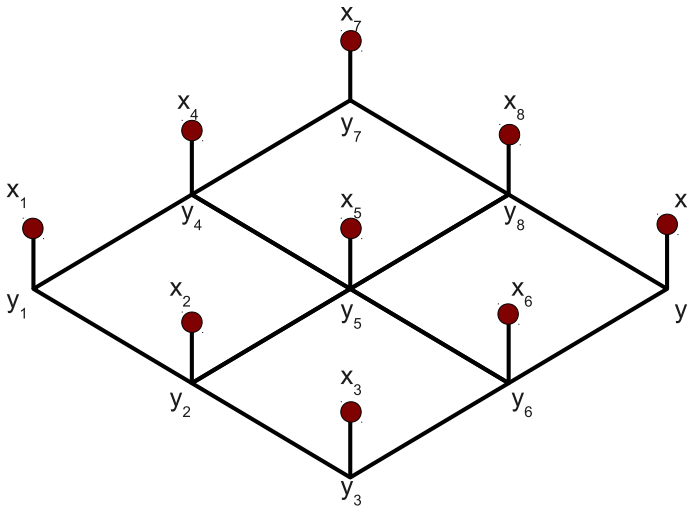
8.1.3 Denoising su immagini B/W

Supponiamo di avere un'immagine B/W in cui i pixel bianchi hanno valore $+1$ e i pixel neri hanno valore -1. Siano $\{x_i\}$ i pixel noti (quelli che vediamo) e siano $\{y_i\}$ i valori corretti da determinare. Non è detto che x_i e y_i siano uguali, a causa della presenza di rumore impulsivo.

Siano

- $P(x_i = y_i) = k_1$ (tipicamento $k_1 = 0.9$);
- $P(y_i = 0) = k_2$.

Il valore di un pixel è fortemente influenzato dal valore dei pixel vicini, cioè le immagini hanno un'elevata coerenza spaziale. Costruiamo un campo di Markov per il nostro sistema:



Vogliamo trovare i pixel $\{y_i\}$ corretti dato che abbiamo osservato gli $\{x_i\}$: quindi vogliamo massimizzare

$$P(y_1 \dots y_n | x_1 \dots x_n) = \prod_{\text{clique massimali}} clique$$

Come funzione-peso per la correlazione tra i pixel, scegliamo

$$e^{-hx_i y_i}$$

Tale funzione dipende dal prodotto $x_i \cdot y_i$ che sarà positivo se x_i e y_i saranno concordi (quindi se x_i è corretto) altrimenti sarà negativo.

L'energia del sistema sarà data quindi da

$$E = \underbrace{\sum_i e^{-\eta x_i y_i}}_{\text{penalità associata ad un errore}} + \sum_{clique} e^{-\vartheta y_i y_{vicino(i)}}$$

Applicando il *simulated annealing* al campo di Markov così definito otterremo il miglioramento dell'immagine.

La qualità dell'immagine corretta dipende fortemente da η e ϑ: una buona scelta è quella di scegliere valori "piccoli".

8.2 Catene di Markov

Le *Catene di Markov* sono un modello utile per studiare sequenze di evoluzione di un sistema. Ad ogni feature considerata, associamo uno **stato**: in questo modo otteniamo un diagramma detto **diagramma di stato** (simile ai diagrammi per gli automi a stati finiti).

Le transizioni da uno stato al successivo dipendo da:

- **stato attuale;**
- **regole.**

Se un sistema è caratterizzato da un vettore di stato $\vec{s} = (x_1 \dots x_n)$, diremo che il sistema è **markoviano di ordine** n se

$$s_t = f\left(s_{t-1}, s_{t-2}, \dots, s_{t-n}\right)$$

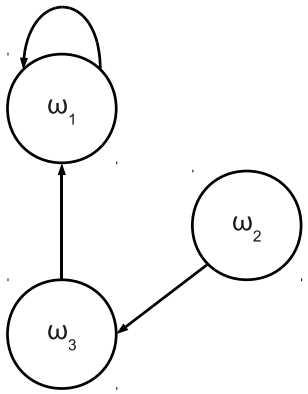
Considereremo sistemi markoviani di ordine 1, tali cioè che

$$s_t = f\left(s_{t-1}\right)$$

Se gli stati sono in numero finito, posso rappresentare questo sistema con un grafo.

8.2.1 Modelli di Markov deterministici

Dato un grafo, posso rappresentarlo con una tabella, come esplicitato in figura



	ω_1	ω_2	ω_3
ω_1	1	0	0
ω_2	0	0	1
ω_3	1	0	0

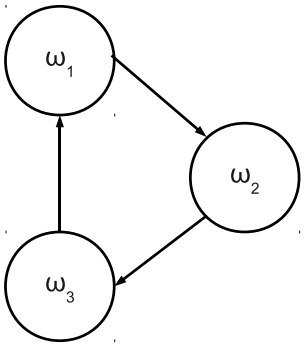
Un sistema di questo tipo è detto **deterministico**, in quanto l'evoluzione non necessita di "azioni" esterne e le transizioni da uno stato all'altro sono ben definite. In formule:

$$\forall \omega_i \; \exists_1 \omega_j : \; \omega_{ij} = 1$$

Se ogni stato è raggiungibile da ogni altro stato, il modello si dice **ergodico**. In formule:

$$\begin{aligned} \forall \omega_i \; \exists_1 \omega_j : \; \omega_{ij} &= 1 \\ \forall \omega_j \; \exists_1 \omega_i : \; \omega_{ij} &= 1 \end{aligned}$$

cioè abbiamo un modello del tipo



	ω_1	ω_2	ω_3
ω_1	0	1	0
ω_2	0	0	1
ω_3	1	0	0

8.2.2 Modelli di Markov probabilistici

Se un modello di Markov non è deterministico, allora è detto **probabilistico**: cioè le matrice indica la probabilitù di passare dallo stato ω_i allo stato ω_j.

Per ogni riga, deve valere che

$$\forall i \quad \sum_j \omega_{ij} = 1$$

Definizione 8.4. Steady state

Dato un modello di Markov descritto da una matrice di transizione M definiamo **steady state (stato stabile)** lo stato

$$q = \lim_{t \to +\infty} f(s_{t-1})$$

Considerando una sequenza di k stati avremo

$$q = \lim_{k \to +\infty} pM^k$$

dove p è lo stato di partenza.

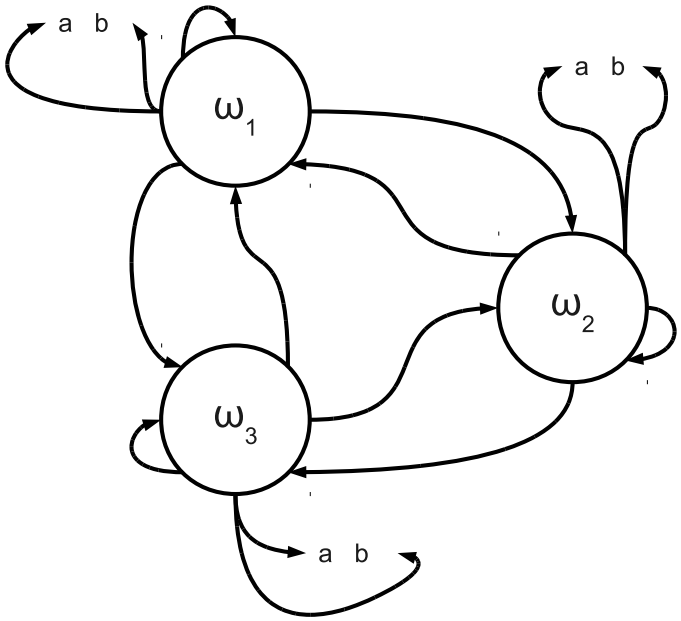
Osservazione 8.1. Osserviamo anche che se $q = \lim_{k \to +\infty} M^k p \Rightarrow Mq = q$, cioè q è un autovettore della matrice M. Poichè non tutte le matrici hanno autovettori, non tutti i sistemi hanno stati stabili.

8.3 HMM - modelli di Markov a stati nascosti

Siano

- S = insieme finito di n stati;
- M = matrice delle transizioni $n \times n$ tale che $\forall j = 1 \dots n \quad \sum_i a_{ij} = 1$;
- A = alfabeto di k simboli;
- E = matrice di **emissione** $n \times k$ tale che $\forall j = 1 \dots n \quad \sum_i e_{ij} = 1$

La matrice E rappresenta la probabilità che in sistema in un certo stato emetta un simbolo. Questi simboli sono detti **stati nascosti**, e nel grafo vengono rappresentati come in figura.



Un sistema così definito è detto **HMM (Hidden Markov Model)**. Serve a modellare sistemi nei quali non conosciamo gli stati interni (oppure è troppo difficile valutarli) ma solo gli effetti che gli stati hanno sul sistema.

Esistono tre tipologie di problemi inerenti gli HMM:

1. *Evaluation (valutazione):*
   dato un modello HMM e una stringa di simboli emessa dal modello, vogliamo conoscere la probabilità di emissione della stringa. Tale problema viene risolto tramite l'**algoritmo di Trellis**.

2. *Decoding (decodifica)*
   dato un modello HMM e una stringa osservata di simboli, vogliamo dedurre la sequenza più probabile di stati interni che corrisponde alla stringa osservata. Tale problema viene risolto tramite l'**algoritmo di Viterbi**.

3. *Learning (apprendimento)*
   dati S, A e TS training set ricavare le matrici di transizione M e di emissione E. Tale problema viene risolto tramite l'**algoritmo di Baum-Welch**.

8.4 Algoritmo di Trellis

Poichè l'approccio di forza bruta, seppur efficace, non è efficiente, visto che per un modello con n stati ed una stringa di lunghezza k il tempo computazionale è $O\left(n^k\right)$, applichiamo il seguente algoritmo per risolvere il problema della valutazione.

Algorithm 3 Algoritmo di Trellis - forward version

1. Si costruisca una tabella del tipo

	b_1	b_2	$\cdots$	b_k
s_1				
s_2				
$\vdots$				
s_n				

2. Partendo dalla colonna b_1, si riempia la tabella con i valori $\alpha_i(t) \equiv$ probabilità di essere nello stato $i-$esimo all'istante t. In formule, il valore di ogni cella è dato da

$$\alpha_i(t) = b_{is_t} \cdot \sum_{j=1}^{n} \alpha_j(t-1) \cdot a_{ji}$$

dove

- b_{is_t} è la probabilità di emettere il simbolo b_i mentre siamo nello stato s_t;
- a_{jt} è la probabilità di passare dallo stato j allo stato i.

3. Nella cella $\alpha_n(k)$ troveremo la probabilità di emissione della stringa desiderata.

Algorithm 4 Algoritmo di Trellis - backward version

1. Si costruisca una tabella del tipo

	b_1	b_2	$\cdots$	b_k
s_1				
s_2				
$\vdots$				
s_n				

2. Partendo dalla colonna b_k, si riempia la tabella con i valori $\beta_i(t) \equiv$ probabilità di osservare a partire dall'istante t una sequenza che comincia con lo stato i e procede come la sequenza osservata. In formule, il valore di ogni cella è dato da:

$$\beta_i(t-1) = b_{is_{t.1}} \cdot \sum_{j=1}^{n} \beta_j(t) \cdot a_{ij}$$

dove $b_{is_{t-1}}$e a_{ij} hanno lo stesso significato visto in precedenza.

3. Nella cella $\beta_1(1)$ troveremo il valore desiderato.

8.5 Expectation-Maximization

Partendo da un *not-so-hidden MM* costruisco la matrice di transizione M e la matrice di emissione E; costruisco quindi la matrice del Trellis Forward.

Noi vogliamo calcolare un numero γ : $\gamma(i,j,t) \equiv$probabilità di trovarmi allo stato i nell'istante t e di passare allo stato j nell'istante successivo.

$$\gamma(i,j,t) = \alpha_i(t) \cdot a_{ij} \cdot \beta_j(t+1)$$

Il numero γ così calcolato **vale per una sola stringa**. Se vogliamo calcolare γ per un TS qualunque, faremo una media, pesata sulle frequenze delle stringhe, per tutti i γ calcolati.

