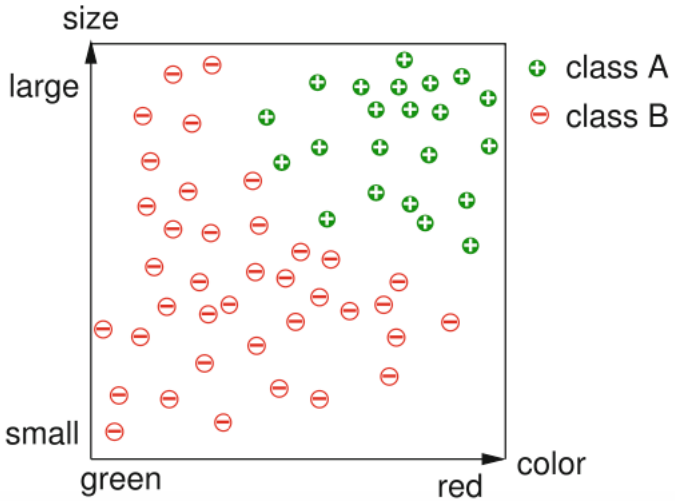
IA
22 maggio 2023
Apprendimento automatico

I concetti di generalizzazione, classificazione, approssimazione

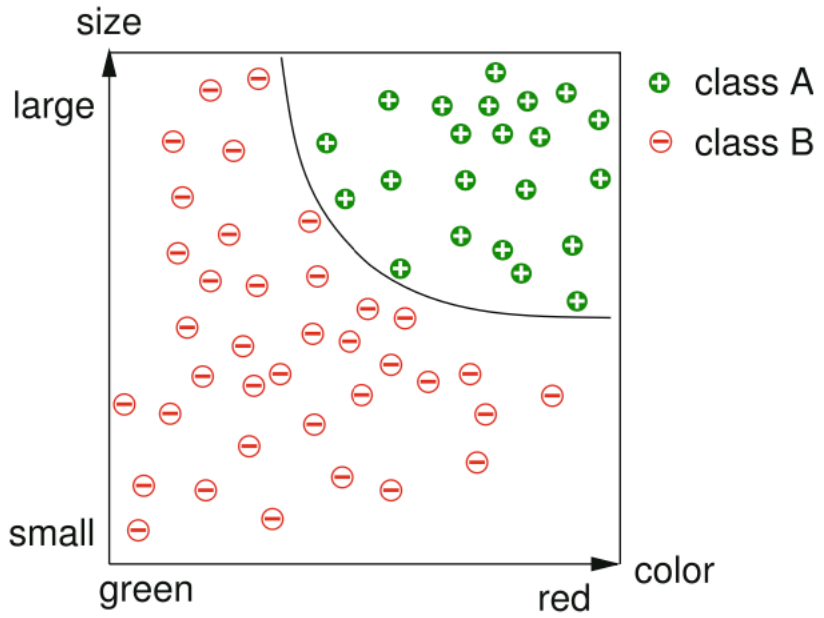
Che cos'è l'*apprendimento*? Imparare il vocabolario di una lingua straniera, o termini tecnici, o anche memorizzare una poesia può essere difficile per molte persone. Per i computer, tuttavia, queste attività sono abbastanza semplici perché sono poco più che salvare del testo in un file. Quindi la memorizzazione non è interessante per l'IA. Al contrario, l'acquisizione di abilità matematiche di solito non viene effettuata tramite la memorizzazione. Per l'addizione di numeri naturali questo non è affatto possibile, perché per ciascuno dei termini nella somma x + y ci sono infiniti valori. Per ogni combinazione dei due valori x e y, dovrebbe essere memorizzato la tripla (x, y, x + y), il che è impossibile. Questo pone la domanda: come impariamo la matematica? La risposta è: l'insegnante spiega il processo e gli studenti lo praticano sugli esempi finché non commettono più errori sui nuovi esempi. Dopo 50 esempi lo studente (si spera) capisce l'addizione. Cioè, dopo soli 50 esempi può applicare ciò che è stato appreso a infiniti nuovi esempi, che fino a quel momento non erano stati visti. Questo processo è noto come *generalizzazione*. Cominciamo con un semplice esempio.

Un frutticoltore vuole dividere automaticamente le mele raccolte nelle classi merceologiche A e B. Il dispositivo di smistamento è dotato di sensori per misurare due caratteristiche, dimensione e colore, quindi decide a quale delle due classi appartiene la mela. Questa è una tipica attività di *classificazione*. I sistemi che sono in grado di dividere i vettori di caratteristiche in un numero finito di classi sono chiamati *classificatori*.
Per configurare la macchina, le mele vengono raccolte a mano da uno specialista, ovvero classificate. Quindi le due misure vengono inserite insieme alla loro etichetta di classe in una tabella (Vedi tabella sotto. La dimensione è data sotto forma di diametro in centimetri e il colore da un valore numerico compreso tra 0 (per il verde) e 1 (per il rosso). La visualizzazione dei dati è data come punti in un diagramma a dispersione nella figura sotto.)

Size [cm]	8	8	6	3	…
Color	0.1	0.3	0.9	0.8	…
Merchandise class	B	A	A	B	…



L'obiettivo nell'*apprendimento automatico* ("machine learning") consiste nel generare una funzione dai dati raccolti e classificati che calcola il valore della classe (A o B) per una nuova mela dalle due caratteristiche di dimensione e colore. Nella figura seguente tale funzione è rappresentata dalla linea di demarcazione tracciata attraverso il diagramma. Tutte le mele con un vettore di caratteristiche in basso a sinistra della linea vengono inserite nella classe B e tutte le altre nella classe A.



In questo esempio è ancora molto semplice trovare una tale linea di divisione per le due classi. È chiaramente un compito più difficile, e soprattutto molto meno visualizzabile, quando gli oggetti da classificare sono descritti non solo da due, ma da molte caratteristiche. In pratica vengono solitamente utilizzate 30 o più funzioni. Per n caratteristiche, il compito consiste nel trovare un iperpiano $(n - 1)$-dimensionale all'interno dello spazio delle caratteristiche n-dimensionali che divide le classi nel miglior modo possibile. Una divisione "buona" significa che la percentuale di oggetti classificati erroneamente è la più piccola possibile.

Un classificatore mappa un vettore di caratteristiche su un valore di classe. Qui ha un numero fisso, di solito piccolo, di alternative. La mappatura desiderata è anche chiamata funzione

target. Se la funzione target non si mappa su un dominio finito, allora non è una classificazione, ma piuttosto un problema di *approssimazione*. Ad esempio, determinare il valore di mercato di un titolo da determinate caratteristiche è un problema di approssimazione. Di seguito introdurremo diversi agenti di apprendimento per entrambi i tipi di mappature.

Machine Learning e Data Mining

Possiamo descrivere formalmente un agente di apprendimento come una funzione che mappa un vettore di caratteristiche su un valore di classe discreto o in generale su un numero reale. Questa funzione *non* è programmata, ma nasce e si modifica durante la fase di apprendimento, influenzata dai dati di allenamento. Durante l'apprendimento, l'agente viene alimentato con i dati già classificati in precedenza da esperti umani. Successivamente l'agente costituisce una mappatura quanto più buona possibile dal vettore delle caratteristiche al valore della funzione (ad es. classe merceologica).
Possiamo ora tentare di avvicinarci a una definizione del termine "apprendimento automatico". Tom Mitchell [1] fornisce questa definizione:
L'apprendimento automatico (Machine Learning) è lo studio di algoritmi informatici che migliorano automaticamente attraverso l'esperienza.

Attingendo a questa definizione, ne diamo un'altra: quella di agente d'apprendimento (learning agent):
Un agente è un agente di apprendimento se migliora nel tempo le sue prestazioni (misurate da un criterio appropriato) su dati nuovi e sconosciuti (dopo aver visto molti esempi di addestramento).

È importante testare la capacità di generalizzazione dell'algoritmo di apprendimento su dati sconosciuti, i dati di *test*, altrimenti ogni sistema che ha appena salvato i dati di allenamento sembrerebbe funzionare in modo ottimale semplicemente richiamando i dati salvati.

Un agente di apprendimento è caratterizzato dai seguenti termini:
Compito: il compito dell'algoritmo di apprendimento è apprendere una mappatura. Ciò potrebbe essere, ad esempio, la mappatura dalla dimensione e il colore di una mela alla sua classe merceologica, ma anche la mappatura dai 15 sintomi di un paziente alla decisione di rimuovere o meno la sua appendice.
Agente: (più precisamente una classe di agenti): qui dobbiamo decidere quale algoritmo di apprendimento verrà utilizzato. Se questo è stato scelto, allora il dominio di tutte le funzioni apprendibili è determinato.
Dati di addestramento: (esperienza): i dati di addestramento (i campioni) contengono la conoscenza che l'algoritmo di apprendimento dovrebbe estrarre e apprendere. Con la scelta dei dati di addestramento è necessario assicurarsi che siano un campione rappresentativo per l'attività da apprendere.
Dati di test: importanti per valutare se l'agente addestrato può generalizzare bene dai dati di addestramento a nuovi dati.
Misura delle prestazioni: per il dispositivo di selezione delle mele, il numero di mele classificate correttamente. Ne abbiamo bisogno per testare la qualità di un agente. Conoscere la misura della performance di solito è molto più facile che conoscere la funzione dell'agente. Ad esempio, è facile misurare le prestazioni (tempo) di un corridore di 10.000 metri. Tuttavia, questo non implica affatto che l'arbitro che misura il tempo possa correre altrettanto

velocemente. L'arbitro sa solo misurare la prestazione, ma non la "funzione" dell'agente di cui sta misurando la prestazione.

Cos'è il *data mining*? Il compito di una macchina che apprende per estrarre conoscenza dai dati di addestramento. Spesso lo sviluppatore o l'utente desidera che la macchina di apprendimento renda la conoscenza estratta leggibile anche per gli esseri umani.
Sfide simili provengono dal commercio elettronico e dal Knowledge Management. Un classico problema si presenta qui: dalle azioni dei visitatori al suo portale web, il titolare di un'attività Internet vorrebbe creare una relazione tra le caratteristiche di un cliente e la classe di prodotti che interessano a quel cliente. Quindi un venditore sarà in grado di inserire annunci pubblicitari specifici per il cliente. Ciò è dimostrato in modo significativo su www.amazon.com, dove al cliente vengono consigliati prodotti simili a quelli visti nella visita precedente. In molte aree della pubblicità e del marketing, così come nella gestione delle relazioni con i clienti (Customer Relationship Management, CRM), stanno entrando in uso le tecniche di data mining. Ogni volta che sono disponibili grandi quantità di dati, si può tentare di utilizzare questi dati per l'analisi delle preferenze del cliente al fine di mostrare annunci pubblicitari specifici del cliente. Il campo emergente dell'apprendimento delle preferenze è dedicato a questo scopo.

Definizione:
Il processo di acquisizione della conoscenza dai dati, così come la loro rappresentazione e applicazione, è chiamato *data mining*. I metodi utilizzati sono generalmente presi dalla statistica o dall'apprendimento automatico e dovrebbero essere applicabili a quantità molto grandi di dati a costi ragionevoli.

Nel contesto dell'acquisizione di informazioni, ad esempio su Internet o in una intranet, il text mining gioca un ruolo sempre più importante. Le attività tipiche includono la ricerca di testo simile in un motore di ricerca o la classificazione dei testi, che ad esempio viene applicata nei filtri antispam per la posta elettronica. Una sfida relativamente nuova per il data mining è l'estrazione di informazioni strutturate, statiche e dinamiche da strutture grafiche come i social network, le reti di traffico o il traffico Internet.
Poiché le due attività descritte di machine learning e data mining sono formalmente molto simili, i metodi di base utilizzati in entrambe le aree sono per la maggior parte identici. Pertanto nella descrizione degli algoritmi di apprendimento non verrà fatta alcuna distinzione tra machine learning e data mining.

Analisi dei dati

La statistica fornisce diversi modi per descrivere i dati con parametri semplici. Tra questi ne scegliamo alcuni particolarmente importanti per l'analisi dei dati di allenamento e li testiamo su un insieme di dati medici raccolti in ambito ospedaliero. In questo set di dati di esempio, sono elencati i sintomi x_1, ..., x_{15} di N = 473 pazienti, descritti in modo conciso nella seguente tabella, con anche l'etichetta della classe, ovvero la diagnosi (appendicite positiva / negativa).