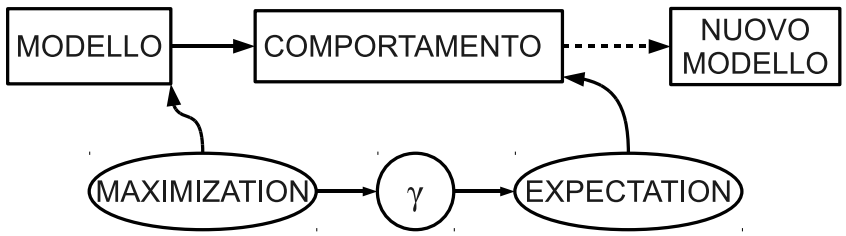
Il numero γ ci fornisce quindi una rappresentazione analitica del comportamento statistico del modello.

Una rappresentazione sintetica è data da:

- $n = \#\text{stati}$;
- $s = \#\text{simboli}$;
- a_{ij};
- b_{jk}.

Calcolando α e β (e quindi essendo in grado di calcolare γ) passo dal modello sintetico al modello analitico.

Utilizzando la rappresentazione analitica γ voglio passare ad un nuovo modello, migliore del precedente. Ovviamente questo processo è reiterabile, ed è noto come **expectation-maximization.**



Il processo E-M è alla base dell'algoritmo di Baum-Welch.

Esiste però una componente arbitraria in questo algoritmo dovuta al modello sintetico di partenza. L'algoritmo termina quando il modello converge, cioè quando un nuovo modello generato dal passo di expectation non differisce o differisce in maniera non apprezzabile dal modello di partenza.

Vediamo come ottenere **effettivamente** un nuovo modello.

Calcoliamo un nuovo a'_{ij}.

$$a'_{ij} = \frac{\#\text{volte in cui sono allo stato } i\text{-esimo e passo allo stato } j\text{-esimo}}{\#\text{volte in cui sono allo stato } i\text{-esimo e passo in un altro stato}} = \frac{H}{K}$$

dove

$$\begin{aligned} H &= \sum_{\forall t} \gamma(i,j,t) \\ K &= \sum_{\forall j} \sum_{\forall t} \gamma(i,j,t) \end{aligned}$$

Calcoliamo un nuovo b'_{ik}

$$\begin{aligned} b'_{ik} &= \frac{\#\text{volte in cui trovandomi nello stato } i\text{-esimo emetto il simbolo } k}{\#\text{volte in cui trovandomi nello stato } i\text{-esimo emetto un simbolo qualunque}} = \\ &= \frac{\sum_{t_1 \ldots t_k} \sum_{\forall j} \gamma(i,j,t_n)}{K} \end{aligned}$$

dove $t_1 \ldots t_k$ sono gli istanti in cui è stato emesso il simbolo k.

Ottengo quindi un nuovo modello!

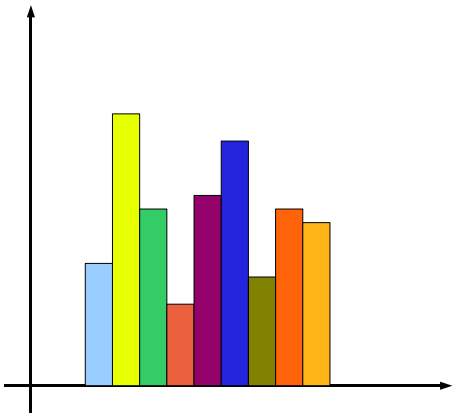
Scelgo quindi il modello M che rende massima $P(M|x_1 \dots x_n)$

$\Downarrow MAXIMIZATION \quad \Uparrow EXPECTATION$

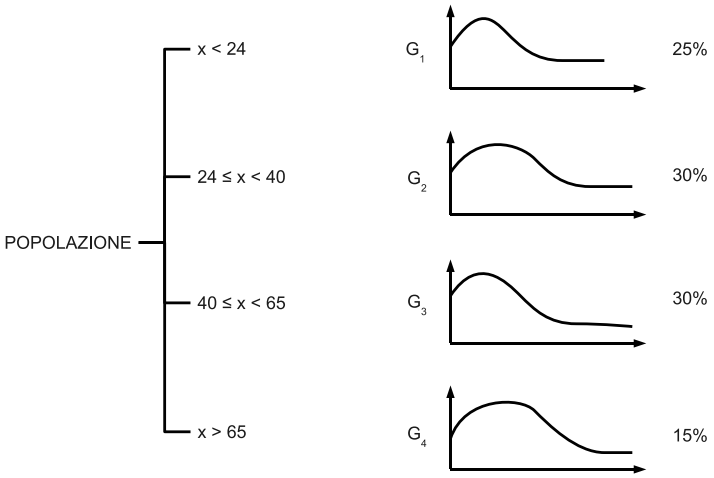
Prevedo il comportamento medio (atteso) del modello M (a partire dai valori γ).

9 Mixture of Gaussians

In certi casi, è difficile approssimare una distribuzione con una curva gaussiana; ad esempio, osserviamo un grafico del reddito pro-capite per anno.



Si può utilizzare allora un processo di *stratificazione* con un tecnica nota come **mixture of gaussians**: si divide la popolazione per "fasce" (ad esempio, per età) più omogenee e quindi più facili da approssimare con una gaussiana.



ed a questo punto si sommano le funzioni pesate per ottenere la funzione finale

$$F = 0.25 \cdot G_1 + 0.30 \cdot G_2 + 0.30 \cdot G_3 + 0.15 \cdot G_4$$

ossia, in generale:

$$F = \sum_{i=1}^{m} \overrightarrow{w_i} \cdot G\left(\overrightarrow{\mu_i}, \overrightarrow{\sigma_i}\right)$$

con

- $m = \#$gaussiane;
- $\mu_i, \ldots, \mu_m =$ medie;

- $\sigma_1, \ldots, \sigma_m$ = covarianze;
- $w_1, \ldots, w_m$ = pesi.

Chiaramente

$$\int_{-\infty}^{+\infty} F = 1$$

Questo approccio è **parametrico**.

Un altro esempio è quello delle immagini a toni di grigio. Otteniamo istogrammi molto complicati, dunque la stratificazione può essere difficile per immagini complesse o con molti dettagli.

9.1 Procedura E-M e mixture of gaussians

PASSO INIZIALE

1. Fissare m (dipende dalla conoscenza del problema, non dal metodo).
2. Dividere (arbitrariamente o con criterio dato dall'esperto) il TS in classi e calcolare, per ciascuna, μ_{i0}, σ_{i0}.
3. Calcolare quindi F, definendo i pesi come $w_{i0} = \frac{\#\text{elementi nella classe } i}{\#\text{elementi totali}}$.

PASSO DI MAXIMIZATION (**M**)

1. Esaminare uno per uno gli elementi del TS: $y_1, \ldots, y_n$.
2. Calcolare $G(\mu_j, \sigma_j)(y_i),\ j = 1 \ldots m$.
3. Scegliere l'indice j_{max} che massimizza il valore della gaussiana ed assegnare l'elemento y_i alla classe $j_{max} - esima$.

- In pratica, il passo **M** produce una partizione ove il grado di verosimiglianza è aumentato.

PASSO DI EXPECTATION (**E**)

1. Ricalcolare, in base agli assegnamenti prodotti al passo **M**, $\overrightarrow{\mu_{i1}}, \overrightarrow{\sigma_{i1}}, \overrightarrow{w_{i1}}$ (in pratica, produrre una migliore descrizione dei dati).
2. GOTO **M** fino a che non ci sono più variazioni di classe per qualche y_i oppure fino alla stabilizzazione dei parametri.

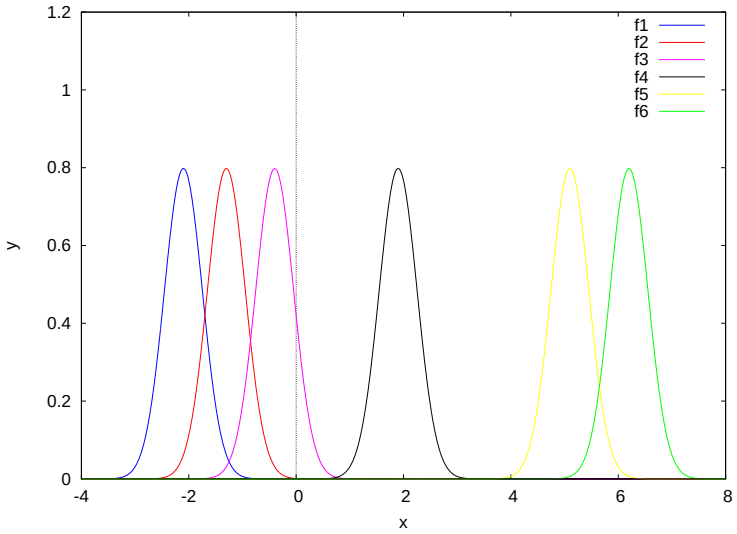
10 Metodi non parametrici per la stima della PDF

10.1 Finestre di Parzen

Il metodo delle finestre di Parzen è un metodo **non parametrico** per la stima e la descrizione di una funzione di densità di probabilità (PDF).

Esso si basa su *kernel*, finestre di dimensione non variabile: quando in tale finestra rientrano degli elementi, a seconda della funzione associata alle osservazioni (i.e. box, gaussiana, custom), vengono aggiunti dei pesi; alla fine sarà possibile generare un istogramma o, nel caso continuo, una funzione di densità di probabilità.

Ad esempio, vogliamo trovare una funzione che stima la distribuzione complessiva data da queste curve



Facciamo però un passo indietro; la densità di probabilità di osservare un evento in una data regione di spazio è data da:

$$\frac{\frac{\#\text{EVENTI NELLA REGIONE}}{\text{N}}}{\text{VOLUME DELLA REGIONE}}$$

Nel caso continuo, possiamo scrivere la formula della densità come segue

$$densità\,(y) = \frac{1}{k}\sum \phi\,(x)$$

dove $\frac{1}{k}$ è il fattore di normalizzazione e ϕ è la funzione da associare alle osservazioni.

Ad esempio, per la funzione di Gauss unidimensionale avremo

$$\phi\,(x, y) = k \cdot e^{-\frac{(x-y)^2}{\sigma^2}}$$

mentre nel caso multidimensionale avremo

$$\phi\,(\overrightarrow{x}, \overrightarrow{y}) = k \cdot e^{-\left(\overrightarrow{x}-\overrightarrow{y}\right)\Sigma^{-1}\left(\overrightarrow{x}-\overrightarrow{y}\right)^{\mathcal{T}}}$$

dove Σ è la matrice di covarianza.

Ovviamente la "densità Parzen" potrà essere stimata anche separatamente per ogni classe, operazione che si presta bene ad essere implementata da una rete neurale[1], che potrà separare per classi, assegnare i pesi ed effettuare la classificazione, ma questo esempio lo tralasciamo.

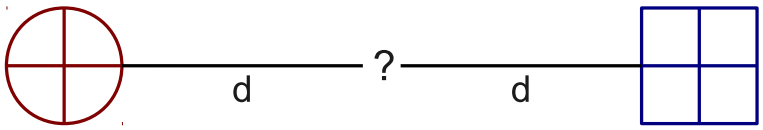
[1] in questo caso, stiamo considerando le reti neurali come "macchine / diagrammi di flusso per algoritmi paralleli" ma questo è un utilizzo riduttivo delle NN.

10.2 K-NN

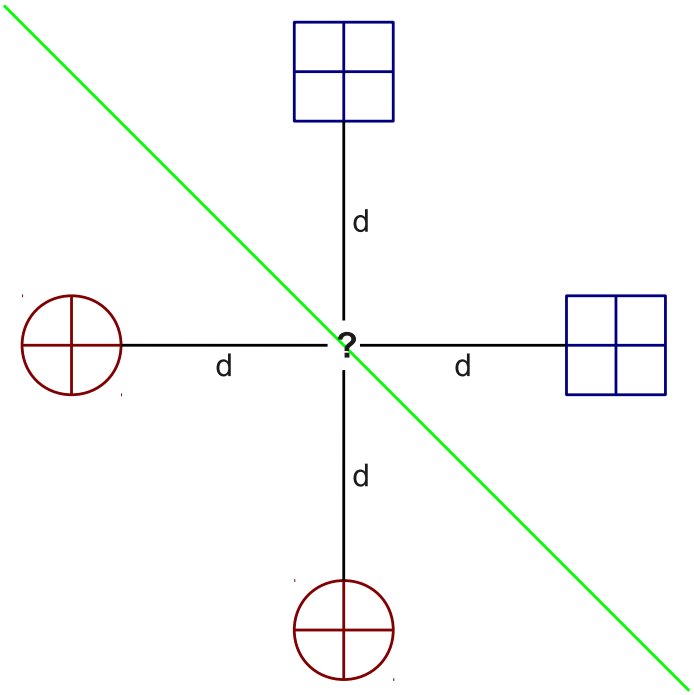
Il metodo di classificazione $k - nn$ (k-nearest-neighbor: i k elementi più vicini) è un metodo dove la densità di probabilità è ricavata mediante l'utilizzo di "finestre" di dimensione variabile (lo scopo è quello di includere k elementi).

In genere, si sceglie k dispari, per evitare situazioni di ambiguità/parità.

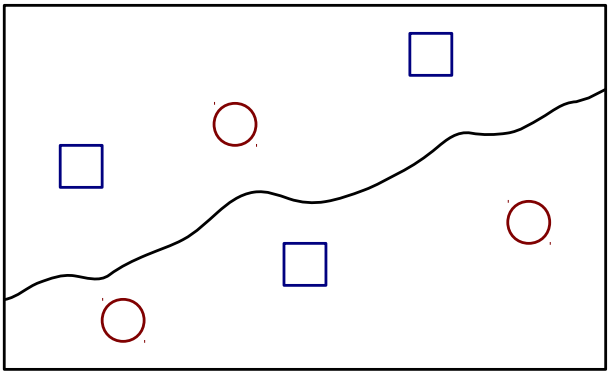
I casi di ambiguità possono verificarsi comunque, per via di elementi detti **outliers** o per via dell'overfitting; ad esempio, in $1 - nn$ potremmo avere:



cioè con la stessa distanza d e quindi decidere di passare a $3 - nn$, ma anche qui potrebbero esserci problemi:



Aumentando il valore di k aumenta la complessità dell'algoritmo e, tra l'altro, si va incontro al problema dell'overfitting per il classificatore ottenuto:

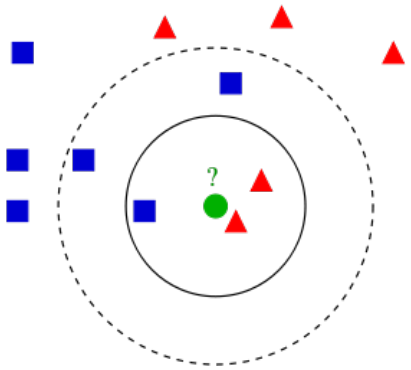


Fortunatamente esistono delle tecniche, che vedremo in seguito, che consentono di considerare gli outliers come errori e quindi di scartarli.

Lo schema di decisione $k-nn$ è basato sul "voto", nel senso che, fissati un elemento x ed il valore di k, si assegna x alla classe $\mathcal{C}$ se questa è la più frequente ai k esempi più vicini al campione in esame.

La "vicinanza" si misura in base alla distanza tra gli elementi; è possibile dunque "pesare" i contributi dei vicini in base alla distanza (ad esempio, dando un peso $1/d$ ad ogni vicino, con d =distanza). Vedremo in seguito come calcolare le distanze.

I vicini sono presi da un insieme di oggetti per i quali è nota la classificazione corretta (c'è quindi una fase di training o comunque possiamo considerarla come tale). Ad esempio



Se $k = 3$ (cerchio più interno), allora ? $\Rightarrow \Delta$; se $k = 5$ (cerchio più esterno), allora ? $\Rightarrow \blacksquare$.

Aumentando il valore di k si riduce il rumore; al tendere di $k \to \infty$, l'algoritmo non supera mai di due volte il rate degli errori commessi con l'approccio bayesiano (addirittura, per certi valori di k, l'algoritmo raggiunge il Bayes Error Rate).

Vediamo adesso alcuni algoritmi che "legano" il $k - nn$ al training set.

Algorithm 5 Algoritmo di Condensing randomizzato

TS originale $\overset{\text{vogliamo ottenere}}{\longrightarrow}$ T_C (TS condensato)

1. Estratto un y a caso dal TS, sia $T_C = \{y\}$. Si crei $S =$TS, ($S =$ insieme di candidati per T_C)
2. `while` $S \neq \emptyset$
3. Si prenda un elemento a caso x da S
4. e lo si classifichi usando $1 - nn$ con training set T_C;
5. se x è classificato correttamente, lo si scarti
6. altrimenti lo si aggiunga a T_C e lo si tolga da S.

L'algoritmo appena presentato è randomizzato, quindi non è detto che eseguendolo più volte si ottengano sempre gli stessi risultati.

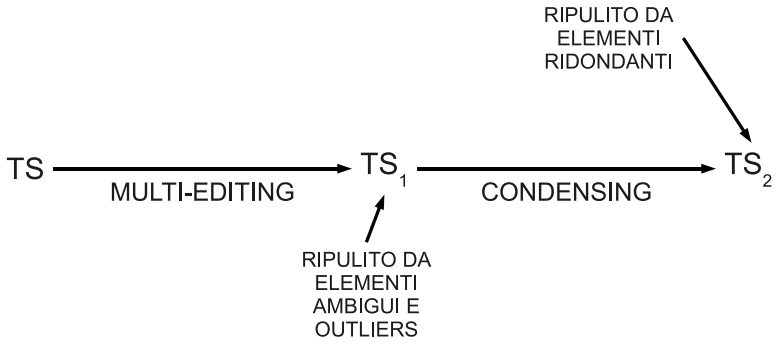
- **Miglioramento:** eseguire l'algoritmo diverse volte ed assegnare una "nota negativa" agli elementi scartati al passo 5.
- **Strategia:** una volta ottenuto T_C, lo si potrebbe riutilizzare e risemplificare ulteriormente fino ad un certo valore di miglioramento.

L'algoritmo di condensing non risolve il problema degli outliers, a meno di non usare k elevati (con ciò che comporta) o l'algoritmo di **multi-editing**, che consente di eliminare gli outliers e di utilizzare algoritmi $k - nn$ con k piccoli.

Algorithm 6 Algoritmo di multi-editing

1. Si ripartisca il TS in h insiemi disgiunti e casuali $S_1, \ldots, S_h$;
2. Si classifichi S_2 usando $k - nn$ con S_1 come TS (S_2 rispetto ad S_1). Se un elemento di S_2 è classificato correttamente lo si tenga, altrimenti lo si scarti.
3. Ripetere l'algoritmo con S_k rispetto a S_{k-1}, $k = 2 \ldots h$.
4. Si concluda con S_1 rispetto ad S_h

Con le due strategia appena viste possiamo anche ripulire il TS dagli elementi ridondanti e ridurre n:



10.2.1 Misura delle distanze

Per i record di un set, la distanza è una misura della differenza tra i record, misurata secondo un certo criterio (metrica).

In matematica, comunque, la distanza ha certe proprietà ben definite; ad esempio, per

$$d: \ S \times S \to \mathbb{R}_0^+$$

si hanno le seguenti proprietà:

1. $d(x,y) \geq 0$;
2. $d(x,x) = 0$;
3. $d(x,y) = d(y,x)$;
4. $d(x,y) = d(x,z) + d(z,y)$.

Metrica di Tanimoto Detta anche misura della similarità di Tanimoto, introdotta per la prima volta per il confronto di insiemi, è definita (nella forma originale) come:

$$d_T(x,y) = \frac{(x,y)}{\|x\|^2 + \|y\|^2 - (x,y)}$$

Ma a noi interessa la forma insiemistica, cioè

$$d_T(A,B) = \frac{n(A \cap B)}{n(A) + n(B) - n(A \cap B)}$$

con A e B non ordinati distinti ed $n(X)$= cardinalità dell'insieme X.

La similarità tra A e B può essere quindi definita come il rapporto tra il numero di elementi in comune ed il numero di elementi totali.

Metriche L^p Le metriche L^p, $o \leq p \leq +\infty$ rappresentano un altro modo per definire la distanza.

Siano $\vec{x}$ e $\vec{y}$; si ha:

$$L^p(\vec{x},\vec{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Per cui, ad esempio:

$$\begin{aligned} L^1(\vec{x},\vec{y}) &= \sum_{i=1}^{n} |x_i - y_i| \Rightarrow \text{distanza } \textbf{Manhattan} \\ L^2(\vec{x},\vec{y}) &= \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \Rightarrow \text{distanza euclidea} \end{aligned}$$

Per $p = 0$, si ha la **distanza di editing**: ci dice su quante coordinate x e y sono differenti e si calcola con

$$L^0(\vec{x},\vec{y}) = \sum_{i=1}^{n} \theta(x_i - y_i)$$

con

$$\theta(t) = \begin{cases} 1 & \text{se } t \neq 0 \\ 0 & \text{se } t = 0 \end{cases}$$

Per $p = +\infty$, allora:

$$L^{\infty}(\vec{x}, \vec{y}) = \min_i \{|x_i - y_i|\}$$

NOTA: nel calcolare le distanze, conviene normalizzare i dati per "ridurne lo spazio" a quadrati o ipercubi (2 o più features):

$$\nu' = \frac{valore - min}{max - min}$$

e quindi $\nu' \in [0, 1]$.

Distanza tangente Con le definizioni di distanza date fino ad ora c'è un problema: dati simili possono apparire molto distanti; ad esempio, con l'OCR.

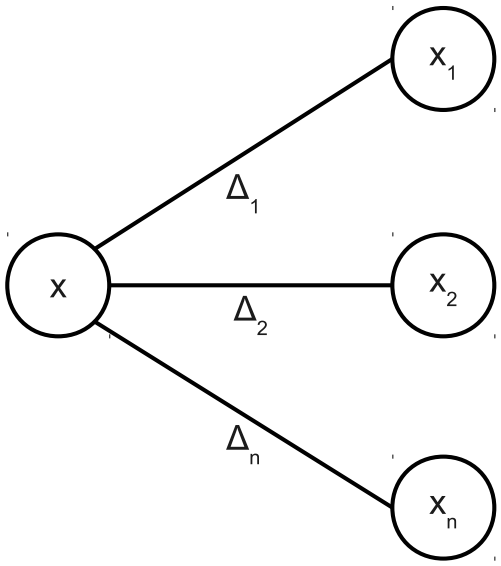
Con poche trasformazioni, tra l'altro, le due figure sono uguali, per cui bisogna definire d in modo che sia robusta rispetto ad una famiglia di trasformazioni:

$$d(\vec{x}, f(\vec{x})) < \varepsilon$$

Quindi si prende il pattern fondamentale $\vec{x}$ rispetto al quale bisogna fare il confronto e una serie di

$$\underbrace{\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n}}_{\text{trasformazioni ammissibili di } \overrightarrow{x}}$$

e si calcolano le varie Δ rispetto ad $\overrightarrow{x}$:



Consideriamo $\left\{\vec{x} + \sum_{trasformazioni} \alpha_i \Delta_i\right\}$ con α_i indeterminati: al variare degli α_i otteniamo diverse variazioni.

E' chiaro, a questo punto, che si parla di un concetto generalizzato di di distanza, detto DISTANZA TANGENTE:

$$d_T(\vec{y}, \vec{x}) = \min_{(\alpha_1 \ldots \alpha_n)} d\left(\vec{y}, \vec{x} + \sum \alpha_i \Delta_i\right)$$

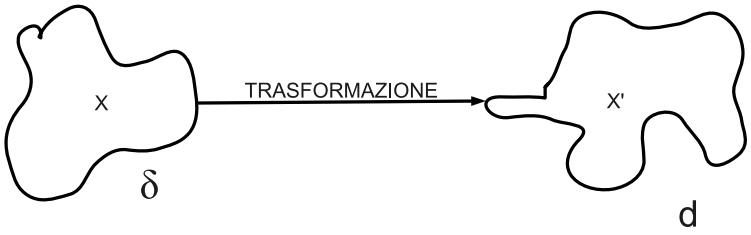
NOTA 1: se la distanza è euclidea, gli α_i saranno al quadrato, quindi per calcolare il minimo faremo la derivata ottenendo un'equazione di primo grado, lineare.