Var. num.	Description	Values
1	Age	Continuous
2	Sex (1 = male, 2 = female)	1, 2
3	Pain quadrant 1	0, 1
4	Pain quadrant 2	0, 1
5	Pain quadrant 3	0, 1
6	Pain quadrant 4	0, 1
7	Local muscular guarding	0, 1
8	Generalized muscular guarding	0, 1
9	Rebound tenderness	0, 1
10	Pain on tapping	0, 1
11	Pain during rectal examination	0, 1
12	Axial temperature	Continuous
13	Rectal temperature	Continuous
14	Leukocytes	Continuous
15	Diabetes mellitus	0, 1
16	Appendicitis	0, 1

Il paziente numero uno x^1, ad esempio, è descritto dal vettore

$$\boldsymbol{x}^1 = (26, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 37.9, 38.8, 23100, 0, 1)$$

Per ogni variabile x_i, la sua media $|x_i|$ è definita come:

$$\bar{x}_i := \frac{1}{N}\sum_{p=1}^{N} x_i^p$$

e la deviazione standard si definisce come misura della sua deviazione media dal valore medio:

$$s_i := \sqrt{\frac{1}{N-1}\sum_{p=1}^{N}(x_i^p - \bar{x}_i)^2}.$$

La questione se due variabili x_i e x_j siano statisticamente dipendenti (correlate) è importante per l'analisi dei dati multidimensionali. Ad esempio, la covarianza

$$\sigma_{ij} = \frac{1}{N-1}\sum_{p=1}^{N}(x_i^p - \bar{x}_i)(x_j^p - \bar{x}_j)$$

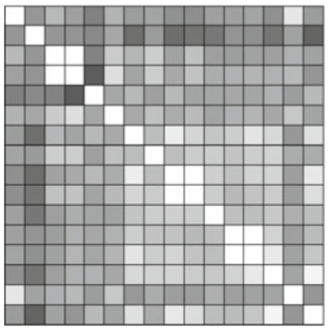
fornisce informazioni su questo. In questa sommatoria, il risultato è positivo per il *p*-esimo vettore di dati quando le deviazioni dalla media delle componenti *i*-esima e *j*-esima hanno entrambe lo stesso segno. Se hanno segni diversi, il risultato è negativo.
Tuttavia, la covarianza dipende anche dal valore assoluto delle variabili, il che rende difficile il confronto tra valori. Per poter confrontare il grado di dipendenza nel caso di più variabili, definiamo quindi il coefficiente di correlazione per due valori x_i e x_j,

$$K_{ij} = \frac{\sigma_{ij}}{s_i \cdot s_j}$$

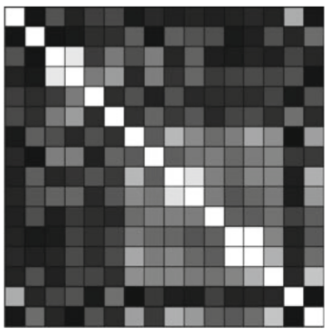
che altro non è che una covarianza normalizzata. La matrice K di tutti i coefficienti di correlazione contiene valori compresi tra -1 e 1, è simmetrica e tutti i suoi elementi diagonali hanno il valore 1. La matrice di correlazione per tutte le 16 variabili (basata su dati qui non riportati) è mostrata nella seguente tabella:

1.	−0.009	0.14	0.037	−0.096	0.12	0.018	0.051	−0.034	−0.041	0.034	0.037	0.05	−0.037	0.37	0.012
−0.009	1.	−0.0074	−0.019	−0.06	0.063	−0.17	0.0084	−0.17	−0.14	−0.13	−0.017	−0.034	−0.14	0.045	−0.2
0.14	−0.0074	1.	0.55	−0.091	0.24	0.13	0.24	0.045	0.18	0.028	0.02	0.045	0.03	0.11	0.045
0.037	−0.019	0.55	1.	−0.24	0.33	0.051	0.25	0.074	0.19	0.087	0.11	0.12	0.11	0.14	−0.0091
−0.096	−0.06	−0.091	−0.24	1.	0.059	0.14	0.034	0.14	0.049	0.057	0.064	0.058	0.11	0.017	0.14
0.12	0.063	0.24	0.33	0.059	1.	0.071	0.19	0.086	0.15	0.048	0.11	0.12	0.063	0.21	0.053
0.018	−0.17	0.13	0.051	0.14	0.071	1.	0.16	0.4	0.28	0.2	0.24	0.36	0.29	−0.0001	0.33
0.051	0.0084	0.24	0.25	0.034	0.19	0.16	1.	0.17	0.23	0.24	0.19	0.24	0.27	0.083	0.084
−0.034	−0.17	0.045	0.074	0.14	0.086	0.4	0.17	1.	0.53	0.25	0.19	0.27	0.27	0.026	0.38
−0.041	−0.14	0.18	0.19	0.049	0.15	0.28	0.23	0.53	1.	0.24	0.15	0.19	0.23	0.02	0.32
0.034	−0.13	0.028	0.087	0.057	0.048	0.2	0.24	0.25	0.24	1.	0.17	0.17	0.22	0.098	0.17
0.037	−0.017	0.02	0.11	0.064	0.11	0.24	0.19	0.19	0.15	0.17	1.	0.72	0.26	0.035	0.15
0.05	−0.034	0.045	0.12	0.058	0.12	0.36	0.24	0.27	0.19	0.17	0.72	1.	0.38	0.044	0.21
−0.037	−0.14	0.03	0.11	0.11	0.063	0.29	0.27	0.27	0.23	0.22	0.26	0.38	1.	0.051	0.44
0.37	0.045	0.11	0.14	0.017	0.21	−0.0001	0.083	0.026	0.02	0.098	0.035	0.044	0.051	1.	−0.0055
0.012	−0.2	0.045	−0.0091	0.14	0.053	0.33	0.084	0.38	0.32	0.17	0.15	0.21	0.44	−0.0055	1.

Questa matrice diventa un po' più leggibile quando la rappresentiamo come un grafico di densità. Invece dei valori numerici, gli elementi della matrice sono riempiti con valori grigi. Nel diagramma a destra vengono mostrati i valori assoluti. Così possiamo vedere molto rapidamente quali variabili mostrano una dipendenza debole o forte. Possiamo vedere, ad esempio, che le variabili 7, 9, 10 e 14 mostrano la correlazione più forte con la variabile di classe appendicite e quindi sono più importanti per la diagnosi rispetto ad altre variabili. Vediamo anche, tuttavia, che le variabili 9 e 10 sono fortemente correlate. Ciò potrebbe significare che uno di questi due valori è potenzialmente sufficiente per la diagnosi.



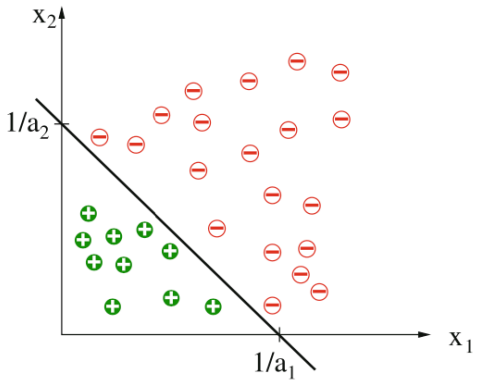
$K_{ij} = -1$: black, $K_{ij} = 1$: white



$|K_{ij}| = 0$: black, $|K_{ij}| = 1$: white

Il percettrone: un classificatore lineare

Nell'esempio di classificazione della selezione delle mele, una linea di divisione curva è tracciata tra le due classi. Un caso più semplice è mostrato nella seguente figura.



Qui, gli esempi di addestramento bidimensionali possono essere separati da una linea retta. Chiamiamo un tale insieme di dati di addestramento *linearmente separabili*. In *n* dimensioni è necessario un iperpiano per la separazione. Questo rappresenta un sottospazio lineare di dimensione *n*-1.
Perché ogni iperpiano (*n*-1)-dimensionale in R^n può essere descritto da un'equazione

$$\sum_{i=1}^{n} a_i x_i = \theta$$

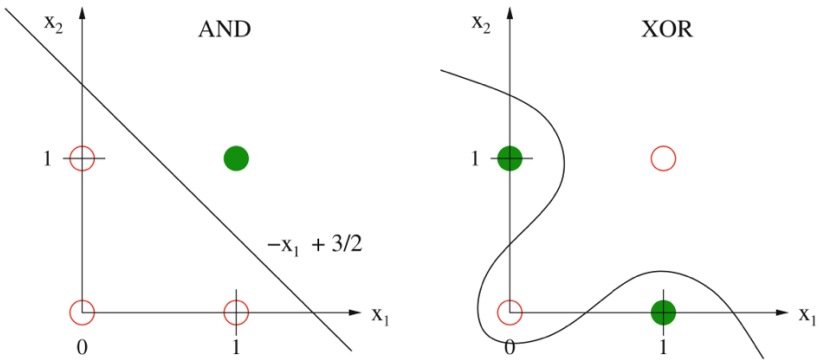
ha senso definire la separabilità lineare come segue.

Definizione
Due insiemi M_1 e M_2 (entrambi sottoinsiemi di R^n) sono detti linearmente separabili se esistono numeri reali $a_1, ..., a_n, \theta$ tali che

$$\sum_{i=1}^{n} a_i x_i > \theta \quad \text{for all } \boldsymbol{x} \in M_1 \quad \text{and} \quad \sum_{i=1}^{n} a_i x_i \leq \theta \quad \text{for all } \boldsymbol{x} \in M_2.$$

θ è chiamato il valore "soglia" (threshold).

Seguono due esempi di funzioni, una linearmente separabile (la funzione logica AND) e una no (la funzione logica XOR).

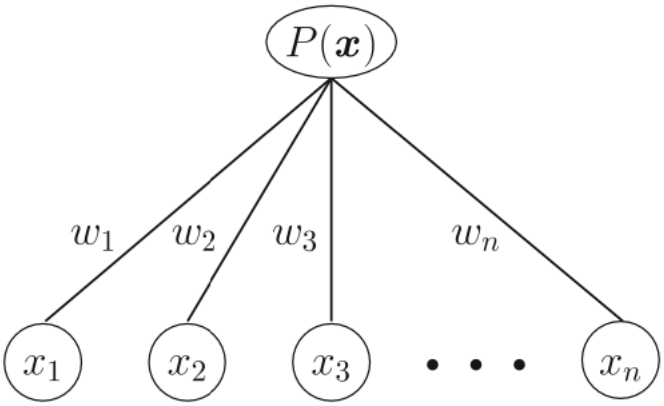


Il percettrone è un algoritmo di apprendimento molto semplice che può separare insiemi separabili linearmente.

Siano w = (w_1, ..., w_n) $\in R^n$ un vettore di pesi e x = (x_1, ..., x_n) $\in R^n$ un vettore di input. Un percettrone è una funzione P: $R^n \rightarrow \{0, 1\}$ che corrisponde alla seguente regola:

$$P(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}\,\boldsymbol{x} = \sum_{i=1}^{n} w_i x_i > 0, \\ 0 & \text{else.} \end{cases}$$

Il percettrone è un algoritmo di classificazione molto semplice. È equivalente a una rete neurale a due strati come in figura:



Per ora, tuttavia, vedremo il percettrone solo come un agente di apprendimento, cioè come una funzione matematica che mappa un vettore di caratteristiche su un valore di funzione. Qui le variabili di input x_i sono denotate come "caratteristiche" (features).
Come possiamo vedere nella formula di *P(x)*, tutti i punti x sopra l'iperpiano $\Sigma_i w_i x_i = 0$ sono classificati come positivi, e tutti gli altri come negativi. L'iperpiano di separazione passa attraverso l'origine perché $\theta = 0$.

La regola d'apprendimento
Con la notazione M_+ e M_- per indicare rispettivamente gli insiemi di dati di allenamento positivi e negativi, la regola d'apprendimento del percettrone è come segue:

```
PERCEPTRONLEARNING[M+, M−]
w = arbitrary vector of real numbers
Repeat
    For all x ∈ M+
        If w x ≤ 0 Then w = w + x
    For all x ∈ M−
        If w x > 0 Then w = w − x
Until all x ∈ M+ ∪ M− are correctly classified
```

Il percettrone dovrebbe fornire il valore 1 per tutti gli $x \in M_+$. Questo è vero quando $\boldsymbol{wx} > 0$. In caso contrario, $\boldsymbol{x}$ viene aggiunto al vettore peso $\boldsymbol{w}$, per cui il vettore peso viene cambiato esattamente nella giusta direzione. Lo vediamo quando applichiamo il percettrone al vettore modificato $\boldsymbol{w}+\boldsymbol{x}$ perché

$$(\boldsymbol{w}+\boldsymbol{x})\,\boldsymbol{x} = \boldsymbol{wx} + \boldsymbol{x}^2 > \boldsymbol{wx}$$

Se questo passaggio viene ripetuto abbastanza spesso, a un certo punto il valore $\boldsymbol{wx}$ diventerà positivo, come dovrebbe essere. Analogamente, vediamo che, per dati di addestramento negativi, il percettrone calcola un valore sempre più piccolo

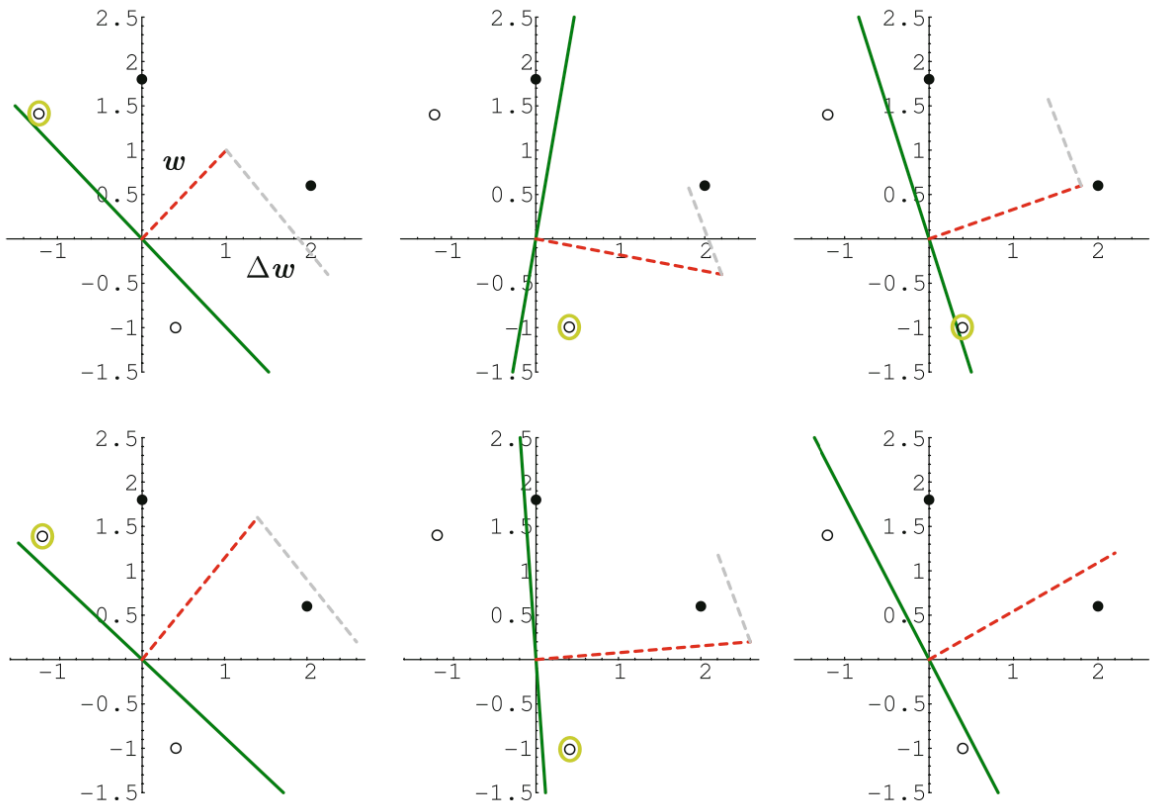
$$(\boldsymbol{w}-\boldsymbol{x})\,\boldsymbol{x} = \boldsymbol{wx} - \boldsymbol{x}^2 < \boldsymbol{wx}$$

che a un certo punto diventa negativo.