NOTA 2: naturalmente, il metodo richiede una conoscenza del problema a priori per scegliere la famiglia di trasformazioni.

10.3 Errore di rappresentazione

Finora, per diminuire di una dimensione lo spazio del problema, abbiamo visto il metodo PCA; tuttavia, a volte si ha l'esigenza di obbedire ad un criterio per "immegere" i dati senza introdurre distorsioni eccessive (a volte si parla di "preservazione delle caratteristiche topologiche degli spazi").

In genere, infatti, quando si passa da uno spazio ad un altro:



È possibile definire, dato N il numero dei dati indicizzati, la quantità:

$$\delta_{ij} = \delta(x_i, x_j), \text{ per } i = 1 \ldots N, \text{ avendo quindi } O\left(N^2\right)$$

e

$$d_{ij} = d(F(x_i), F(x_j))$$

e misurare lo scarto come segue:

$$(d_{ij} - \delta_{ij})^2$$

Se tale differenza è elevata, la rappresentazione scelta non è buona.

L'errore trovato precedentemente può essere normalizzato dividendolo per δ_{ij}^2 giungendo così alla definizione di **errore di rappresentazione:**

$$\sum_{i<j} \frac{(d_{ij} - \delta_{ij})^2}{\delta_{ij}^2}$$

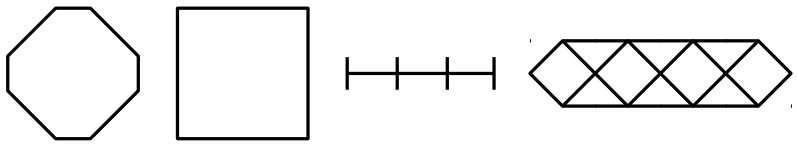
10.4 Mappe auto-organizzanti di Kohonen

Questa tecnica ci consente di trasformare i dati portandoli da uno spazio originale ad uno più semplice, rispettandone la struttura.

Per creare le SOM (Self-Organizing Maps) ci servono quattro ingredienti:

1. i dati (insieme dei dati S);

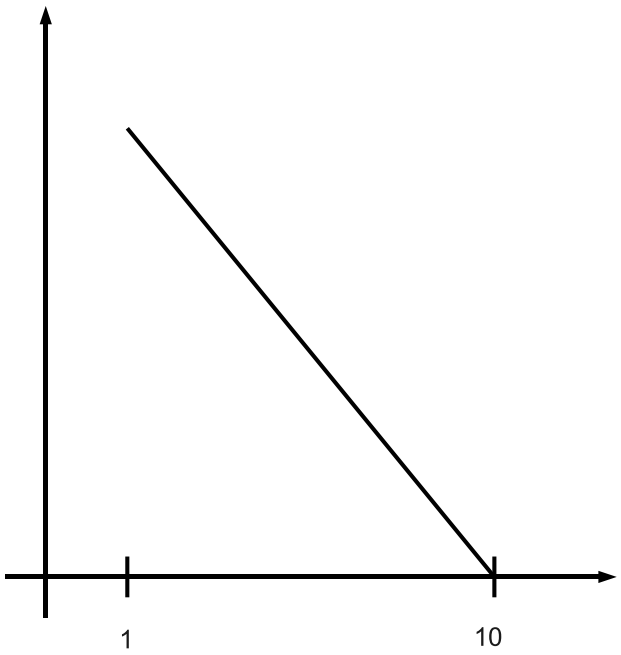
2. un supporto $\Rightarrow$ un grafo, in genere un reticolo regolare, ad esempio



Sono importanti, nel grafo, i vertici ed il "vicinato" (i "vicini dei vicini dei vicini", fino ad un certo livello di profondità, fissato al punto 3);
Def: un intorno di diametro n di un vettore $\vec{v}$ in un grafo è l'insieme di tutti i vicini $\vec{w}$ tali che esiste un cammino tra $\vec{v}$ e $\vec{w}$ non più lungo di n.

3. una funzione di influenza $g(n)$ non negativa ma decrescente che dica, per ogni vertice, quanto questo influisce sui vicini di distanza n; ad esempio

$$g(n) = \begin{cases} 10 & se\, n = 1 \\ 9 & se\, n = 2 \\ \vdots & \\ 1 & se\, n = 9 \\ 0 & se\, n \geq 0 \end{cases}$$



4. una funzione $\eta(t) : [0, +\infty] \underset{decrescente}{\longrightarrow} [0, 1]$ per forzare la convergenza della mappa.

Definiti gli ingredienti, vediamo l'algoritmo, che è suddiviso in due fasi:

Algorithm 7 SOM

1. INIZIALIZZAZIONE (random):
   Si estraggano da S tanti dati a caso quanti sono i vertici del grafo e si associno i dati ai vertici.

2. TRAINING (o: competizione tra i nodi):
   Si estragga un record alla volta a caso da S, aggiornando nel frattempo il tempo $t \Rightarrow t+1$, e si trovi il nodo del grafo più simile al dato estratto (considerando quindi le distanze). Si aggiungano il nodo ed i suoi vicini nel modo seguente:
   se $\nu(t)$ è il valore di una feature del nodo, allora

$$v_i(t+1) = v_i(t) + \eta(t) \cdot g(\text{distanza di } \nu \text{ dal record}) \cdot (\nu_{i\,NODO} + \nu_{RECORD\,ESTRATTO})$$

Quindi, man mano che estraggo record, record simili tenderanno ad aggregarsi, in quanto ogni nuovo nodo aggiunto rende il nodo a lui simile (ed anche i vicini) ancora più simili.

Quello di Kohonen è quindi un metodo non lineare per ridurre la dimensione dei dati.

Tale metodo funziona anche in presenza di features qualitative (ad esempio, maschio/femmina); per poter far funzionare il metodo in questo caso, si prendono le qualità e si mettono in una scala di valori, ad esempio

MASCHIO=0
FEMMINA=1

oppure

BUONO=1 , SUFFICIENTE=0.5 , SCARSO=0.

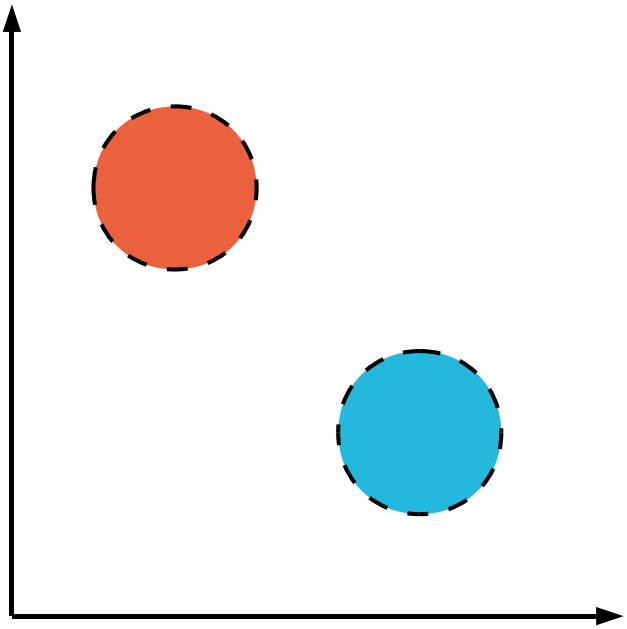
Ovviamente ciò non è sempre possibile, perchè non sempre è possibile definire una scala o ordinare i valori.

NOTA: usando come spazio un grafo, una possibile distanza tra u e v è il cammino minimo tra u e v, detto $d(u,v)$.

11 Clustering

Il **clustering** (o **vector quantization, vector coding, coding**) è il raggruppamento dei dati (un gruppo è detto **cluster**).

Supponiamo di avere, ad esempio:

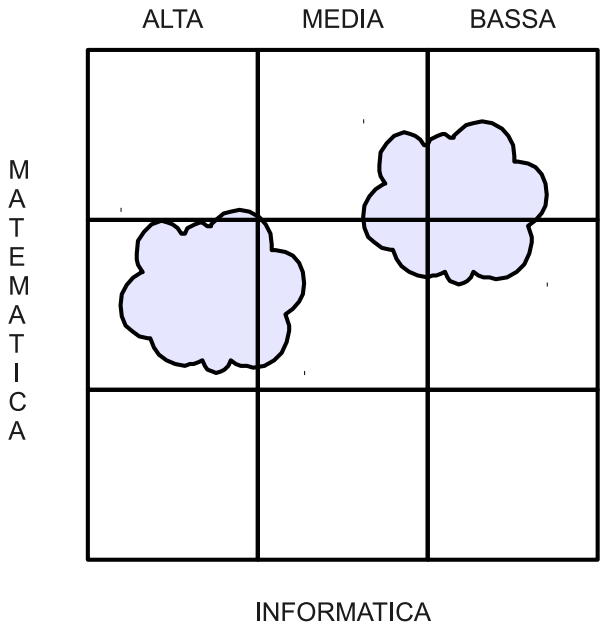


Qui possiamo individuare graficamente dei gruppi. Raggruppare in cluster può essere utile anche per memorizzare i dati, perchè se ad esempio anzichè memorizzare, per ogni elemento, le coordinate (x, y), memorizziamo $(\Delta x, \Delta y)$ con $\Delta x = x - \overline{x_1}$, $\Delta y = y - \overline{y_1}$ e $(\overline{x_1}, \overline{y_1})$ baricentro del cluster di appartenenza del punto (x, y), allora per memorizzare le coordinate di punti molto distanti occorrerà meno memoria (scala più piccola). L'esempio può essere esteso a n dimensioni (es. 3D: lo spazio RGB dei colori di un'immagine).

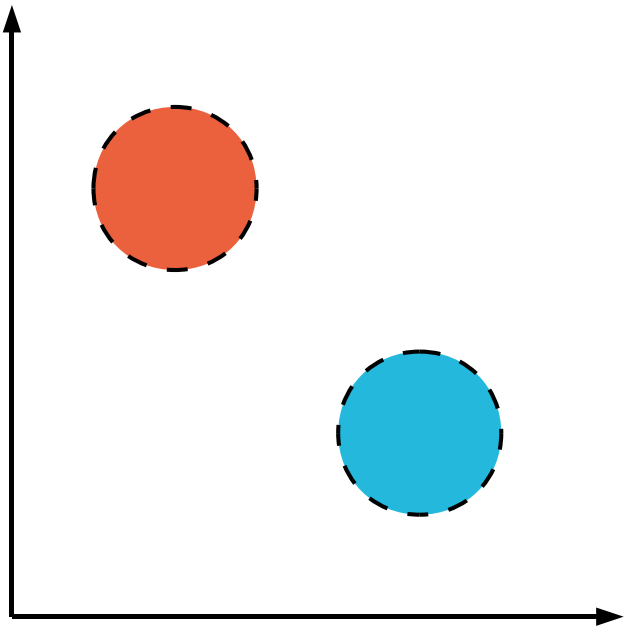
Esistono tre tipi di approcci/metodi di clustering:

- **supervisionato:**
  l'esperto decide quanti e quali cluster creare; ad esempio *salmoni/sea bass;*
- **semi-supervisionato:**
  l'esperto dice solo quanti cluster creare, ma non quali; ad esempio: forma base del $k-$means;
- **non supervisionato:**
  l'algoritmo decide autonomamente quanti e quali cluster creare.

Il guaio dell'approccio supervisionato è che potrebbero esserci gruppi vuoti (con conseguente spreco di risorse per memorizzazione e calcoli); ad esempio, già nel 2D, volendo classificare gli studenti in base alla media nelle materie matematiche ed alla media nelle materie informatiche, con tre "livelli" per ogni feature, potremmo avere



con $3 \times 3 = 9$ cluster ci sono zone vuote e sprechi, mentre magari un approccio non supervisionato ci avrebbe dato



con solo 2 cluster.