Esempio
Un percettrone deve essere addestrato sugli insiemi $M_+ = \{(0, 1.8), (2, 0.6)\}$ e $M_- = \{(-1.2, 1.4), (0.4, -1)\}$. Il vettore di peso iniziale $\boldsymbol{w}$ è (1, 1). Nella prima iterazione attraverso il ciclo dell'algoritmo di apprendimento, l'unico esempio di addestramento classificato erroneamente è (−1.2, 1.4) perché

$$(-1.2, 1.4) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0.2 > 0.$$

Ciò si traduce in $\boldsymbol{w} = (1, 1) - (-1.2, 1.4) = (2.2, -0.4)$. Dopo un totale di cinque modifiche, la linea di demarcazione si trova tra le due classi e il percettrone classifica quindi tutti i dati correttamente.



È stato dimostrato [2] che il percettrone converge sempre per dati separabili linearmente. Abbiamo che:

Teorema
Siano le classi M_+ e M_- separabili linearmente da un iperpiano $\boldsymbol{wx} = 0$. Allora PERCEPTRONLEARNING converge con qualunque inizializzazione del vettore **w**. Il percettrone *P* con il vettore peso così calcolato divide le classi M_+ e M_-, ovvero:
$P(x) = 1 \Leftrightarrow x \in M_+$
$P(x) = 0 \Leftrightarrow x \in M_-$

Come possiamo vedere chiaramente nella figura precedente, i percettroni come definiti sopra non possono dividere insiemi linearmente separabili arbitrari, piuttosto solo quelli che sono divisibili da una linea attraverso l'origine, o in R^n da un iperpiano attraverso l'origine, perché manca il termine costante θ all'equazione $\Sigma_i w_i x_i = 0$.
Con il seguente trucco possiamo generare il termine costante. Manteniamo costante l'ultimo componente x_n del vettore di input $\boldsymbol{x}$ e lo impostiamo al valore 1. Il peso w_n a $-\theta$ funziona come un valore soglia perché

$$\sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n-1} w_i x_i - \theta > 0 \quad \Leftrightarrow \quad \sum_{i=1}^{n-1} w_i x_i > \theta.$$

Nell'applicazione dell'algoritmo di apprendimento, un bit con il valore costante 1 viene aggiunto al vettore dei dati di addestramento. Osserviamo che il peso w_n (la soglia θ) viene appreso durante il processo di apprendimento.
È stato dimostrato che un percettrone P_θ: $R^{n-1} \rightarrow \{0, 1\}$

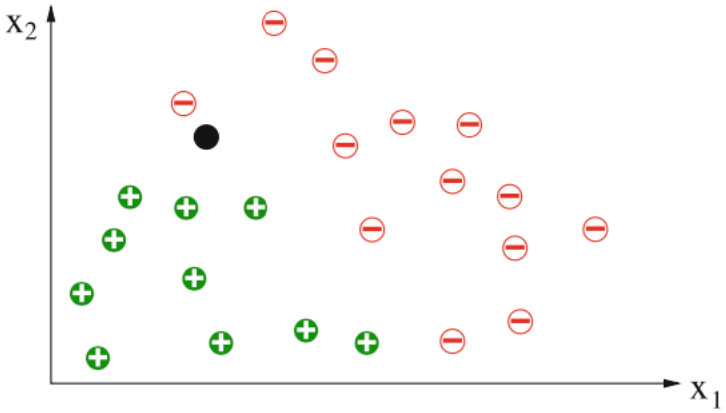
$$P_\theta(x_1, \ldots, x_{n-1}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n-1} w_i x_i > \theta, \\ 0 & \text{else} \end{cases}$$

con una soglia arbitraria θ può essere simulato da un percettrone P: $R^n \rightarrow \{0, 1\}$ con la soglia 0. Ossia,

Teorema
Una funzione f: $R^n \rightarrow \{0, 1\}$ può essere rappresentata da un percettrone se e solo se i due insiemi di vettori di input positivi e negativi sono linearmente separabili.

Il metodo Nearest Neighbor (il vicino più vicino)

Per un percettrone, la conoscenza disponibile nei dati di allenamento viene estratta e salvata in forma compressa nei pesi w_i. In tal modo le informazioni sui dati vengono perse. Ma questo è esattamente ciò che si desidera, se si suppone che il sistema faccia una generalizzazione dai dati di addestramento a nuovi dati. La generalizzazione in questo caso è un processo che richiede molto tempo e che ha l'obiettivo di trovare una rappresentazione compatta dei dati sotto forma di una funzione che classifichi i nuovi dati nel miglior modo possibile.
La memorizzazione dei dati è molto diversa e in questo caso l'apprendimento è estremamente semplice. Tuttavia, come accennato in precedenza, la conoscenza salvata non è così facilmente applicabile a nuovi esempi sconosciuti. Un tale approccio è molto inadatto per imparare a sciare, ad esempio. Un principiante non può mai diventare un bravo sciatore solo guardando video di bravi sciatori. Evidentemente, quando vengono eseguiti automaticamente movimenti di apprendimento di questo tipo, accade qualcosa di simile come nel caso del percettrone. Dopo una pratica sufficientemente lunga, la conoscenza immagazzinata negli esempi di addestramento viene trasformata in una rappresentazione interna nel cervello. Tuttavia, ci sono esempi di successo di memorizzazione in cui è anche possibile la generalizzazione. Durante la diagnosi di un caso difficile, un medico potrebbe cercare di ricordare casi simili del passato. Se la sua memoria è solida, allora potrebbe imbattersi in questo caso, cercarlo nei suoi file e finalmente arrivare a una diagnosi simile. Per questo approccio il medico deve avere una buona percezione della *somiglianza*, in modo da ricordare il caso più simile. Se lo ha trovato, allora deve chiedersi se è abbastanza simile da giustificare la stessa diagnosi.
Cosa significa somiglianza nel contesto formale che stiamo costruendo? Rappresentiamo i dati di addestramento come al solito in uno spazio delle caratteristiche (feature) multidimensionali e definiamo: minore è la loro distanza nello spazio delle feature, più due esempi sono simili. Applichiamo ora questa definizione al semplice esempio bidimensionale nella figura sotto. Qui il vicino prossimo al punto nero è un esempio negativo. Quindi il dato è assegnato alla classe negativa.



La distanza *d(**x**, **y**)* tra due punti $\boldsymbol{x} \in R^n$ e $\mathbf{y} \in R^n$ può essere misurata ad esempio dalla distanza euclidea

$$d(\boldsymbol{x},\boldsymbol{y}) = |\boldsymbol{x} - \boldsymbol{y}| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}.$$

Poiché ci sono molte altre metriche di distanza oltre a questa, ha senso pensare ad alternative per un'applicazione concreta. In molte applicazioni, alcune funzionalità sono più importanti di altre. Pertanto è spesso sensato ridimensionare le feature in modo diverso in base ai pesi w_i. La formula quindi diventa:

$$d_w(\boldsymbol{x},\boldsymbol{y}) = |\boldsymbol{x} - \boldsymbol{y}| = \sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2}.$$

Il seguente semplice programma di classificazione del vicino prossimo cerca tra i dati di addestramento il vicino ***t*** prossimo al nuovo esempio ***s*** e quindi classifica ***s*** esattamente come ***t***.

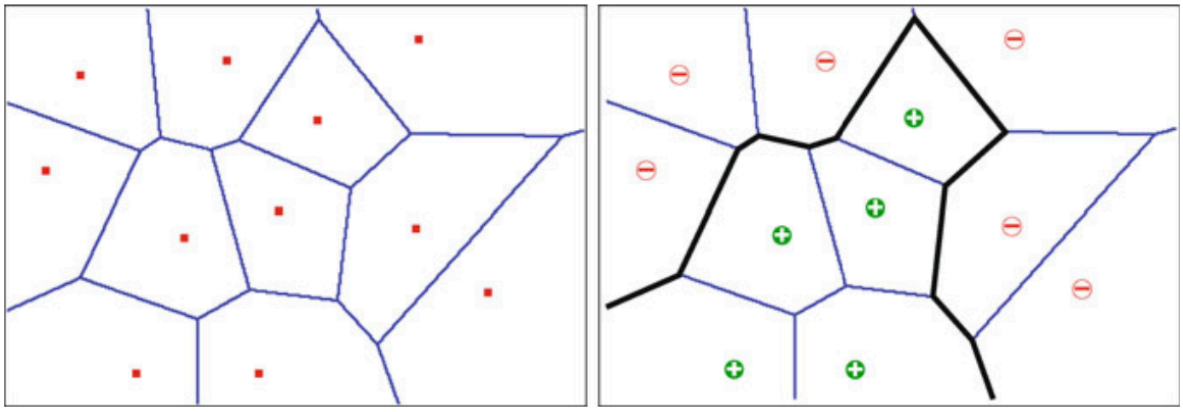
NEARESTNEIGHBOR[$M_+, M_-, \boldsymbol{s}$]
$\boldsymbol{t} = \text{argmin}_{\boldsymbol{x} \in M_+ \cup M_-}\{d(\boldsymbol{s}, \boldsymbol{x})\}$
If $\boldsymbol{t} \in M_+$ **Then Return** („+")
Else Return(„–")

I funzionali argmin e argmax determinano, analogamente a min e max, il minimo o il massimo di un insieme o di una funzione. Tuttavia, invece di restituire il valore del massimo o del minimo, danno la posizione, cioè l'input da dare alla funzione per avere quel valore estremo (minimo o massimo).

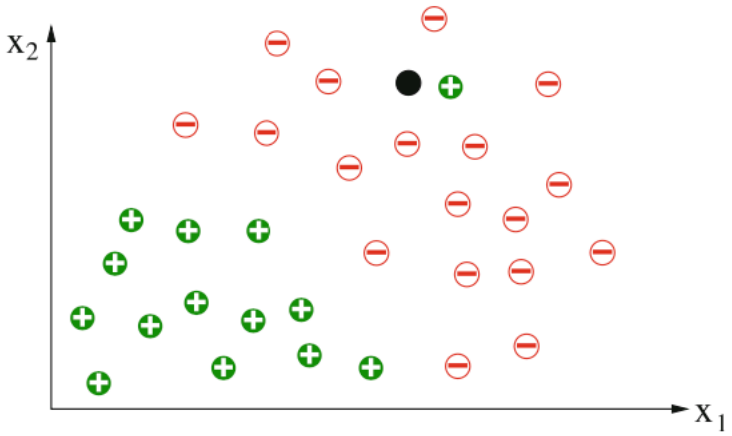
A differenza del percettrone, il metodo del vicino prossimo non genera una linea che divide i dati di addestramento. Tuttavia, esiste certamente una linea immaginaria che separa le due classi. Possiamo generarla generando prima il cosiddetto diagramma di Voronoi. Nel

diagramma di Voronoi, ogni punto è circondato da un poligono convesso, che definisce quindi un intorno attorno al punto. Il diagramma Voronoi ha la proprietà che per un nuovo punto arbitrario, il vicino prossimo tra i dati presenti è quel punto nella cui zona il nuovo arrivato si trova. Se viene determinato il diagramma di Voronoi per un insieme di dati di addestramento, è semplice trovare il vicino prossimo per un nuovo punto che deve essere classificato. La classe di appartenenza verrà presa dal vicino prossimo.

Qui sotto, un insieme di punti insieme al loro diagramma di Voronoi (a sinistra) e alla linea di divisione generata per le due classi M_+ e M_- (a destra).



In figura vediamo chiaramente che il metodo del vicino prossimo è significativamente più potente del percettrone. È in grado di realizzare correttamente linee divisorie arbitrariamente complesse (in generale: iperpiani). Tuttavia, qui c'è un pericolo. Un singolo punto errato può in determinate circostanze portare a risultati di classificazione molto negativi. Un caso del genere si verifica nella seguente figura durante la classificazione del punto nero.



Il metodo del vicino prossimo potrebbe classificare questo punto in maniera errata. Se il punto nero è immediatamente accanto a un punto positivo che è un valore anomalo della classe positiva, verrà classificato positivo anziché negativo come dovrebbe essere. Un adattamento errato a errori casuali (rumore) è chiamato *overfitting*.