Un'idea di partenza è quella di scegliere k valori medi ed assegnare gli altri elementi ad uno dei k cluster così

Algorithm 8 Algoritmo $k-$means

1. INIZIALIZZAZIONE
   si scelgano a caso le k medie.
   (a) ASSEGNAZIONE
   si assegni ciascun elemento x al cluster con la media più vicina;
   (b) AGGIORNAMENTO
   si (ri)calcolino per ciascun gruppo i valori medi (means) individuando così i centroidi.
2. Si ripetano i passi (a) e (b) fino ad ottenere una certa stabilità.

definiti in base alla distanza tra l'elemento osservato ed i vari valori medi/baricentri dei cluster: questa è la base dell'algoritmo dei $k-$means.

11.1 K-means

L'algoritmo $k-$means (o $k-$medie) è un algoritmo di clustering che consente di suddividere n osservazioni in k cluster; ogni osservazione apparterrà al cluster con la media più vicina.

Definiamo un valore J, detto "distorsione", che è un indicatore della distanza intra-cluster:

$$J_s = \sum_{i=1}^{k} \sum_{\overrightarrow{x_j} \in S_i} \|\overrightarrow{x_j} - \mu_i\|^2$$

dove μ_i è la media "centroide" di S_i.

Lo scopo è quello di minimizzare J. La soluzione è NP-Hard, per cui ci accontenteremo di cercare soluzioni approssimate.

Osservazione 11.1. Se J deve essere minimo, ciascun elemento x deve appartenere all'insieme S_J tale che $d(x, c_J)$ sia minimo

Osservazione 11.2. La derivata nel minimo deve essere $0 \Rightarrow \frac{\partial J}{\partial C_i} = 0$. Si può dimostrare che se J deve essere minimo, ciascun C_J deve essere il valore medio dell'insieme S_J

L'algoritmo $k-$means è basato su due passi ripetuti più volte al fine di convergere verso una soluzione stabile; il "ciclo" $a \to b \to a \to b \to \cdots$ viene eseguito dopo una prima fase di inizializzazione durante la quale vengono scelte a caso le k medie.

Dal momento che l'inizializzazione è casuale, la scelta iniziale potrebbe non essere "buona" e condurre ad una soluzione non ottimale (o comunque ad una esecuzione non performante). Una possibile soluzione è quella di scegliere più volte a caso più volte l'inizializzazione.

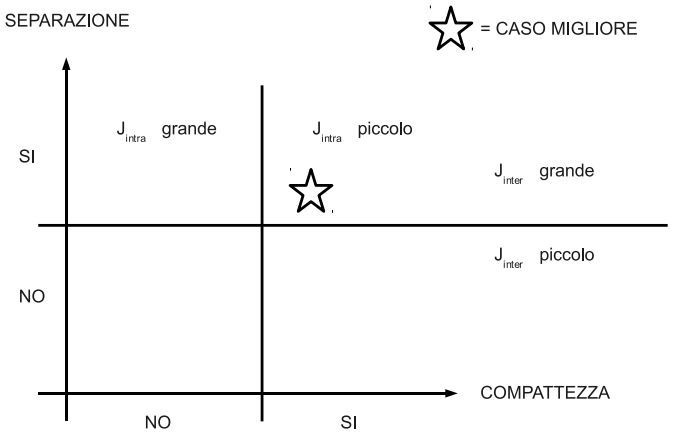
La variante $k-$means++ cerca proprio di scegliere meglio i cluster di partenza.

K-means non supervisionato Ovviamente, k l'abbiamo definito noi; una variante del $k-$means in grado da definire i gruppi da sè è detta **non supervisionata.**

Per definire un gruppo, vogliamo che sia:

- compatto al suo interno;
- separato dagli altri gruppi.

Definiamo quindi due misure, J_{intra} (distanza media tra gli elementi del gruppo ed il loro centroide) e J_{inter} (distanza media tra tutte le coppie di centroidi)



Adesso vogliamo massimizzare

$$J = \frac{\sum J_{inter}}{J_{intra}}$$

quindi potrei applicare l'algoritmo di prima con $k = 2$, poi con $k = 3$, e così via. Se J aumenta sto migliorando, viceversa sto peggiorando. In pratica massimizzo J al variare di k.

11.2 Statistica di Hopkins

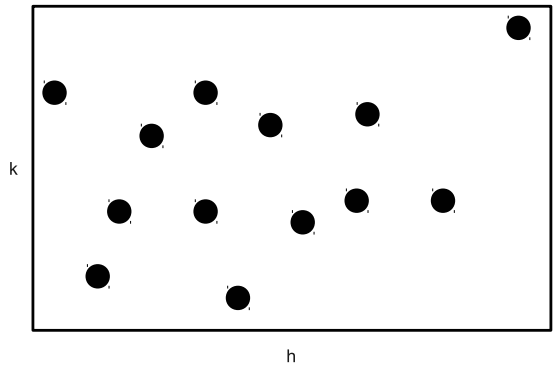
Dato un gruppo di dati, ci possiamo porre delle domande sulla validità di una loro eventuale clusterizzazione. Ad esempio:

- esiste una struttura nella collezione?
- i gruppi quanti sono?

Per rispondere a domande di questo tipo si ricorre ad un test statistico: il **test di randomicità di Hopkins**.

Vediamolo in 2 dimensioni:

sia $h \times k$ il più piccolo rettangolo che contiene i dati:



Scegliamo a caso m luoghi (punti) dentro il rettangolo, con $m \ll \#$ dati, denominandoli $y_1, \ldots, y_m$.

Sorteggiamo m punti tra i dati $x_1, \ldots, x_n$.

Calcoliamo

$$H = \frac{\sum_{j=1}^{m} u_j^d}{\sum_{j=1}^{m} u_j^d + \sum_{j=1}^{m} v_j}, \text{ per cui } H \in [0,1]$$

con

- $d =$ dimensione (nel nostro caso 2);
- $u_j =$ minima distanza di y_j da un altro punto y qualunque;
- $v_j =$ minima distanza di x_j da un altro punto x qualunque.

Se $H = 1/2$ la struttura è randomica; al tendere di H verso uno dei due estremi, ci sarà una struttura e sarà opportuno dividere il gruppo.

11.3 Fuzzy e clustering

La logica fuzzy ammette infiniti gradi di verità (un qualsiasi valore in $[0,1]$), non solo 0 (falso) o 1 (vero) come la logica CRISP.

Tre semplici regole:

1. se abbiamo P vera in grado x, allora $\neg P$ avrà grado $1 - x$;
2. se P ha grado di verità $t(P)$ e Q ha grado di verità $t(Q)$, allora
$$P \wedge Q = \min(t(P), t(Q))$$
3. analogamente
$$P \bigvee Q = \max(t(P), t(Q))$$

Alla fine bisogna comunque prendere una decisione, per cui si passa attraverso un processo di de-fuzzyficazione, che in alcuni casi è molto semplice (a volte basta prendere il massimo) ma potrebbe essere anche complesso (potrebbero esserci, regole logiche, elementi correlati, vincoli, ecc.).

Un esempio: sistemi industriali con caldaie correlate; "se la caldaia 1 è calda e la 2 pure, apri la valvola C"; ma QUANTO devono essere calde? Se mettiamo come valore 180°, anche a 178° è calda, ci vuole un po' di elasticità.

A noi la logica fuzzy interessa per la classificazione, per cui vediamo un altro esempio: classificare in base all'altezza in ALTO e BASSO.

Nella logica classica, dato un universo di elementi Ω e definito $P(\Omega)$ l'insieme delle parti di Ω, per ogni sottoinsieme di Ω possiamo definire una funzione caratteristica (o di appartenenza) χ_A che ci dice quali elementi di Ω appartengono ad A:

$$A \subseteq \Omega \Longleftrightarrow \chi_A : \Omega \to \{0,1\}$$

con

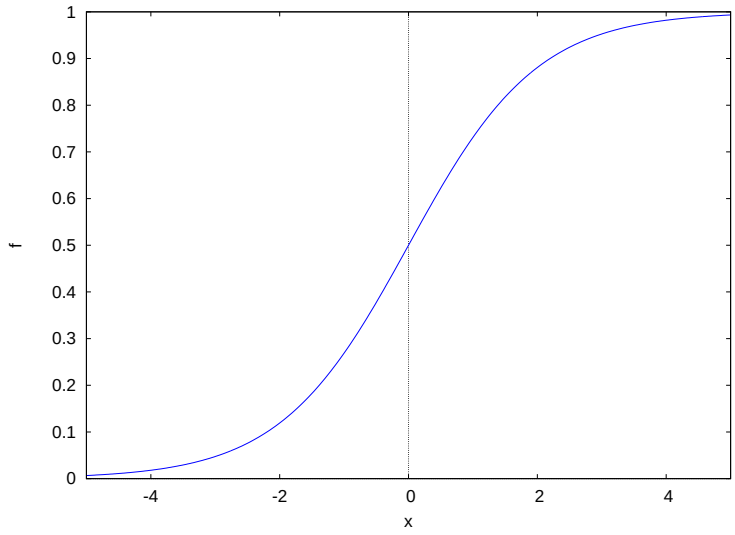
$$\chi_A(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A \end{cases}$$

Nel caso ALTO/BASSO, potremmo definire ad esempio:

$$\begin{aligned} ALTO &= \{x : x \in \Omega \wedge altezza(x) \geq 170\} \\ BASSO &= \{x : x \in \Omega \wedge altezza(x) < 170\} \end{aligned}$$

ma è evidente che questo approccio è troppo rigido (un individuo alto 169 cm. andrebbe nella stessa classe di un individuo alto 150 cm.): una differenza insignificante porta a differenze enormi in fase di classificazione!

La proposta fuzzy è quella di sfumare la funzione, per cui ad esempio $173\,cm \Rightarrow$ "alto 0.6".



Considerando più features, potremmo fare confronti più interessanti (ad esempio, dire che una donna giapponese di 16 anni alta 165 cm. è "ALTA" e classificare, invece, uno svedese di 28 anni alto 170 cm. come "BASSO"... tutto dipende da come definiamo regole, vincoli, pesi e funzione di de-fuzzyficazione).

Chiaramente

$$\begin{aligned} ALTO\,(x) + \neg ALTO\,(x) &= 1 \\ BASSO\,(x) + \neg BASSO\,(x) &= 1 \end{aligned}$$

Per certe features, non ha senso mettere dei gradi (e quindi non lo facciamo): ad esempio, gli occhiali.

NOTA: non confondere le curve della probabilità "classica" con le curve Fuzzy: non è solo un problema di valore dell'area (che potrebbe essere norm.alizzata) ma proprio concettuale. Tornando all'esempio di salmoni/seabass e la loro lunghezza:

nel caso della probabilità classica il valore della curva è compreso tra 0 e 1 e rappresenta la probabilità (appunto)

nel caso della funzione fuzzy di appartenza rappresenta un "*grado di salmonitudine*" compreso tra 0 e n.

CONVENZIONE:

Scriveremo le funzioni caratteristiche χ in versione fuzzy come $\tilde{\chi}$ (ad esempio $\tilde{\chi}_{ALTO}$).

Dato A, se $\tilde{\chi}_A : \Omega \to [0,1]$, allora $\Omega - A = \bar{A}$ e $\tilde{\chi}_{\bar{A}} = 1 - \tilde{\chi}_A$;

inoltre, $\tilde{\chi}_{A \cup B} = \max(\tilde{\chi}_A, \tilde{\chi}_B)$ e $\tilde{\chi}_{A \cap B} = \min(\tilde{\chi}_A, \tilde{\chi}_B)$.

Non parleremo in maniera approfondita della de-fuzzyficazione e altri argomenti della logica fuzzy.

In certe situazioni, può tornare utile il concetto di gradualità: dunque è lecito cercare versioni fuzzy della divisione in gruppi (clustering) $\Rightarrow$ fuzzy-clustering di un insieme di dati Ω.

Per ogni $x \in \Omega$, per ogni classe $S_1, \ldots, S_k$, definisco $\chi_{S_i}(x) \in [0,1]$

Attenzione:

$$x \to \begin{cases} \chi_{S_1}(x) \\ \vdots \\ \chi_{S_k}(x) \end{cases}$$

Devo porre qualche vincolo su $\sum_{i=1}^{k} \chi_i(x)$?

E' una vera e proprio questione di "scuola di pensiero":

Algorithm 9 Algoritmo $c-$means fuzzy

1. INIZIALIZZAZIONE
   assegnare μ_{ij}, anche in maniera casuale, ma sempre rispettando il vincolo.
2. CALCOLO DEI CENTROIDI DELLE CLASSI
$$C_J = \frac{\sum_{i=1}^{N} \mu_{iJ} x_i}{\sum_{i=1}^{N} \mu_{iJ}}$$
3. AGGIORNAMENTO DEI GRADI DI APPARTENENZA
$$\mu'_{ij} = \frac{1}{\sum_{classi} \left[\frac{d(x_i, C_J)}{d(x_i, C_h)}\right]^{\frac{2}{m-1}}}$$

- alcuni pongono il vincolo aggiuntivo "$= 1$" (più semplice);
- altri, detti "possibilisti", non pongono vincoli (la loro argomentazione è che se ho un sintomo non per forza la probabilità di avere la malattia A deve salire e la probabilità di avere la malattia B deve scendere, possono anche salire o scendere entrambe (e non fare 1)).

11.4 Fuzzy $c-$means

Abbiamo visto che nel $k-$means si ha:

$$J = \sum_{classi\, C} \sum_{x \in C} d\left(x, C_j\right)^2$$

dove C_j è il baricentro della classe, il centro del gruppo.

Nella logica fuzzy, come sappiamo, per ogni elemento si hanno dei gradi di appartenenza alle varie classi; preliminarmente, quindi, definiamo per gli elementi $x_1, \ldots, x_N$ i valori di appartenenza μ_{ij}, con $i = 1 \ldots N$ e $j = 1 \ldots k$ (# classi).

μ_{ij} è quindi il grado di appartenenza di x_i alla classe j.

IMPORTANTE: vale qui il vincolo

$$\sum_{j=1}^{k} \mu_{ij} = 1, \ \forall i = 1 \ldots N$$

Supponiamo di avere assegnato a ciascun elemento i valori di appartenenza in maniera arbitraria (non rispettando il vincolo per classi); allora l'approccio di Bezdeck è:

$$J_{fuzzy} = \sum_{classi\, C} \sum_{x \in C} \mu_{ij} \cdot d\left(x, C_j\right)^2$$

dove

- m è un parametro di classificazione fuzzyficante, con vincolo $m > 1$ (in genere, si pone $m = 2$); a valori alti di m corrispondono classificazioni più nette;
- $\mu =$ grado di appartenenza, con valore $\in [0, 1]$;
- $d\left(x, C_j\right)^2$ definisce, in generale, il valore di dispersione del cluster.

Adesso vediamo l'algoritmo

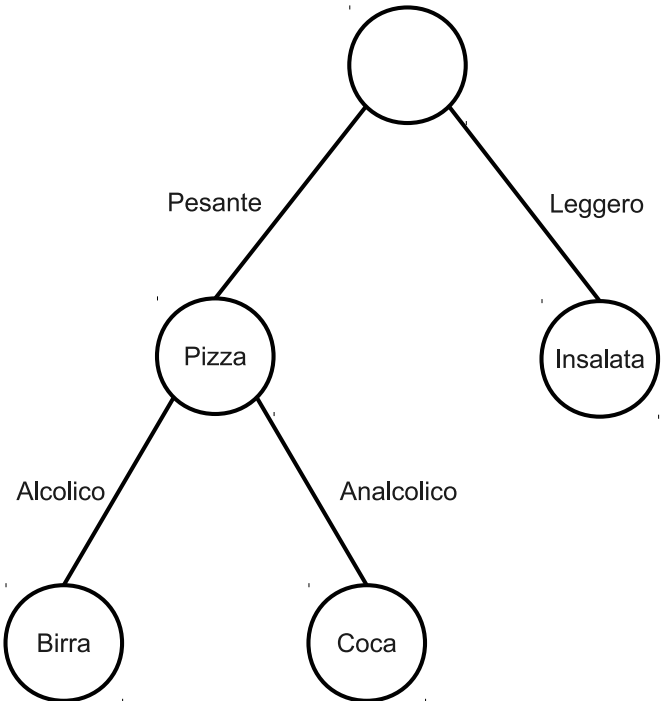
Al passo 3, l'esponente $\frac{2}{m-1}$ "sfuma" la classificazione e mantiene il vincolo.

$c-$means e $c-$means fuzzy utilizzano un modello già visto: lo schema E-M.

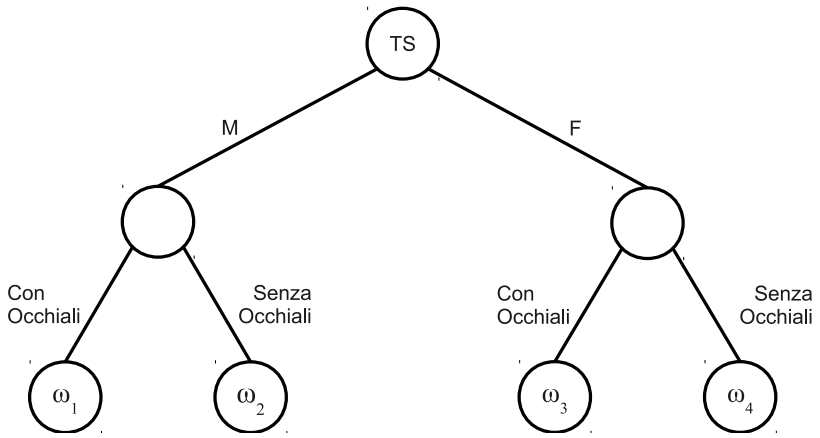
11.5 Metodi senza metriche

I metodi $k-$means e fuzzy $c-$means utilizzano metriche per il calcolo delle distanze. Esistono anche metodi che non prendono in considerazione le distanze, ad esempio il metodo dell'**albero di decisione**.

Partiamo da un esempio:



Insomma si parte da un TS e lo si suddivide, ad esempio, in due categorie ω_1 e ω_2, e così via.



A seconda del tipo di problema (e di features) è possibile provare a fare delle semplificazioni, comunque il principio di base è questo; ovviamente, si tratta di un approccio supervisionato.

Un algoritmo basato sugli alberi di decisione è, ad esempio, **CART** (Classification And Regression Trees).

12 Algoritmi basati su alberi

Algoritmi supervisionati: CART, Algoritmi di Quinlan; basati su alberi o foreste di decisione.

Algoritmi non supervisionati: dendrogramma; basato su alberi gerarchici.

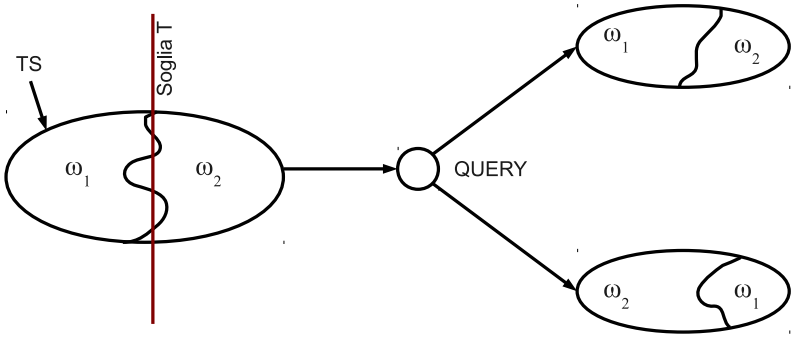
12.1 Alberi di decisione - CART

Negli algoritmi CART, ad ogni passo si sceglie una feature da "interrogare" (fase della query) con una soglia (dati numerici) e si creano due nuovi sottoinsiemi.

Alla fine si otterrà un albero binario (potenzialmente non completo) del quale ci interessano le foglie; alle varie foglie assegneremo quindi delle etichette ω_i secondo un certo criterio.

La profondità dell'albero è difficile da prevedere (una possibile stima è $\log n$ per n elementi, ma ciascuna feature può essere utilizzata più volte, se conviene, nel corso delle suddivisioni).

Gli insiemi ottenuti nel corso delle suddivisioni possono però presentare delle impurità:



Esistono dei criteri per misurare le impurità:

Entropia (Shannon)

Dati i due insiemi ω_1 e ω_2 e dette P_1 e P_2 le frequenze degli elementi del 1° e 2° insieme, allora

$$i(N) = -P_1 \log_2 P_1 - P_2 \log_2 P_2$$

In generale dati ω_i insiemi con elementi di frequenza P_i,

$$i(N) = -\sum_i P_i \log_2 P_i$$

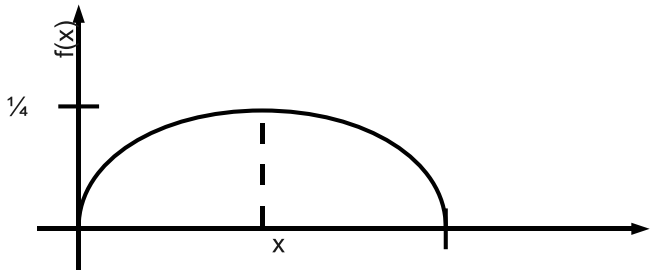
La misura dell'entropia di Shannon può essere rappresentata graficamente da un arco di parabola con la concavità verso il basso, passante per l'origine e per il punto $(1, 0)$ e con vertice in $\left(\frac{1}{2}, 1\right)$.

Impurità di varianza Si applica solo al caso in cui abbiamo due classi, per cui

$$P_1 \cdot P_2 = P_1 \cdot (1 - P_1)$$

La formula è del tipo

$$f(x) = x \cdot (1 - x) = x - x^2$$



Misura di impurità di Gini

$$\sum_{i \neq j} P_i P_j = 1 - \sum_{classi\, i} P_i^2$$

Misclassification

$$1 - \max_i (P_i)$$

Quindi a seconda della suddivisione effettuata l'impurità varia. Il nostro obiettivo è quello di ridurre al minimo l'impurità. Un approccio "forza bruta" per risolvere questo problema è chiaramente improponibile, se non per problemi molto semplici, per cui si fa ricorso ad un criterio greedy:

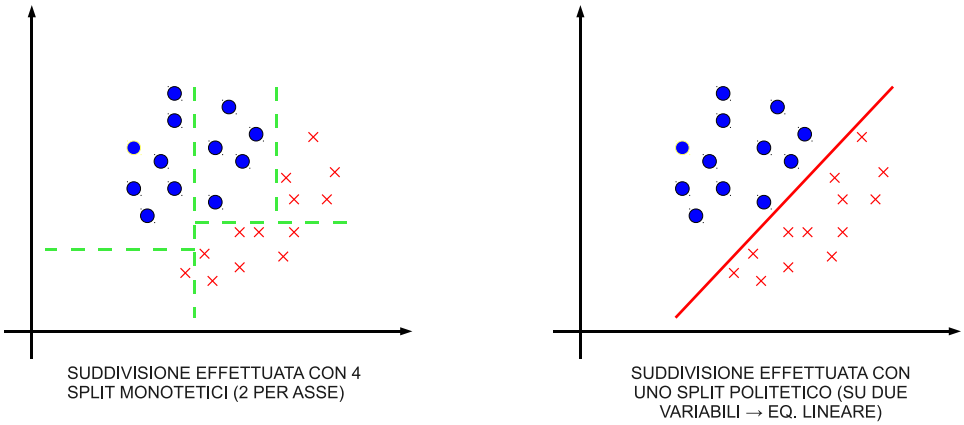
SCEGLIERE TRA TUTTI I POSSIBILI SPLIT QUELLO CHE DECREMENTA MAGGIORMENTE L'IMPURITÀ DELLE FOGLIE DELL'ALBERO

Va detto che gli split possibili sono comunque molti, per cui il più delle volte anche questo approccio greedy si risolve con una ricerca esaustiva.

Definizione 12.1. Split

Uno split è detto **monotetico** se relativo ad una sola feature, altrimenti è detto **politetico**.