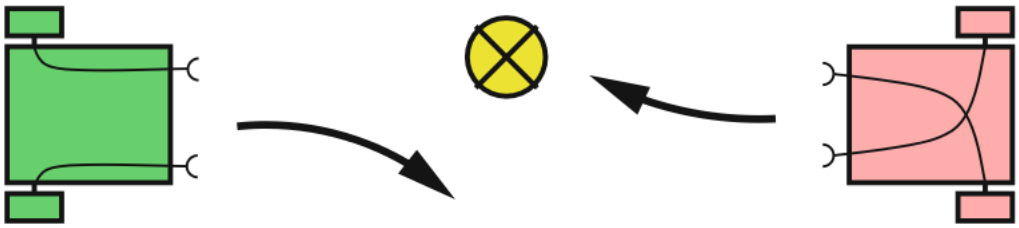
Per evitare false classificazioni dovute a singoli valori anomali, si consiglia di appianare, levigare in qualche modo la superficie di divisione. Ciò può essere ottenuto, ad esempio, con l'algoritmo K-NEARESTNEIGHBOR mostrato di seguito, che prende una decisione a maggioranza tra i *k* vicini più vicini.

```
K-NEARESTNEIGHBOR(M+, M−, s)
    V = {k nearest neighbors in M+ ∪ M−}
    If      |M+ ∩ V| > |M− ∩ V| Then    Return(„+")
    ElseIf  |M+ ∩ V| < |M− ∩ V| Then    Return(„−")
    Else                                Return(Random(„+", „−"))
```

La classificazione in più classi

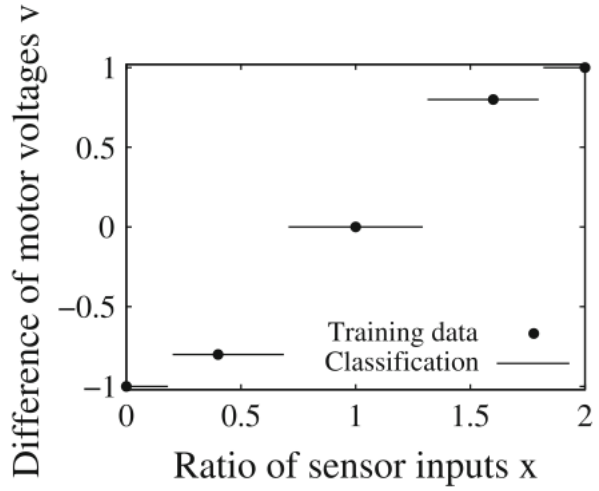
La classificazione del vicino prossimo può essere applicata anche a più di due classi. Proprio come nel caso di due classi, la classe del vettore di caratteristiche da classificare viene semplicemente impostata come la classe del vicino prossimo. Per il metodo *k*-vicini-prossimi, la classe deve essere determinata come la classe con il maggior numero di membri tra i *k* vicini più vicini.
Se il numero di classi è elevato, di solito non ha più senso utilizzare algoritmi di classificazione perché la dimensione dei dati di addestramento necessari cresce rapidamente con il numero di classi. Inoltre, in determinate circostanze, informazioni importanti vengono perse durante la classificazione. Questo diventerà chiaro nel seguente esempio.

Esempio
Un robot autonomo con sensori dovrebbe imparare ad allontanarsi dalla luce. Ciò significa che dovrebbe imparare nel modo più ottimale possibile a mappare i valori dei sensori su un segnale dello sterzo che controlla la direzione di guida. Il robot è dotato di due semplici sensori di luce sul lato anteriore.

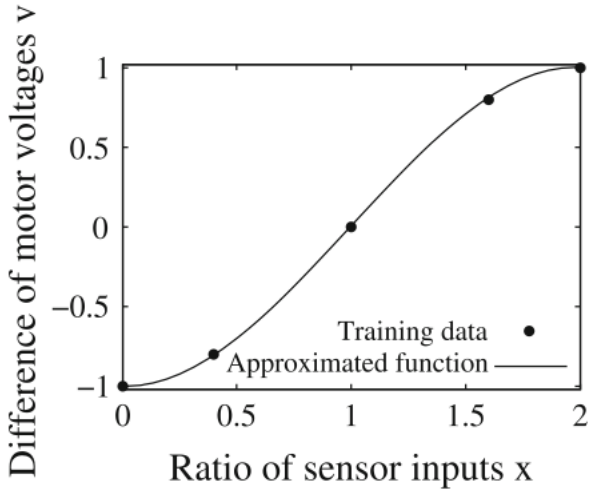


Dai due segnali dei sensori (chiamati s_l per il sensore sinistro e s_r per il sensore destro, rispettivamente), viene calcolata la relazione $x = s_r / s_l$. Per controllare i motori elettrici delle due ruote, usiamo la differenza $v = U_r - U_l$ delle due tensioni U_r e U_l dei motori destro e sinistro, rispettivamente. Il compito dell'agente di apprendimento è ora quello di evitare un segnale luminoso. Deve quindi apprendere una mappatura f che calcoli il valore "corretto" $v = f(x)$.

Per questo eseguiamo un esperimento in cui, per alcuni valori misurati *x*, troviamo un valore *v* ottimale. Questi valori sono tracciati come punti dati nella seguente figura e serviranno come dati di addestramento per l'agente di apprendimento. Durante la classificazione del vicino prossimo, ogni punto nello spazio delle caratteristiche (ovvero, sull'asse *x*) viene classificato esattamente come il suo vicino prossimo tra i dati di addestramento.



La funzione per sterzare i motori è quindi una funzione a gradini con grandi salti. Se vogliamo passaggi più precisi, dobbiamo fornire corrispondentemente più dati di addestramento. D'altra parte, possiamo ottenere una funzione continua se approssimiamo una funzione "smooth" da adattare ai cinque punti (vedi figura successiva). Richiedere che la funzione *f* sia continua porta a risultati molto buoni, anche senza punti dati aggiuntivi.
Per l'approssimazione delle funzioni sui punti dati esistono molti metodi matematici, come l'interpolazione polinomiale, l'interpolazione spline o il metodo dei minimi quadrati. L'applicazione di questi metodi diventa problematica nelle dimensioni superiori. La difficoltà speciale nell'IA è che sono necessari metodi di approssimazione senza modello. Cioè, una buona approssimazione dei dati deve essere fatta senza conoscere le proprietà speciali dei dati o dell'applicazione. Risultati molto buoni sono stati ottenuti qui con reti neurali e altri approssimatori di funzioni non lineari, che saranno discussi in seguito.



Il metodo *k*-vicino-prossimo può essere applicato in modo semplice al problema di approssimazione. Nell'algoritmo K-NEARESTNEIGHBOR, dopo che l'insieme $V = \{x_1, x_2, ..., x_k\}$ è determinato, il valore medio della funzione *k*-vicino-prossimo

$$\hat{f}(\boldsymbol{x}) = \frac{1}{k}\sum_{i=1}^{k} f(\boldsymbol{x}_i)$$

viene calcolato e preso come approssimazione $\hat{f}$ per il vettore di input $\boldsymbol{x}$. Più grande *k* diventa, più smooth è la funzione $\hat{f}$.

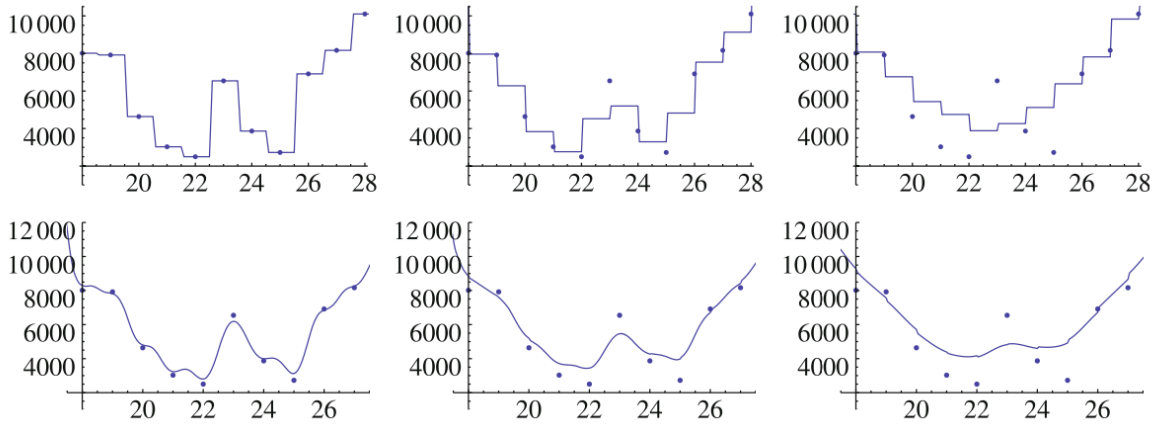
Nell'applicazione pratica di varianti discrete e continue del metodo *k*-vicino-prossimo, spesso si verificano problemi. Man mano che *k* diventa grande, tipicamente esistono più vicini con una grande distanza rispetto a quelli con una piccola distanza. In tal modo il calcolo di $\hat{f}$ è dominato dai vicini che sono lontani. Per evitare ciò, i *k* vicini sono ponderati in modo tale che i vicini più distanti abbiano un'influenza minore sul risultato. Durante la decisione a maggioranza nell'algoritmo K-NEARESTNEIGHBOR, i "voti" sono ponderati con il peso

$$w_i = \frac{1}{1 + \alpha d(\boldsymbol{x}, \boldsymbol{x_i})^2}$$

che diminuisce con il quadrato della distanza. La costante α determina la velocità di diminuzione dei pesi. La $\hat{f}$ viene quindi ridefinita così:

$$\hat{f}(\boldsymbol{x}) = \frac{\sum_{i=1}^{k} w_i f(\boldsymbol{x}_i)}{\sum_{i=1}^{k} w_i}$$

Con una distribuzione uniforme di punti nello spazio delle caratteristiche, questa definizione garantisce che l'influenza dei punti si avvicini asintoticamente allo zero all'aumentare della distanza. In questo modo diventa possibile utilizzare molti o anche tutti i dati di addestramento per classificare o approssimare un dato vettore di input.
Per avere un'idea di questi metodi, nella seguente figura il metodo k-vicino-prossimo (nella riga superiore) viene confrontato con la sua ottimizzazione ponderata sulla distanza (riga inferiore).



Confronto del metodo k-vicino-prossimo (riga superiore) con k = 1 (a sinistra), k = 2 (al centro) e k =6 (a destra), con la sua variante ponderata sulla distanza (riga inferiore) con α = 20 (a sinistra), α = 4 (al centro) e α = 1 (a destra) su un set di dati unidimensionale.

A causa della media, entrambi i metodi possono generalizzare, o in altre parole, annullare il rumore, se il numero di vicini per k-vicino-prossimo o il parametro α è impostato in modo appropriato. I diagrammi mostrano bene che il metodo ponderato in base alla distanza fornisce un'approssimazione molto più fluida rispetto a k-vicino-prossimo. Per quanto riguarda la qualità di approssimazione, questo metodo molto semplice può competere bene con sofisticati algoritmi di approssimazione come reti neurali non lineari, Support Vector Machine (SVM) e processi gaussiani.
Ci sono molte alternative alla funzione peso (chiamata anche kernel) fornita nella definizione

$$w_i = \frac{1}{1 + \alpha d(\boldsymbol{x}, \boldsymbol{x_i})^2}$$

Ad esempio può essere usata una funzione gaussiana o simile a forma di campana. Per la maggior parte delle applicazioni, i risultati non sono molto influenzati dalla selezione del kernel. Tuttavia, il parametro α che per tutte queste funzioni deve essere impostato manualmente ha una grande influenza sui risultati, come mostrato nella figura sopra. Per evitare un adattamento manuale così scomodo, sono stati sviluppati metodi di ottimizzazione per l'impostazione automatica di questo parametro.

PERCEPTRONLEARNING[M_+, M_-]
$\boldsymbol{w}$ = arbitrary vector of real numbers
Repeat
  For all $\boldsymbol{x} \in M_+$
    If $\boldsymbol{w}\,\boldsymbol{x} \leq 0$ **Then** $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}$
  For all $\boldsymbol{x} \in M_-$
    If $\boldsymbol{w}\,\boldsymbol{x} > 0$ **Then** $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x}$
Until all $\boldsymbol{x} \in M_+ \cup M_-$ are correctly classified

NEARESTNEIGHBOR[$M_+, M_-, \boldsymbol{s}$]
  $\boldsymbol{t} = \text{argmin}_{\boldsymbol{x} \in M_+ \cup M_-} \{d(\boldsymbol{s}, \boldsymbol{x})\}$
  If $\boldsymbol{t} \in M_+$ **Then Return** („+")
  Else Return(„–")

Confrontando il percettrone con il metodo k-vicino-prossimo, ci accorgiamo che in quest'ultimo non accade nulla nella fase di apprendimento: nessun parametro che caratterizza l'algoritmo viene modificato.
Algoritmi di questo tipo sono anche denotati come "lazy learning" (apprendimento pigro), in contrasto con "eager learning" (apprendimento impaziente), in cui la fase di apprendimento può essere costosa, ma l'applicazione a nuovi esempi è molto efficiente. Il percettrone e tutte le altre reti neurali, i decision tree e le reti Bayesiane sono metodi di apprendimento impaziente. Poiché i metodi di apprendimento pigro richiedono accesso alla memoria con tutti i dati di addestramento per approssimare un nuovo input, sono anche chiamati apprendimento basato sulla memoria. I metodi come k-vicino-prossimo sono adatti per tutte le situazioni in cui è necessaria una buona approssimazione locale, ma che non impongono requisiti di velocità del sistema.

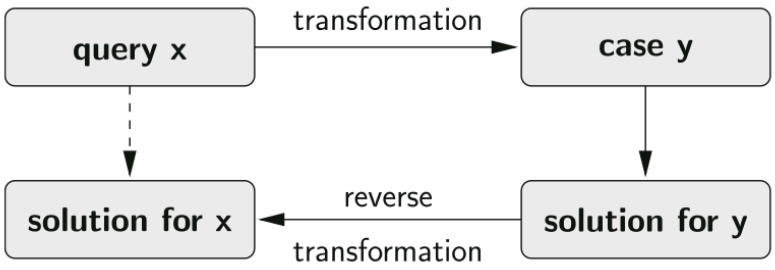
Per poter utilizzare i metodi descritti, i dati di addestramento devono essere disponibili sotto forma di vettori di numeri interi o numeri reali. Sono quindi inadatti per applicazioni in cui i dati sono rappresentati simbolicamente, ad esempio come formule logiche del primo ordine. Discuteremo ora brevemente di questo.