Gli split politetici sono più potenti ma anche più difficili da trovare (nel senso di calcolo).



Criterio di arresto per la generazione di alberi Quando fermarsi nel processo di splitting?

Si parla di **criterio di arresto**, che si verifica in questi casi:

1. Quando ogni foglia è pura.
   Problemi:
   - albero complesso;
   - overfitting.
2. Quando l'impurità scende al di sotto di un valore di soglia prefissato.
3. Quando il numero di elementi in una foglia è minore di una certa quantità prefissata.

A volte si usa una combinazione dei punti 1, 2 e 3.

Record incompleti In presenza di record incompleti, si può decidere di:

- ignorare i record incompleti (idea "dispendiosa" in certi casi, per cui la scartiamo);
- dividere ulteriormente l'albero (rendendolo quindi ternario) per depositare lì i nodi ignoti (a seconda dei casi, si potrà decidere se ignorarli o trattarli con politiche *ad hoc*, a seconda dell'algoritmo).

Test statistico - tecnica χ^2 Sia $\Omega = \omega_1 \cup \omega_2, \omega_1 \cap \omega_2 = \emptyset$

1. Suddividiamo Ω in A e B seguendo un certo criterio:
2. Suddividiamo poi Ω in C e D a caso.

Se applicare 1 dà gli stessi risultati di applicare 2, tanto vale scegliere a caso piuttosto che applicare un criterio.

Valore χ^2:

$$\chi^2 = \frac{n_{1-split} - n_{random-split}}{n_{random-split}} + \frac{n_{2-split} - n_{random-split}}{n_{random-split}}$$

Se χ^2 è sufficientemente grande, la query ha significativamente isolato gli elementi di ω_1 nella classe A e quindi vale la pena di seguire il criterio, altrimenti è inutile. Se nessuna delle query dà un buon valore di χ^2 allora applicheremo un approccio random.

NOTA: i vari metodi di misura dell'impurità sono equivalenti, quello che conta è il criterio di arresto.

NOTA SULLA NOTA: gli alberi possono diventare estremamente profondi e complessi, per questo o si scelgono criteri di arresto molto rigidi o si scelgono criteri blandi cercando però di sistemare le cose dopo.

Strategie:

1. mantenere l'albero semplice in fase di training, cioè imporre limiti stringenti per decidere di fare uno split.
2. **Pruning:** accettare facilmente gli split e semplificare a posteriori l'albero risultato

12.2 Alberi di decisione - Algoritmi di Quinlan

Nella loro "forma base" i CART presentano diversi problemi:

- se ci sono molti nodi ignoti, l'albero può diventare molto grande (pesante), quindi profondo e confusionario;
- i dati, o meglio le features, non sempre sono di tipo numerico (dove è sufficiente una soglia T per avere un criterio dicotomico) ma possono essere anche di tipo *categoriale* (ad es: capelli rossi, neri, biondi, ecc.), per i quali tra l'altro non esiste un criterio di ordinamento/confronto.

Algorithm 10 ID3

1. Si prenda un insieme S ed un set di features F.
   (a) se S è puro rispetto alle classi $\Rightarrow STOP$
   (b) altrimenti, per ciascuna feature $f \in F$,
      i. si calcoli il *gain di informazione (vedi sotto)* che si avrebbe partizionando S in $S_1 \dots S_k$ (se f ha k possibili valori);
      ii. si selezioni la feature che fornisce il gain maggiore e la si rimuova da F.
2. Si ripeta ricorsivamente su $S_1 \dots S_k$.

Quinlan ha proposto diverse varianti/soluzioni per questi problemi, come la famiglia di algoritmi ID.

Prendiamo ad esempio ID3, che è un algoritmo di classificazione basato su alberi di decisione di arietà multipla (non per forza alberi binari) che si adatta bene ai dati categoriali.

ID3 usa ciascuna feature una sola volta in un percorso radice-foglia; in particolare, ecco come funziona:

Definizione 12.2. Gain di informazione

$$gain = E(S) - \sum \frac{N_i}{N} E(S_i)$$

cioè deriva da una media pesata di E, l'entropia; il tutto deriva dal fatto che l'entropia E di S vale:

$$E(S) = -\sum_i p_i \log_2 p_i$$

con i =indice delle classi da selezionare; quindi

$$S_1 \dots S_k \Rightarrow E(S_1) \dots E(S_k)$$

e statisticamente a $|S|$ grande corrisponde $E(S)$ grande, per cui bisogna correggere questo aspetto.

Le foglie possono risultare un po' impure; a seconda del livello di impurità, si può scegliere di etichettare una foglia come "dubbia".

L'algoritmo ID3 va bene per tipi di features categoriali, ma può sempre presentarsi il problema di features "miste" (alcune categoriali, altre numeriche); lo stesso Quinlan ha quindi elaborato un'altra famiglia di algoritmi, la famiglia C, per risolvere questi problemi.

L'ultimo algoritmo pubblicato sulla famiglia C è $C4.5$, mentre i successivi (da $C5$ in poi) sono proprietari.

L'algoritmo $C4.5$ fa uso di:

- partizioni binarie sui dati numerici;
- partizioni multiple sulle categorie;
- policy molto elaborate per trattare i dati mancanti.

NOTA FINALE: CART, ID3, ,$C4.5$ sono tutti algoritmi supervisionati; non c'è bisogno di un TS di partenza per poter valutare la purezza.

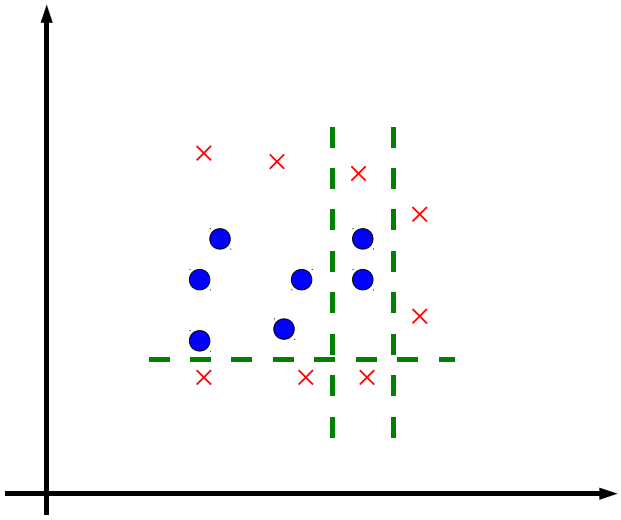
12.3 Decision forest

Piuttosto che lavorare con un unico grande albero, si può decidere di lavorare con una foresta di alberi, più facile da gestire e che si presta bene al calcolo parallelo. La differenza è nota come *"leoni contro formiche"*:

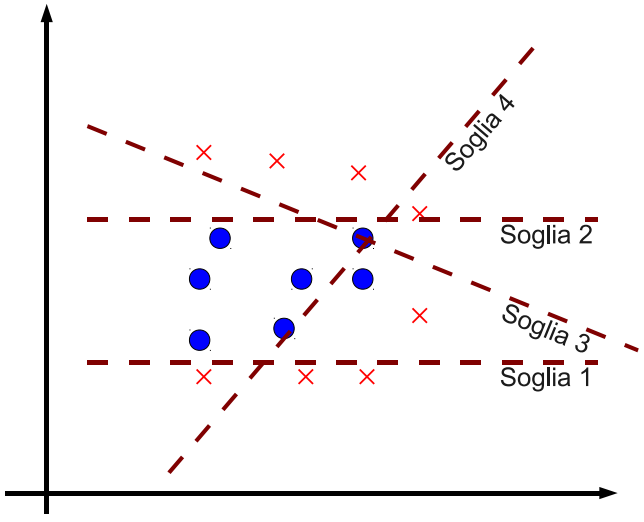
- leoni ⇒ pochi e forti ⇒ alberi di decisione;
- formiche ⇒molte e deboli ⇒ foreste di decisione.

Gli algoritmi "formiche" sono noti anche come algoritmi di **meta-classificazione**.

Dato un set di osservazioni, con CART otteniamo un risultato come quello in figura



Possiamo provare un altro approccio, impostando una soglia e cercando un albero di decisione su tale suddivisione, e poi ancora una seconda soglia (ad esempio: sulla diagonale principale ⇒ approccio "generico", non mirato, che lascia delle impurità in ciascun albero della foresta), e così via.

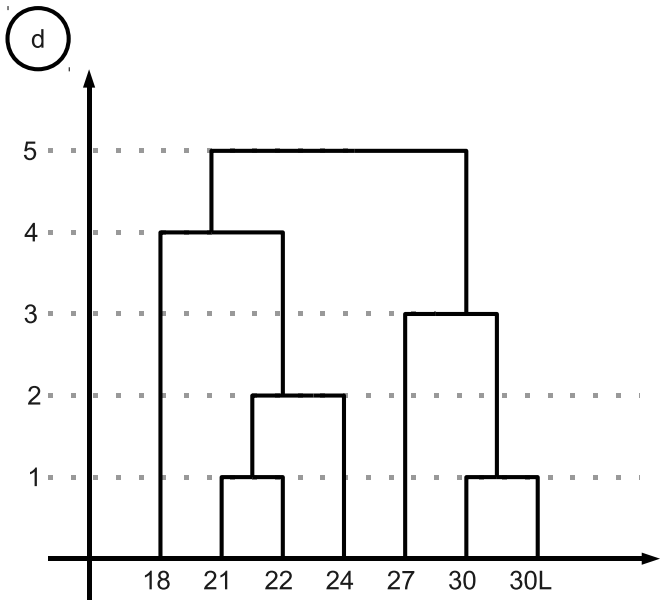


Nessuno degli alberi ricavati dalle varie soglie è perfetto, in quanto tutti avranno delle impurità, ma nell'insieme essi forniscono uno strumento piccolo ed agile, più facile da gestire (anche in parallelo) a livello algoritmico.

12.4 Alberi gerarchici - Dendrogramma

L'idea di base è quella di organizzare i dati in base ad una *pseudo-distanza*, una misura di similarità tra i record.

Prendiamo ad esempio il caso dei voti di alcuni studenti (unidimensionale): possiamo associare dei pesi e fondere per grado di similarità 1, scegliendo come rappresentante di una fusione la media dei valori e ripetendo tale procedimento fino ad arrivare ad un unico progenitore, comune a tutti i valori; la struttura grafica così ottenuta è detta **dendrogramma**.



A questo punto, per capire quanto sono dissimili tra loro i gruppi, possiamo ricavare una metrica d' (che, a differenza di d, sarà una vera distanza), così definita:

$d'(x, y) =$ lunghezza, in passi, del percorso dalla foglia x alla foglia y passando per il primo progenitore comune

Un albero gerarchico raduna, quindi, gruppi di dati nelle foglie e nei nodi intermedi.

Questo algoritmo è di tipo *bottom-up* ed il suo "motore" è stato studiato per la prima volta da Kruskal per il problema dei MST:

connettere i vertici di un grafo qualunque creando un grafo aciclico di peso minimo(calcolato come somma dei pesi per ogni arco del cammino). In generale, comunque, il nostro problema è quello di creare dei gruppi, non di creare alberi minimi di connessione.

L'algoritmo di base è il seguente:

Algorithm 11 Clustering basato su alberi gerarchici

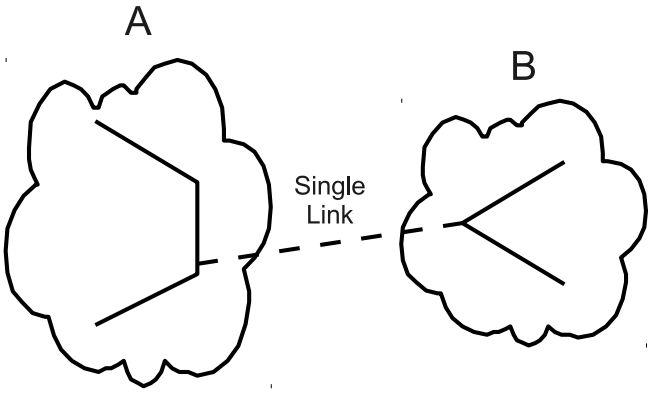
1. Si selezionino i due cluster più prossimi (vedi in seguito).
2. Si fondano questi due cluster.
3. C'è un solo cluster?
   (a) SI: ci si fermi;
   (b) NO: si torni al punto 1.