Case Based Reasoning (CBR)

Nel ragionamento basato sui casi (CBR), il metodo del vicino prossimo viene esteso alle descrizioni dei problemi simbolici e alle loro soluzioni. CBR viene utilizzato nella diagnosi di problemi tecnici nel servizio clienti o per hotline telefoniche. L'esempio mostrato nella seguente figura sulla diagnosi dello spegnimento di una luce di bicicletta illustra questo tipo di situazione. Si cerca una soluzione per l'interrogazione di un cliente con una luce posteriore per bicicletta difettosa.

Feature	**Query**	**Case from case base**
Defective part:	Rear light	Front light
Bicycle model:	Marin Pine Mountain	VSF T400
Year:	1993	2001
Power source:	Battery	Dynamo
Bulb condition:	ok	ok
Light cable condition:	?	ok
	Solution	
Diagnosis:	?	Front electrical contact missing
Repair:	?	Establish front electrical contact

Nella colonna di destra, viene fornito un caso simile alla query presente nella colonna centrale. Ciò deriva dal *case-base* (database di casi), che corrisponde ai dati di addestramento nel metodo del k-vicino-prossimo. Se prendessimo semplicemente il caso più simile, come facciamo nel metodo del k-vicino-prossimo, finiremmo per cercare di riparare la luce anteriore quando è la luce posteriore a essere rotta. Abbiamo quindi bisogno di una trasformazione della soluzione al problema trovato nel case-base simile alla query. I passaggi più importanti nella soluzione di un caso CBR sono mostrati nella seguente figura.



La trasformazione in questo esempio è semplice: la luce posteriore è mappata su quella anteriore. Per quanto bello e semplice possa sembrare in teoria questo metodo, in pratica la costruzione di sistemi diagnostici CBR è un compito molto difficile. Le tre principali difficoltà sono:
Modellazione: i domini dell'applicazione devono essere modellati in un contesto formale. Lo sviluppatore può prevedere e mappare tutti i possibili casi speciali e le varianti di problemi?
Somiglianza: trovare una metrica di somiglianza adatta per caratteristiche simboliche e non numeriche.
Trasformazione: anche se viene trovato un caso simile, non è ancora chiaro che forma hanno

la mappatura della trasformazione e la sua inversa inverso.
Esistono nella pratica sistemi CBR per applicazioni diagnostiche in uso oggi, tuttavia, per i motivi citati, questi rimangono molto indietro rispetto agli esperti umani in termini di prestazioni e flessibilità.
Quando la rappresentazione simbolica del problema può anche essere mappata su caratteristiche numeriche discrete o continue, i metodi di apprendimento induttivo menzionati come alberi decisionali o reti neurali possono essere utilizzati con successo.

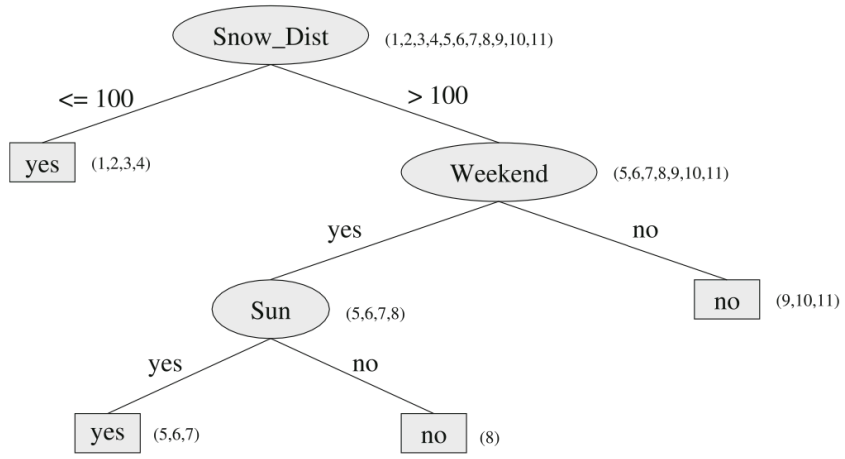
Decision Tree Learning

L'apprendimento con albero delle decisioni (Decision Tree Learning) è un algoritmo straordinariamente importante per l'IA perché è molto potente, ma anche semplice ed efficiente per estrarre conoscenza dai dati. Rispetto ai due algoritmi di apprendimento già introdotti, ha un vantaggio importante: la conoscenza estratta non è solo disponibile e utilizzabile come una black box, ma può essere anche facilmente compresa, interpretata e controllata dagli esseri umani sotto forma di un albero decisionale leggibile. Ciò rende anche l'apprendimento con albero decisionale uno strumento importante per il data mining. Ora mostriamo in un semplice esempio come un albero decisionale può essere costruito dai dati di addestramento, per poi analizzare l'algoritmo.

Uno sciatore appassionato vuole un albero decisionale che lo aiuti a decidere se vale la pena guidare la sua auto verso una stazione sciistica in montagna. Abbiamo quindi un problema a due classi: sì / no, basato sulle variabili elencate nella seguente tabella.

Variable	Value	Description
Ski (goal variable)	yes, no	Should I drive to the nearest ski resort with enough snow?
Sun (feature)	yes, no	Is there sunshine today?
Snow_Dist (feature)	≤ 100, >100	Distance to the nearest ski resort with good snow conditions (over/under 100 km)
Weekend (feature)	yes, no	Is it the weekend today?

Un albero decisionale è un albero i cui nodi interni rappresentano gli attributi del problema, i cui rami rappresentano i diversi valori di questi attributi, e i cui nodi foglia rappresentano i valori della classificazione. La seguente figura mostra un albero decisionale per il problema dello sciatore.



I dati utilizzati per la costruzione dell'albero decisionale sono mostrati nella seguente tabella.

Day	*Snow_Dist*	*Weekend*	*Sun*	*Skiing*
1	≤100	yes	yes	yes
2	≤100	yes	yes	yes
3	≤100	yes	no	yes
4	≤100	no	yes	yes
5	>100	yes	yes	yes
6	>100	yes	yes	yes
7	>100	yes	yes	no
8	>100	yes	no	no
9	>100	no	yes	no
10	>100	no	yes	no
11	>100	no	no	no

Ciascuna riga della tabella contiene i dati per un giorno e come tale rappresenta un campione. A un esame più attento, vediamo che la riga 6 e la riga 7 si contraddicono a vicenda. Pertanto, nessun algoritmo di classificazione deterministico può classificare correttamente tutti i dati. Il numero di dati classificati erroneamente deve quindi essere >=1. L'albero in figura classifica quindi i dati in modo ottimale.
Come viene creato un albero del genere dai dati? Per rispondere a questa domanda, inizialmente ci limiteremo ad attributi discreti con un numero finito di valori. Poiché anche il numero di attributi è finito e ogni attributo può essere presente al massimo una volta per percorso, esistono molti alberi decisionali diversi. Un semplice, ovvio algoritmo per la costruzione di un albero genererebbe semplicemente tutti gli alberi, quindi per ogni albero calcolare il numero di classificazioni errate dei dati, e alla fine scegliere l'albero con il numero minimo di errori. Quindi avremmo anche un algoritmo ottimale (in termini di errori per i dati di addestramento) per l'apprendimento dell'albero decisionale.
L'ovvio svantaggio di questo algoritmo è il suo tempo di calcolo inaccettabilmente alto, non appena il numero di attributi cresce. Svilupperemo ora un algoritmo euristico che, partendo dalla radice, costruisce ricorsivamente un albero decisionale. Per prima cosa viene scelto l'attributo con il maggior guadagno di informazione (information gain) (Snow_Dist) per il nodo radice dall'insieme di tutti gli attributi. Per ogni valore di attributo (<=100, >100) c'è un ramo nell'albero. Ora per ogni ramo questo processo viene ripetuto ricorsivamente. Durante la generazione dei nodi, viene sempre scelto l'attributo con il maggior information gain tra gli attributi non ancora utilizzati, nello spirito di una strategia *greedy*.

Entropia come misura di informazione

Introduciamo ora l'entropia come metrica per il contenuto informativo di un insieme di dati di allenamento *D*. Se guardiamo solo la variabile binaria *skiing* nell'esempio sopra, allora D può essere descritto come

$$D = (\text{yes, yes, yes, yes, yes, yes, no, no, no, no, no})$$

con probabilità stimate

$$p_1 = P(\text{yes}) = 6/11 \quad \text{and} \quad p_2 = P(\text{no}) = 5/11.$$

Qui evidentemente abbiamo una distribuzione di probabilità $\boldsymbol{p}$ = (6/11, 5/11). In generale, per un problema con *n* classi si ha

$$\boldsymbol{p} = (p_1, \ldots, p_n)$$

con

$$\sum_{i=1}^{n} p_i = 1.$$

Per introdurre il contenuto informativo di una distribuzione osserviamo due casi estremi. Sia

$$\boldsymbol{p} = (1, 0, 0, \ldots, 0)$$

una distribuzione di probabilità in cui il primo degli *n* eventi accadrà sicuramente e tutti gli altri no. L'incertezza sull'esito degli eventi è quindi minima. Al contrario, per la distribuzione uniforme

$$\boldsymbol{p} = \left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)$$

l'incertezza è massima perché nessun evento può essere distinto dagli altri. Qui Claude Shannon si è chiesto quanti bit sarebbero stati necessari per codificare un evento del genere. Nel primo caso sono necessari zero bit perché sappiamo che il caso $i = 1$ si verifica sempre. Nel secondo caso, uniformemente distribuito, ci sono *n* possibilità ugualmente probabili. Per la codifica binaria, qui sono necessari $\log_2 n$ bit. Poiché tutte le probabilità individuali sono $p_i = 1/n$, per questa codifica sono necessari $\log_2(1/p_i)$ bit.
Nel caso generale $\boldsymbol{p} = (p_1, ..., p_n)$, se le probabilità degli eventi elementari deviano dalla distribuzione uniforme, allora bisogna calcolare un valore atteso H per il numero di bit necessari. A tal fine peseremo tutti i valori dei bit necessari per ciascun evento $\log_2 (1/p_i) = -\log_2 p_i$ con la probabilità dell'evento stesso e otteniamo:

$$H = \sum_{i=1}^{n} p_i(-\log_2 p_i) = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

Maggiore è l'incertezza sul risultato, maggiore è il numero di bit di cui abbiamo bisogno per codificare un evento. Pertanto definiamo:

l’entropia H come metrica per l’incertezza di una distribuzione di probabilità

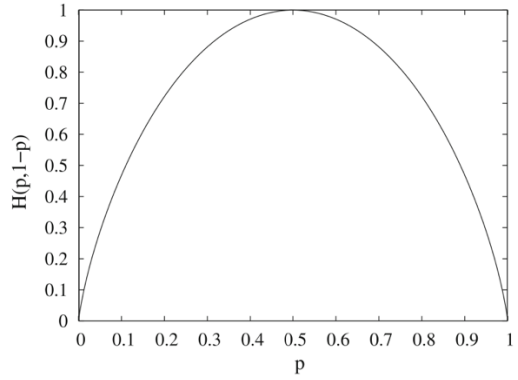
$$H(\boldsymbol{p}) = H(p_1, \ldots, p_n) := -\sum_{i=1}^{n} p_i \log_2 p_i.$$

Se sostituiamo l’evento certo $\boldsymbol{p}$ = (1, 0, 0, ..., 0) in questa definizione, avremmo $0\log_2 0$, che risulta un’espressione indefinita. Risolviamo questo problema con la definizione $0\log_2 0 := 0$ (a cui possiamo arrivare estrapolando il limite della funzione $x\log_2 x$ per $x \to 0$).

Ora possiamo quindi calcolare H(1, 0, ..., 0) = 0. Si dimostra anche che l’entropia nell’ipercubo $[0, 1]^n$ con il vincolo che $\Sigma_i p_i = 1$assume il suo valore massimo con la distribuzione uniforme (1/n, …, 1/n). Nel caso di un evento con due possibili esiti, che corrispondono a due classi, il risultato è

$$H(\boldsymbol{p}) = H(p_1, p_2) = H(p_1, 1 - p_1) = -(p_1 \log_2 p_1 + (1 - p_1) \log_2 (1 - p_1)).$$

Questa espressione è mostrata come una funzione di p_1 nella seguente figura, con il suo massimo a $p_1 = 1/2$.



Poiché a ciascun set di dati D classificato viene assegnata una distribuzione di probabilità $\boldsymbol{p}$ stimando le probabilità di ciascuna classe, possiamo estendere il concetto di entropia ai dati mediante la definizione:

$$H(D) = H(\boldsymbol{p}).$$

Ora, poiché il contenuto informativo I(D) dell’insieme di dati D è inteso essere l’opposto dell’incertezza. Quindi definiamo il contenuto informativo di un set di dati D come

$$I(D) := 1 - H(D).$$

Se applichiamo la formula dell’entropia all’esempio dello sciatore, il risultato è

$$H(6/11, 5/11) = 0.994$$

Durante la costruzione di un albero decisionale, il set di dati viene ulteriormente suddiviso per ogni nuovo attributo. Più un attributo aumenta il contenuto informativo della distribuzione dividendo i dati, migliore è l'attributo. Definiamo di conseguenza quanto segue. Il guadagno di informazione (information gain) G(D, A) attraverso l'uso dell'attributo A è determinato dalla differenza tra il contenuto informativo medio del dataset
$D = D_1 \cup D_2 \cup \cdots \cup D_n$ diviso per l'attributo a *n* valori A
e il contenuto informativo I(D) dell'insieme di dati non diviso, ossia

$$G(D,A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i) - I(D).$$

da cui otteniamo

$$\begin{aligned} G(D,A) &= \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i) - I(D) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} (1 - H(D_i)) - (1 - H(D)) \\ &= 1 - \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) - 1 + H(D) \\ &= H(D) - \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i). \end{aligned}$$

Applicato al nostro esempio per l'attributo Snow_Dist, questo produce

$$\begin{aligned} G(D, Snow_Dist) &= H(D) - \left(\frac{4}{11} H(D_{\leq 100}) + \frac{7}{11} H(D)_{>100} \right) \\ &= 0.994 - \left(\frac{4}{11} \cdot 0 + \frac{7}{11} \cdot 0.863 \right) = 0.445. \end{aligned}$$

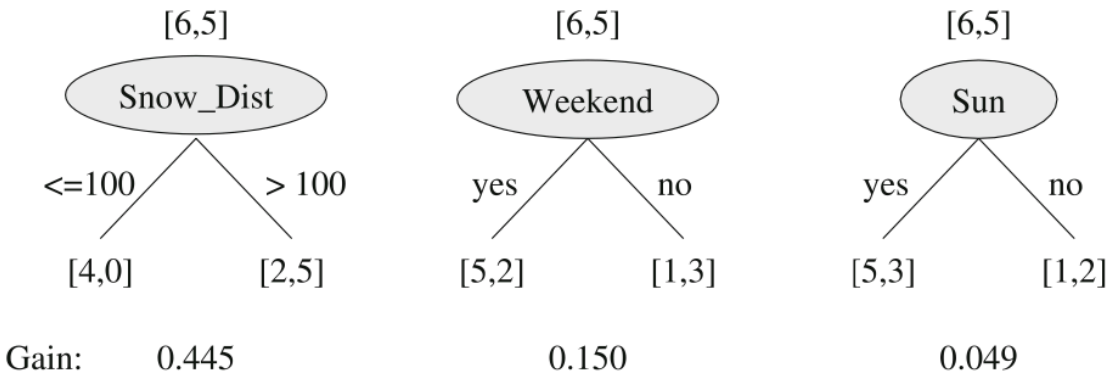
Analogamente otteniamo

$$G(D, Weekend) = 0.150$$

e

$$G(D, Sun) = 0.049.$$

L'attributo Snow_Dist quindi diventa il nodo radice dell'albero decisionale. La situazione della selezione di questo attributo è ancora una volta chiarita nella seguente figura:



Il guadagno calcolato per i vari attributi riflette se la divisione dei dati per il rispettivo attributo risulta in una migliore divisione di classe. Più le distribuzioni generate dall'attributo si discostano dalla distribuzione uniforme, maggiore è il guadagno di informazione (information gain).

I due valori di attributo <=100 e >100 generano due archi nell'albero, che corrispondono ai sottoinsiemi $D_{<=100}$ e $D_{>100}$. Per il sottoinsieme $D_{<=100}$ la classificazione è chiaramente "sì" quindi l'albero termina qui. Nell'altro ramo $D_{>100}$ non c'è un risultato chiaro. Quindi l'algoritmo si ripete ricorsivamente. Tra i due attributi ancora disponibili, Sun e Weekend, si deve scegliere quello migliore. Calcoliamo quindi

$$G(D_{>100}, Weekend) = 0.292$$

e

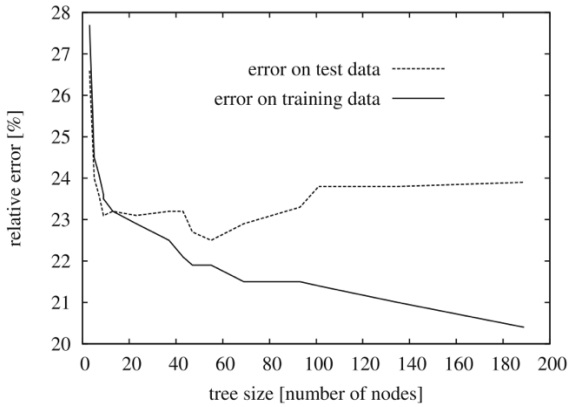
$$G(D_{>100}, Sun) = 0.170.$$

Al nodo si assegna perciò l'attributo Weekend. Per Weekend = no l'albero termina con la decisione Ski = no. Un calcolo del guadagno qui restituisce il valore 0. Per Weekend = yes, Sun produce un guadagno di 0,171. Quindi la costruzione dell'albero termina perché non sono disponibili ulteriori attributi, sebbene l'esempio numero 7 sia classificato erroneamente.

Potatura dell'albero

Sin dai tempi di Aristotele è stato stabilito che tra due teorie scientifiche che spiegano ugualmente bene la stessa situazione, si preferisce quella più semplice. Questa legge dell'economia, ora nota anche come rasoio di Occam, è di grande importanza per l'apprendimento automatico e il data mining.
Un albero decisionale è una teoria per descrivere i dati di addestramento. Una teoria diversa per descrivere i dati sono i dati stessi. Se l'albero classifica tutti i dati senza errori, ma è molto più compatto e quindi più facilmente comprensibile dall'uomo, allora è preferibile, secondo il rasoio di Occam. Lo stesso vale per due alberi decisionali di dimensioni diverse. Pertanto, l'obiettivo di ogni algoritmo per la generazione di un albero decisionale deve essere quello di generare l'albero decisionale più piccolo possibile per un dato tasso di errore. Tra tutti gli alberi con un tasso di errore fisso, dovrebbe essere sempre selezionato l'albero più piccolo. Finora non abbiamo definito con precisione il termine tasso di errore. Come già accennato più volte, è importante che l'albero appreso non solo memorizzi i dati di allenamento, ma che generalizzi bene. Per testare la capacità di generalizzare di un albero, dividiamo i dati disponibili in un insieme di dati di addestramento e un insieme di dati di testing. I dati di testing sono nascosti dall'algoritmo di apprendimento e utilizzati solo per il

test. Se è disponibile un set di dati di grandi dimensioni, possiamo ad esempio utilizzare due terzi dei dati per l'apprendimento e il terzo rimanente per il test.
Oltre a una migliore comprensibilità, il rasoio di Occam ha un'altra importante giustificazione: la capacità di generalizzazione. Più complesso è il modello (in questo contesto un albero decisionale), più dettagli sono rappresentati, ma nella stessa misura meno il modello è trasferibile a nuovi dati. Questa relazione è illustrata nella figura seguente.



Alberi decisionali di varie dimensioni sono stati addestrati con dati medicali sulla diagnosi di appendicite in base ai sintomi. Nel grafico sono riportati gli errori di classificazione sia sui dati di addestramento che sui dati di test. Il tasso di errore sui dati di addestramento diminuisce in modo monotono con la dimensione dell'albero. Fino a una dimensione dell'albero di 55 nodi, diminuisce anche il tasso di errore sui dati di test. Se l'albero cresce ulteriormente, tuttavia, il tasso di errore ricomincia ad aumentare. Questo effetto, che abbiamo già visto nel metodo del vicino prossimo, è chiamato overfitting.

Definizione (tratta da [1]):
Sia dato uno specifico algoritmo di apprendimento, ossia un agente d'apprendimento. Diciamo che un agente A ha fatto overfit sui dati di addestramento se c'è un altro agente A′ il cui errore sui dati di addestramento è maggiore di quello di A, ma il cui errore sull'intera distribuzione dei dati è minore dell'errore di A.

Come possiamo ora trovare questo punto di minimo errore sui dati del test? L'algoritmo più ovvio è chiamato convalida incrociata (cross-validation). Durante la costruzione dell'albero, l'errore sui dati di test viene misurato in parallelo. Non appena l'errore aumenta in modo significativo, viene salvato l'albero con l'errore minimo.

Spesso i singoli valori degli attributi non sono presenti nei dati di addestramento. Nel set di dati LEXMED, ad esempio, si può avere la seguente entry:
56; 1; 1; 1; 1; 1; 0; 1; 1; 1; 0; 390; ?; 14100; 0; 1;
in cui manca uno dei valori di febbre. Tali dati possono tuttavia essere utilizzati durante la costruzione dell'albero decisionale. Possiamo assegnare all'attributo il valore più frequente dell'intero set di dati o il più frequente di tutti i punti dati della stessa classe. È anche meglio sostituire il valore mancante con la distribuzione di probabilità di tutti i possibili valori e dividere l'esempio di addestramento su diversi rami in base a questa distribuzione. I valori mancanti possono verificarsi non solo durante l'apprendimento, ma anche durante la classificazione. Questi vengono gestiti allo stesso modo dell'apprendimento.

Cross-validation

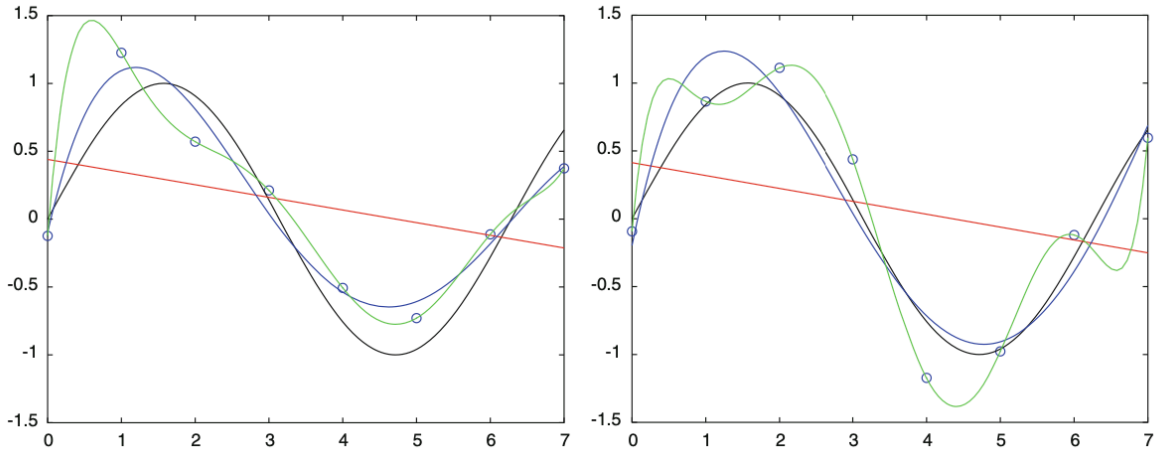
Molti algoritmi di apprendimento hanno il problema dell'eccessivo adattamento (overfitting). Potenti algoritmi di apprendimento, come ad esempio l'apprendimento con albero decisionale, possono adattare la complessità del modello appreso alla complessità dei dati di addestramento. Questo porta a un overfitting se c'è rumore nei dati.
Con la cross-validation si tenta di ottimizzare la complessità del modello in modo tale da ridurre al minimo l'errore di classificazione o di approssimazione su un set di dati di test finora mai visti. Ciò richiede che la complessità del modello sia controllabile da un parametro γ. Per gli alberi decisionali, questa è, ad esempio, la dimensione dell'albero. Per il metodo k-vicino-prossimo il parametro è *k*, il numero dei vicini prossimi, mentre per le reti neurali è il numero dei neuroni nei livelli nascosti (come vedremo più avanti).
Variamo il parametro γ durante l'addestramento dell'algoritmo su un set di dati di addestramento e scegliamo il valore di γ che riduce al minimo l'errore su un set di dati di test indipendente. La cross-validation k-aria funziona secondo il seguente schema:

CROSSVALIDATION($\boldsymbol{X}, k$)
Partition data into k equally sized blocks $\boldsymbol{X} = \boldsymbol{X}_1 \cup \ldots \cup \boldsymbol{X}_k$
For all $\gamma \in \{\gamma_{min}, \ldots, \gamma_{max}\}$
  For all $i \in \{1, \ldots, k\}$
    Train a model of complexity γ on $\boldsymbol{X} \backslash \boldsymbol{X}_i$
    Compute the error $E(\gamma, \boldsymbol{X}_i)$ on the test set $\boldsymbol{X}_i$
  Compute the mean error $E(\gamma) = \frac{1}{k} \sum_{i=1}^{k} E(\gamma, \boldsymbol{X}_i)$
Choose the value $\gamma_{opt} = \text{argmin}_\gamma E(\gamma)$ with smallest mean error
Train the final model with complexity γ_{opt} on the whole data set $\boldsymbol{X}$

L'intero set di dati X è diviso in k blocchi di uguali dimensioni. Quindi l'algoritmo viene addestrato k volte su k - 1 blocchi e testato sul blocco rimanente. I k errori calcolati vengono mediati e il valore γ_{opt} con l'errore medio più piccolo viene scelto per addestrare il modello finale sull'intero set di dati X.
Se il training set X è grande, può essere diviso, ad esempio, in k = 3 o k = 10 blocchi. Il conseguente aumento della complessità computazionale è generalmente accettabile. Per insiemi di addestramento di piccole dimensioni, è preferibile eseguire l'addestramento su tutti gli n vettori di feature, se possibile.
La cross-validation è il metodo di ottimizzazione automatica più importante e più utilizzato per stabilire la complessità del modello. Risolve il problema dell'overfitting. Possiamo ottenere ulteriori informazioni su questo problema da una breve occhiata al cosiddetto bias variance tradeoff. Se viene utilizzato un modello eccessivamente semplice, ciò forza l'approssimazione non ottimale dei dati in una certa direzione (bias). D'altra parte, se viene utilizzato un modello eccessivamente complesso, tenderà a fare overfitting su qualsiasi set di dati. Pertanto, per un nuovo campione di dati dalla stessa distribuzione, potrebbe apprendere un modello molto diverso. Il modello appreso, quindi, può variare notevolmente a causa di un cambiamento nei dati (varianza alta).
Questa relazione può essere illustrata con l'esempio di approssimazione di funzioni utilizzando i polinomi. Se si sceglie il grado del polinomio come parametro di complessità γ, il grado 1 approssima una linea, che non è una buona approssimazione per dati non banali. Se invece il grado polinomiale è uguale a n - 1 per n punti, tende a centrare ogni punto e a ridurre a zero l'errore sui dati di addestramento. Questo può essere visto nella seguente figura a sinistra, in cui sono stati generati otto punti dati utilizzando

$$y(x) = \sin(x) + g(0, 0.2)$$

dove $g(0, 0.2)$ genera rumore normalmente distribuito con media zero e una deviazione standard di 0.2.