Per definire la distanza tra cluster abbiamo tre scelte:

1. **single linkage;**
2. **complete linkage;**
3. **media.**

Single linkage È esattamente l'algoritmo di Kruskal: dati due cluster S_1 e S_2:

$$d(S_1, S_2) = \min_{\substack{x \in S_1 \\ y \in S_2}} d(x, y)$$



In tal modo si genera un albero. Tale algoritmo è estremamente sensibile anche a piccole variazioni nei dati.

Complete linkage In questo caso,

$$d(S_1, S_2) = \max_{\substack{x \in S_1 \\ y \in S_2}} d(x, y)$$

Non basta un solo link: per tenere il costo a 1 serve l'insieme completo dei link (chiaramente, questo algoritmo non genera un albero). Nel caso peggiore, il costo è il costo del massimo.

Media

$$d(S_1, S_2) = \sum_{\substack{x \in S_1 \\ y \in S_2}} d(x, y) \cdot \frac{1}{|S_1| \times |S_2|}$$

13 Neural Networks

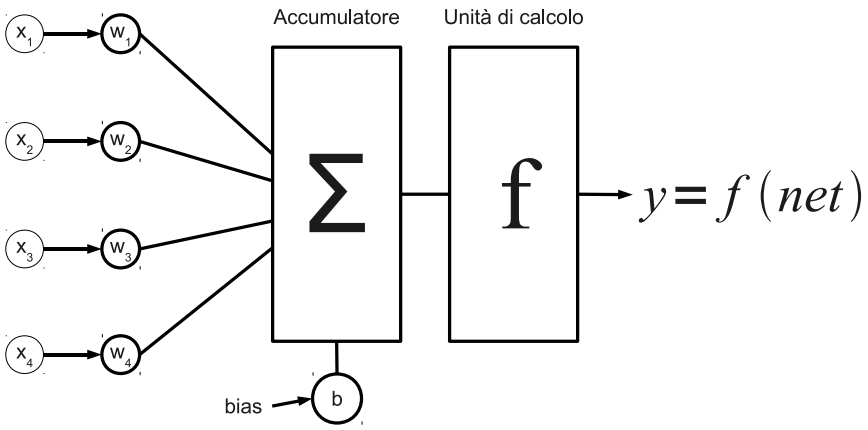
Le reti neurali (Neural Networks - NN) artificiali sono un modello che prova a simulare la struttura e le funzionalità di una rete neurale biologica (il cervello umano). Com'è ovvio, sono uno strumento non lineare di modellazione di dati.

Noi vedremo inizialmente il modello di percettrone elaborato da Rosemblatt e poi passeremo alla presentazione di una rete a livelli, con lo studio dell'algoritmo di backpropagation.

13.1 Percettrone di Rosemblatt

Un percettrone è un modello matematico che simula un neurone animale.

Ecco una rappresentazione grafica del percettrone:



Il funzionamento è il seguente:

i valori x_i vengono pesati dai pesi w_i e sommati nell'accumulatore insime al *bias*. Il risultato, chiamato *net*, viene passato all'unità di calcolo che dopo l'applicazione di una funzione f restituisce un valore in output. In formule avremo

$$\begin{aligned} net = \sum_i w_i x_i &= \langle w, x \rangle = \vec{w} \cdot \vec{x} \\ y &= f(net) \end{aligned}$$

Quindi un percettrone è una formula matematica!

Le funzioni f più usate sono:

- identità: $y = x$;
- costante: $y = k$;
- lineare: $y = a \cdot net + b$;
- soglia: $y = \begin{cases} 1 & se\ net > 0 \\ 0 & se\ net \leq 0 \end{cases}$;
- sigmoide: $\frac{e^x}{1+e^x}$.

Il *bias* modella l'attività elettrica autonoma del neurone. Quindi

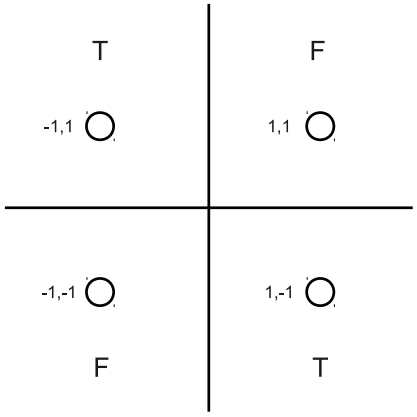
$$\begin{aligned} net &= bias + \sum_i w_i x_i \\ y &= f(net) \end{aligned}$$

Nel modello originale, Rosemblatt utilizzava la funzione soglia, quindi il percettrone calcola

$$y = sign(net)$$

La difficoltà nella costruzione del percettrone è la scelta del *bias* b e dei pesi w_i.

Inoltre il percettrone non è strutturalmente in grado di classificare alcuni pattern, come la cd. **popolazione XOR**.



In pratica non c'è modo di tracciare una retta che separi T da F. Una soluzione potrebbe essere trovata esplodendo il problema in dimensioni maggiori o mettendo in rete più percettroni.

Algorithm 12 Perceptron learning

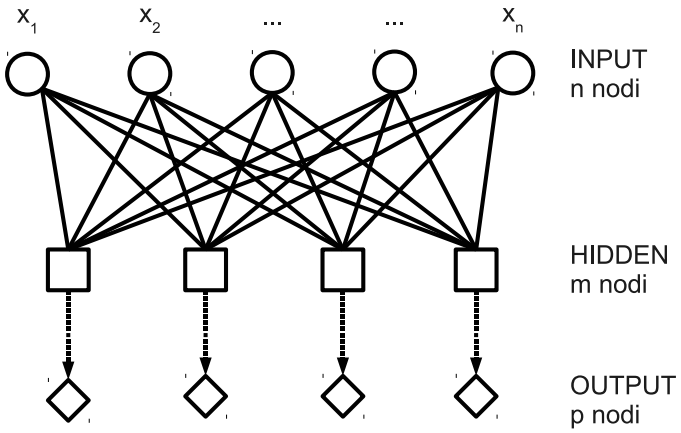
INIZIALIZZAZIONE:
si scelgano $w_i^{(0)}$ random;
si scelga b;
si scelga α - learning rate;
TS= $\{\overrightarrow{x_i}, \overrightarrow{y_i}\}$, dove $\overrightarrow{y_i}$ sono le risposte attese da un classificatore corretto.
PASSO ITERATIVO $t+1$:
si estragga a caso un elemento dal TS e lo si presenti al percettrone;
si calcoli l'output y_t;
si calcoli l'errore $E = \overline{y_t} - y_t$;
si correggano i pesi $w_i^{(t+1)} = w_i^{(t)} + \alpha E_i^{(t)} \cdot x_i^{(t)}$ - valori di α troppo elevati non fanno convergere mai l'algoritmo

13.2 Layered Neural Networks

Le reti neurali sono reti di percettroni atte a risolvere i problemi dovuti alla "semplicità" del percettrone.

Si può dimostrare che 3 strati possono risolvere qualsiasi problema risolvibile con $n > 3$ strati.

Una NN è fatta in questo modo:



INPUT: $I_1 \dots I_n$
HIDDEN: $H_1 \dots H_m$
OUTPUT: $O_1 \dots O_p$

Tutti i nodi esclusi gli I_i hanno lo stesso *bias*. Esiste quindi un nodo B che fornisce un bias costante.

Con la notazione $w_{\alpha\beta}$ sia il peso tra ogni coppia I_β, H_α che il peso tra ogni coppia H_β, O_α. Sarà chiaro dal contesto a quale dei due casi ci si riferisce. I pesi $w_{\alpha\beta}$ sono detti **pesi sinaptici**.

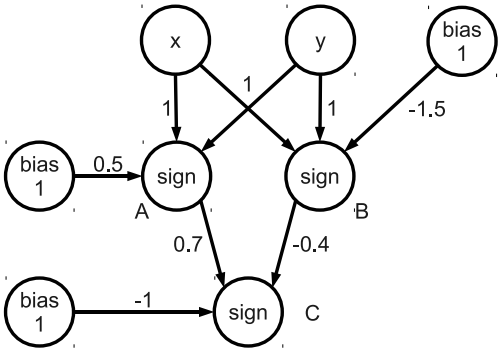
Un altro accorgimento è l'introduzione di un nodo dummy al livello input, che indicheremo con I_0. Serve solo a stabilizzare l'algoritmo.

Riassumendo, per descrivere completamente una rete occorrono:

- # nodi INPUT;
- # nodi HIDDEN;
- # nodi OUTPUT;
- valore del bias B;
- funzione f;
- pesi w_{ij}.

Solo i pesi sono parametri di ottimizzazione; tutti gli altri vengono fissati una volta per tutte all'inizio.

Grazie alle NN è possibile classificare popolazioni XOR (utilizzando la funzione soglia).



In pratica al livello hidden stiamo individuando le rette nel piano $\widehat{xy}$

$$\begin{aligned} y_A &= sign\left(x+1+\frac{1}{2}\right) \\ y_B &= sign\left(x+y-\frac{3}{2}\right) \end{aligned}$$

Al livello output ciò porterà ad individuare la striscia nel piano $\widehat{uv}$ definita da

$$z = y_C = sign\left(\frac{7}{10}u - \frac{4}{10}v - 1\right)$$

NOTA: con le NN, non c'è mai l'unicità della soluzione!!!

In questo esempio abbiamo visto la rete in modalità **forward**.

Diamo adesso alcune definizioni:

w_{ij}= peso con cui il nodo j di un livello manda $f(net_j)$ verso il nodo i del livello sottostante.

w_{ib}= peso del bias

$$\begin{aligned} z_i = f(net_i) &= f\left(bias \cdot w_{ib} + \sum_{j=1}^{k} w_{ij} y_j\right) = \\ &= f\left(b \cdot w_{ib} + \sum_{j=1}^{k} w_{ij} \cdot f(net_j)\right) = \\ &= f\left[b \cdot w_{ib} + \sum_{j=1}^{k} w_{ij} \cdot f\left(b \cdot w_{ib} + \sum_{l=0}^{n} w_{jl} x_l\right)\right] \end{aligned}$$

Formula di Kolmogorov Kolmogorov ha dimostrato che qualunque funzione può essere scritta tramite una formula strutturalmente identica alla formula per il calcolo di z_i (la funzione sarebbe la nostra f). La dimostrazione di Kolmogorov è esistenziale, cioè la f non è nota. Praticamente, semplificando estremamente, posso calcolare qualsiasi funzione continua in un intervallo $[a,b]^n$.

13.3 Tecnica di discesa del gradiente

Supponiamo di avere un TS

Se l'elaborazione della NN è corretta, produrra $z_1 = t_1, \ldots, z_n = t_n$. L'errore dipende dalla topologia della rete (l'HW non viene aggiornato però) e dai pesi, e sarà dato da

$$J(w_{ij}) = \frac{1}{2} \sum_{i=1}^{n} (t_i - z_i)^2$$

Se voglio decrementare J devo "muovere" il vettore dei pesi in direzione opposta al gradiente di f e per ogni componente calcolerò quindi

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial J}{\partial w_{ij}}$$

Omettendo le formule, distinguo due casi:

I caso: pesi dal livello hidden al livello output

$$\Delta w_{ij} = -(t_i - z_i)\, y_i f'(net_i)$$

II caso: pesi dal livello input al livello hidden

$$\Delta w_{ij} = -\sum \left[(t_i - z_i)\, f'(net_i)\, w_{ij} \right] x_j f'(net_i)$$

13.4 Modalità di training di una rete

Training on-line estraiamo i campioni uno alla volta fino a riduzioni non più apprezzabili

Batch raccogliamo tutti gli esempi e li "diamo in pasto" alla NN senza effettuare correzioni, poi consideriamo l'errore medio ed effettuiamo la correzione sull'errore medio

A epoche fissiamo $T \gg 0$ ed estraiamo T record dal TS, quindi calcoliamo l'errore medio ed effettuiamo la correzione. Poi iteriamo il procedimento.