Nell'immagine di sinistra, oltre ai punti dati, è inscritta la funzione seno sottostante (in nero), nonché una linea retta (rossa) approssimata con il metodo dei minimi quadrati. Inoltre, un polinomio di settimo grado (in verde) viene interpolato attraverso i punti, che centra ogni punto. È presente anche un polinomio di quarto grado (in blu), che si avvicina di più alla curva sinusoidale. Nell'immagine a destra, le stesse approssimazioni sono state eseguite di nuovo su otto nuovi punti dati. Si può vedere molto bene che nonostante dati diversi, entrambe le linee rette si discostano notevolmente dai dati (bias grande), ma hanno un grafico di funzione simile (varianza molto piccola). Entrambi i polinomi di settimo grado, d'altra parte, si adattano perfettamente ai dati (bias zero), ma hanno grafici di funzione molto diversi (varianza molto grande). Questo è un evidente effetto overfitting. Il polinomio di quarto grado presenta un buon compromesso. La cross-validation produrrebbe presumibilmente il grado quattro come soluzione ottimale.

One-class Learning

Le attività di classificazione nell'apprendimento *supervisionato* richiedono che a tutti i dati di formazione siano assegnate etichette di classe. Tuttavia, ci sono applicazioni per le quali è disponibile solo una classe di etichette. Un classico esempio è il rilevamento di errori in sistemi tecnici complessi. Il compito è riconoscere se un dispositivo è difettoso o meno. Sembra un normale problema a due classi. Nei dati di allenamento, lo stato del dispositivo (buono o difettoso) deve essere fornito manualmente. In pratica, l'addestramento del classificatore avviene durante la distribuzione del dispositivo o successivamente su richiesta mentre il dispositivo viene utilizzato. L'acquisizione dei dati in condizioni prive di errori non è un problema. Tuttavia, la raccolta dei dati dal sistema difettoso è problematica per i seguenti motivi:

- La misurazione sull'attrezzatura di produzione che è stata intenzionalmente resa difettosa è associata a costi elevati perché la misurazione porta a costosi tempi di fermo.
- La misurazione su apparecchiature difettose con errori che si verificano effettivamente nella realtà è spesso impossibile perché, durante l'installazione dell'apparecchiatura, i potenziali errori che si verificano in seguito potrebbero essere sconosciuti.

- Nessun ingegnere sa in anticipo esattamente quale tipo di errore si verificherà in una nuova apparecchiatura. Se ora, durante l'allenamento, vengono effettuate misurazioni con alcuni tipi di errori, ciò potrebbe portare a risultati di classificazione scadenti. Se gli errori utilizzati per l'addestramento non sono rappresentativi per il dispositivo, ovvero se alcuni tipi di errori mancano nei dati di addestramento, l'ipersuperficie di separazione delle classi appresa nello spazio delle caratteristiche spesso porterà a classificazioni di falsi positivi.

In questi casi, non è possibile addestrare un classificatore a due classi nel modo standard. Invece, è possibile utilizzare l'apprendimento di una classe, che usa i dati di una classe durante l'addestramento.
Supponiamo che non siano disponibili dati negativi. Tuttavia, sono disponibili dati positivi, ovvero dati da un funzionamento senza errori, in quantità sufficiente. Pertanto è necessario un algoritmo di apprendimento che può, sulla base dei dati privi di errori, acquisire tutti gli stati operativi privi di errori del dispositivo e classificare tutti gli altri come negativi.
A tale scopo, esiste tra gli altri un algoritmo chiamato Nearest Neighbor Data Description (NNDD, descrizione dei dati del vicino prossimo).

Nearest Neighbor Data Description

Simile al metodo del vicino prossimo visto in precedenza, l'algoritmo NNDD appartiene alla categoria degli algoritmi di apprendimento pigro. Durante l'apprendimento, le uniche cose che accadono sono la normalizzazione e la memorizzazione dei vettori delle caratteristiche. La normalizzazione di ogni singola caratteristica è necessaria per dare a ciascuna caratteristica lo stesso peso. Senza normalizzazione, una caratteristica nell'intervallo [0; 10^{-4}] verrebbe persa nel rumore di un'altra caratteristica nell'intervallo [-10000; +10000]. Quindi tutte le caratteristiche sono scalate linearmente sull'intervallo [0, 1]. L'algoritmo del vicino prossimo entra in gioco per primo durante la classificazione.

Sia $\mathbf{X} = (\mathbf{x}_1; \ldots; \mathbf{x}_n)$ un insieme di addestramento costituito da n vettori di caratteristiche (features). Un nuovo punto **q**, che deve ancora essere classificato, è accettato se

$$D(\boldsymbol{q}, \mathrm{NN}(\boldsymbol{q})) \leq \gamma \bar{D},$$

ossia, se la distanza dal vicino prossimo non è maggiore di $\gamma\underline{D}$. Qui $D(x, y)$ è una metrica della distanza, ad esempio, qui è usata la distanza euclidea. La funzione

$$\mathrm{NN}(\boldsymbol{q}) = \underset{\boldsymbol{x} \in \boldsymbol{X}}{\operatorname{argmin}} \{D(\boldsymbol{q}, \boldsymbol{x})\}$$

restituisce il vicino prossimo a **q**, e

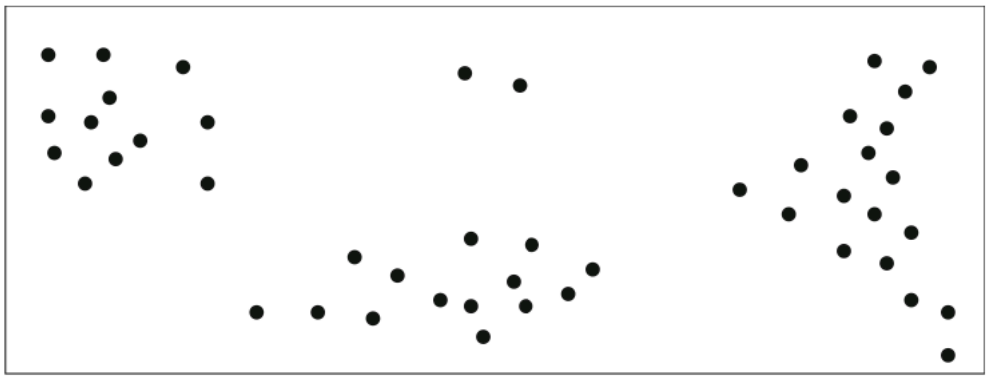
$$\bar{D} = \frac{1}{n} \sum_{i=1}^{n} D(\boldsymbol{x}_i, \mathrm{NN}(\boldsymbol{x}_i))$$

è la distanza media dei vicini più vicini a tutti i punti dati in **X**. Per calcolare $\underline{D}$ viene calcolata la distanza di ciascun punto dal suo vicino prossimo. $\underline{D}$ è quindi la media aritmetica di queste distanze calcolate. Si potrebbe usare $\gamma = 1$ nella diseguaglianza di cui sopra, ma si

otterrebbero risultati di classificazione non ottimali. È meglio determinare il parametro γ usando la cross-validation come illustrato nella lezione precedente in modo tale che il tasso di errore del classificatore NNDD sia il più piccolo possibile.

Clustering

Se cerchiamo in un motore di ricerca il termine “mars”, otterremo risultati come “il pianeta marte (mars in inglese)” e “conglomerato aziendale produttore di cioccolato, dolciumi e bevande”, che sono semanticamente abbastanza diversi. Nella serie di documenti scoperti ci sono due gruppi notevolmente diversi. Google, ad esempio, elenca ancora i risultati in modo non strutturato. Sarebbe meglio se il motore di ricerca separasse i *cluster* e li presentasse separati all’utente, perché l’utente è solitamente interessato solo a uno dei cluster.
La distinzione del clustering (raggruppamento) dall’apprendimento supervisionato è che i dati di addestramento non sono etichettati. Manca quindi la pre-strutturazione dei dati da parte del supervisore. Piuttosto, trovare le strutture è il punto centrale del clustering. Nello spazio dei dati di addestramento, si trovano accumuli di dati come quelli in figura.



In un cluster, la distanza dei punti adiacenti è in genere inferiore alla distanza tra i punti di cluster diversi. Pertanto la scelta di una metrica di distanza adeguata per i punti, cioè per gli oggetti da raggruppare e per i cluster, è di fondamentale importanza. Come prima, assumiamo di seguito che ogni oggetto dati sia descritto da un vettore di attributi numerici.

Metriche della distanza
Per ciascuna applicazione, le varie metriche di distanza sono definite per la distanza *d* tra due vettori *x* e *y* in R^n. La più comune è la distanza euclidea:

$$d_e(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}.$$

Un po’ più semplice è la somma delle distanze al quadrato:

$$d_q(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{n}(x_i - y_i)^2,$$

che, per algoritmi in cui si confrontano solo le distanze, è equivalente alla distanza euclidea.
Viene utilizzata anche la distanza di Manhattan:

$$d_m(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

così come la distanza del componente massimo:

$$d_\infty(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1,\ldots,n} |x_i - y_i|$$

Nel contesto della classificazione dei testi, si usa spesso il prodotto scalare normalizzato

$$\frac{\boldsymbol{x}\,\boldsymbol{y}}{|\boldsymbol{x}|\,|\boldsymbol{y}|}$$

dove | **x** | è la norma euclidea (modulo) del vettore **x**. Poiché questa formula è una metrica per la somiglianza dei due vettori, come metrica della distanza si usa spesso l'inverso

$$d_s(\boldsymbol{x},\boldsymbol{y}) = \frac{|\boldsymbol{x}|\,|\boldsymbol{y}|}{\boldsymbol{x}\,\boldsymbol{y}}$$

Per un dizionario con 50.000 parole, il valore x_i è uguale alla frequenza della i-esima parola del dizionario nel testo. Poiché normalmente quasi tutti i componenti sono zero in un tale vettore, durante il calcolo del prodotto scalare, quasi tutti i termini della somma sono zero. Sfruttando questo tipo di informazioni, l'implementazione può essere velocizzata in modo significativo.

k-Means e l'algoritmo EM

Ogni volta che il numero di cluster è già noto in anticipo, è possibile utilizzare l'algoritmo k-Means. Come suggerisce il nome, k cluster sono definiti dal loro valore medio. Per prima cosa i k punti medi dei cluster μ_1, ..., μ_k vengono inizializzati alle coordinate di k punti dati, selezionati casualmente o manualmente. [Nota: la selezione di punti arbitrari che non sono punti dati come centri di cluster può portare a cluster vuoti.] Quindi i due passaggi seguenti vengono eseguiti ripetutamente:

- classificazione di tutti i dati rispetto al punto medio del cluster più vicino:
- ricalcolo del punto medio del cluster.

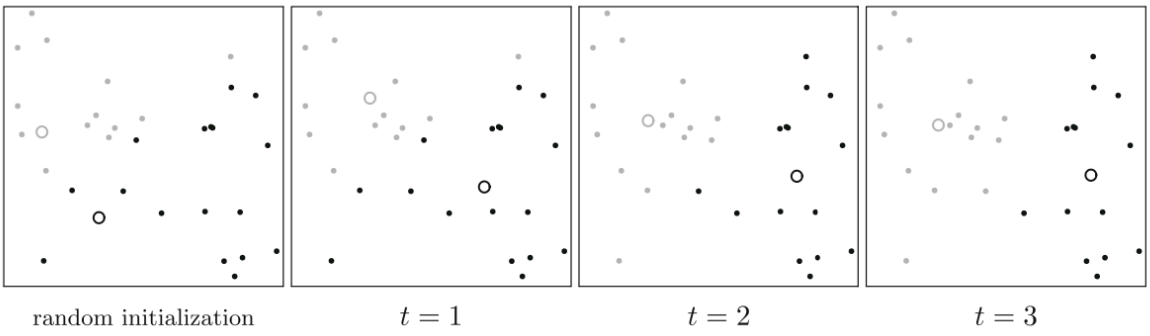
Ne risulta il seguente algoritmo:

K-MEANS($\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, k$)
initialize cluster centers $\boldsymbol{\mu}_1 = \boldsymbol{x}_{i_1}, \ldots, \boldsymbol{\mu}_k = \boldsymbol{x}_{i_k}$ (e.g. randomly)
Repeat
    classify $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to each's nearest $\boldsymbol{\mu}_i$
    recalculate $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
Until no change in $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
Return($\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$)

Il calcolo del punto medio del cluster μ per i punti x_1, ..., x_l viene eseguito così:

$$\boldsymbol{\mu} = \frac{1}{l} \sum_{i=1}^{l} x_i.$$

L'esecuzione su un esempio è mostrata in figura per il caso di due classi.



k-means con due classi (k = 2) applicate a 30 punti dati. All'estrema sinistra il set di dati con i centri iniziali e a destra i cluster dopo ogni iterazione. Dopo tre iterazioni si raggiunge la convergenza.

Vediamo come, dopo tre iterazioni, i centri di classe, che sono stati inizialmente scelti a caso, si stabilizzano. Sebbene questo algoritmo non garantisca la convergenza, di solito converge molto rapidamente. Ciò significa che il numero di passaggi di iterazione è in genere molto inferiore al numero di punti dati. La sua complessità è *O(ndkt)*, dove *n* è il numero totale di punti, *d* la dimensionalità dello spazio delle caratteristiche e *t* il numero di passaggi di iterazione.
In molti casi, la necessità di dare il numero di classi in anticipo pone una limitazione scomoda. Quindi introdurremo in seguito un algoritmo più flessibile.
Prima di ciò, tuttavia, menzioneremo l'algoritmo EM, che è una variante continua di k-means, poiché non effettua un'assegnazione definitiva dei dati alle classi, piuttosto, per ogni punto restituisce la probabilità che appartenga alle varie classi. Qui dobbiamo presumere che il tipo di distribuzione di probabilità sia noto. Spesso viene utilizzata la distribuzione normale. Il compito dell'algoritmo EM è determinare i parametri (media μ_i e matrice di covarianza Σ_i delle k distribuzioni normali multidimensionali) per ciascun cluster.
Analogamente a k-means, i due passaggi seguenti vengono eseguiti ripetutamente:
aspettativa (Expectation): per ogni punto dati viene calcolata la probabilità $P(C_j|x_i)$ che appartenga a ciascun cluster;

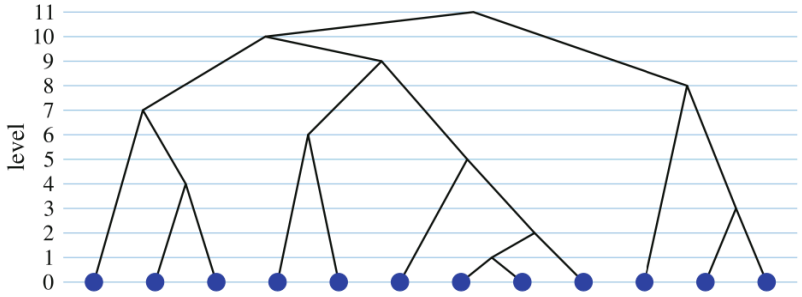
massimizzazione (Maximization): utilizzando le probabilità appena calcolate, i parametri della distribuzione vengono ricalcolati.
In tal modo si ottiene un raggruppamento più "soft", che in molti casi porta a risultati migliori. Questa alternanza tra aspettativa e massimizzazione dà il nome all'algoritmo EM.

Clustering gerarchico

Nel clustering gerarchico iniziamo con *n* cluster costituiti da un punto ciascuno. Quindi i cluster più vicini tra loro vengono combinati fino a quando tutti i punti sono stati combinati in un unico cluster, o fino a quando non viene raggiunto un criterio di terminazione. Otteniamo il seguente algoritmo:

HIERARCHICALCLUSTERING$(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, k)$
initialize $C_1 = \{\boldsymbol{x}_1\}, \ldots, C_n = \{\boldsymbol{x}_n\}$
Repeat
    Find two clusters C_i and C_j with the smallest distance
    Combine C_i and C_j
Until Termination condition reached
Return(tree with clusters)

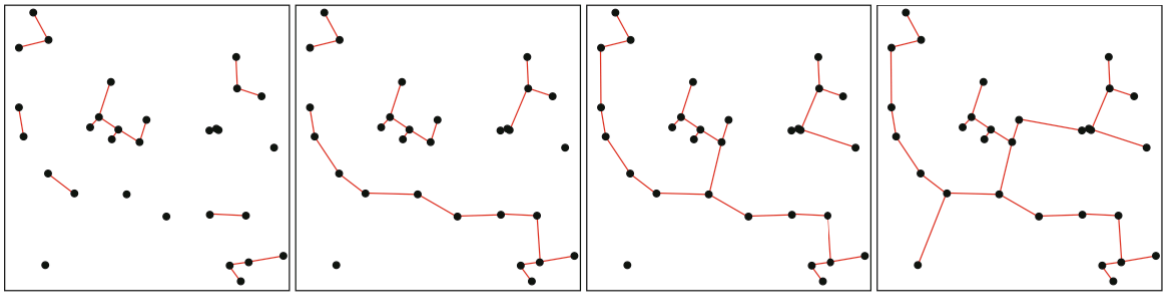
La condizione di terminazione potrebbe essere, ad esempio, un numero desiderato di cluster viene raggiunto, oppure una distanza massima tra i cluster viene raggiunta. Nella seguente figura questo algoritmo è rappresentato schematicamente come un albero binario, in cui dal basso verso l'alto ad ogni passo, cioè ad ogni livello, vengono collegati due sottoalberi. Al livello più in alto tutti i punti sono unificati in un unico grande cluster.



Finora non è abbiamo specificato come vengano calcolate le distanze tra i cluster. In effetti, nella sezione precedente abbiamo definito varie metriche di distanza per i punti, ma queste non possono essere utilizzate sui cluster. Una metrica comoda e spesso utilizzata è la distanza tra i due punti più vicini nei due cluster C_i e C_j:

$$d_{min}(C_i, C_j) = \min_{\boldsymbol{x} \in \boldsymbol{C_i},\, \boldsymbol{y} \in \boldsymbol{C_j}} d(\boldsymbol{x}, \boldsymbol{y}).$$

Otteniamo così l'algoritmo del vicino prossimo (da non confondere con quello visto per la classificazione dei punti), la cui applicazione è mostrata nella seguente figura.



L'algoritmo del vicino più vicino applicato ai dati che avevamo prima diviso in 2 cluster con k-means, a diversi livelli con 12, 6, 3, 1 cluster.

Vediamo che questo algoritmo genera uno spanning tree minimo, ossia un grafo aciclico e non orientato con la somma minima delle lunghezze dei lati. L'esempio mostra inoltre che i due algoritmi descritti generano cluster abbastanza differenti. Questo ci dice che per i grafici con cluster non chiaramente separati, il risultato dipende fortemente dall'algoritmo o dalla metrica di distanza scelta.
Per un'implementazione efficiente di questo algoritmo, creiamo prima una matrice di adiacenza in cui vengono salvate le distanze tra tutti i punti, che richiede tempo e memoria $O(n^2)$. Se il numero di cluster non ha un limite superiore, il ciclo itererà n - 1 volte e il tempo di calcolo asintotico diventa $O(n^3)$.
Per calcolare la distanza tra due cluster, possiamo anche utilizzare la distanza tra i due punti più lontani:

$$d_{max}(C_i, C_j) = \max_{\boldsymbol{x} \in C_i,\, \boldsymbol{y} \in C_j} d(\boldsymbol{x}, \boldsymbol{y})$$

e otteniamo l'algoritmo del vicino più lontano. In alternativa, viene utilizzata la distanza del punto medio del cluster $d_{\mu}(C_i, C_j) = d(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j)$.

Come determinare il numero di cluster?

In tutti gli algoritmi di clustering discussi finora, l'utente deve specificare il numero di cluster. In molti casi, gli utenti non hanno idea di quale sarebbe un numero ragionevole di cluster, piuttosto desiderano semplicemente un partizionamento "buono", ovvero la separazione dei loro *n* punti dati in cluster. Poiché sfortunatamente non esiste uno standard assoluto e generalmente accettato per la qualità di una partizione, presentiamo un metodo euristico per valutare un raggruppamento utilizzando il criterio della larghezza della sagoma (silhouette width, SW).
Prima di discutere questo criterio, mostriamo innanzitutto come può essere applicato. Supponiamo di avere un programma in grado di enumerare in modo combinatorio tutte le partizioni possibili per un insieme di *n* punti dati. Quindi potremmo semplicemente applicare il criterio SW a ciascuna partizione e cercare il massimo per determinare la partizione migliore. Ma poiché il numero di partizioni cresce rapidamente con il numero di punti dati, questo metodo non è pratico. Potremmo, tuttavia, semplicemente applicare uno degli algoritmi di clustering presentati (ad esempio k-means), lasciarlo funzionare per k da 1 a n, quindi utilizzare il criterio SW per determinare il k migliore e la sua rispettiva partizione. Quello che ci manca ancora è un tale criterio che misuri la qualità di una partizione. L'idea, molto approssimativamente, è la seguente: la distanza media tra due punti arbitrari all'interno dello stesso cluster dovrebbe essere inferiore alla distanza tra due punti arbitrari che si

trovano in diversi cluster vicini. Il rapporto tra la distanza media tra i punti nei cluster vicini e la distanza media tra i punti all'interno del cluster dovrebbe essere massimizzato.
Siano dati i punti ($\mathbf{x}_1, \ldots, \mathbf{x}_n$), così come una funzione di raggruppamento *c*

$$c : \boldsymbol{x}_i \mapsto c(\boldsymbol{x}_i),$$

che assegna ogni punto a un cluster.
Sia $\underline{d}(i,l)$ la distanza media da $\mathbf{x}_i$ a tutti i punti che non sono $\mathbf{x}_i$ nel cluster *l*.
Allora $a(i) = \underline{d}(i,c(\mathbf{x}_i))$ è la distanza media da $\mathbf{x}_i$ a tutti gli altri punti nel proprio cluster. Se $\mathbf{x}_i$ è l'unico punto nel cluster, poniamo $a(i) = 0$.

$$b(i) = \min_{j \neq c(\boldsymbol{x}_i)} \{\bar{d}(i,j)\}$$

è la distanza media più piccola dal punto $\mathbf{x}_i$ a un cluster a cui $\mathbf{x}_i$ non appartiene. Definiamo allora la seguente funzione:

$$s(i) = \begin{cases} 0 & \text{if } a(i) = 0 \\ \dfrac{b(i) - a(i)}{\max\{a(i), b(i)\}} & \text{otherwise} \end{cases}$$

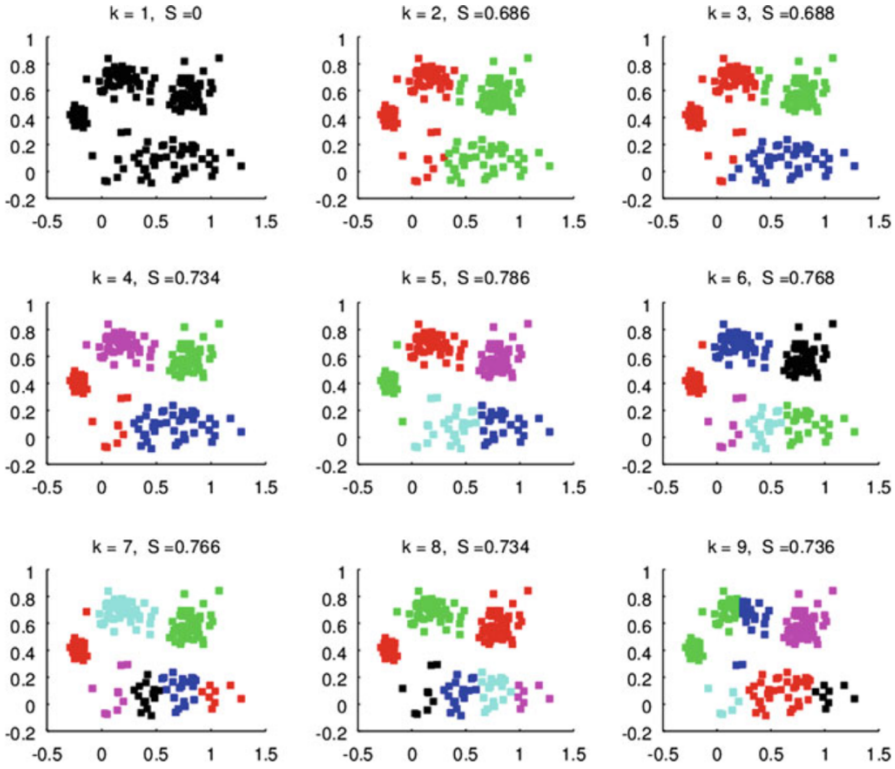
che misura quanto bene il punto $\mathbf{x}_i$ sta in un cluster. I valori possibili di $s(i)$ sono -1 se $\mathbf{x}_i$ è nel cluster sbagliato, 0 se $\mathbf{x}_i$ è tra due cluster e 1 se $\mathbf{x}_i$ è nel mezzo di un cluster ben delineato.
Con questi concetti in mente, ora cerchiamo una partizione che massimizzi il valor medio di $s(i)$ su tutti i punti (criterio SW):

$$S = \frac{1}{n} \sum_{i=1}^{n} s(i)$$

Lavorando con il clustering k-means, questa massimizzazione può essere fatto con l'algoritmo OMRk (Ordered Multiple Runs k):

OMRK($\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, p, k_{max}$)
$S^* = -\infty$
For $k = 2$ **To** k_{max}
  For i=1 **To** p
    Generate a random partition with k clusters (initialization)
    Obtain a partition P with k-means
    Determine S for P
    If $S > S^*$ **Then**
      $S^* = S$; $k^* = k$; $P^* = P$
Return (k^*, P^*)

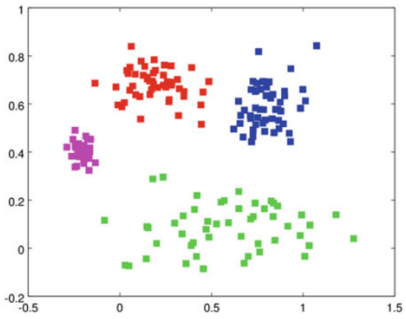
Questo algoritmo applica ripetutamente *k*-means per diversi valori di *k*. Poiché il risultato di *k*-means dipende fortemente dalla sua inizializzazione, per ogni *k*, vengono provate *p* diverse inizializzazioni casuali nel ciclo interno, e quindi la funzione restituisce l'ottimo *k** e la corrispondente partizione migliore *P**. L'algoritmo OMRk può essere utilizzato anche con altri algoritmi di clustering come l'algoritmo EM e il clustering gerarchico.



Risultati dell'algoritmo OMRk per k = da 2 a 9. Il miglior valore di S = 0,786 è stato trovato con k = 5.

Il diagramma in alto a sinistra nella figura sopra mostra un insieme di punti dati bidimensionali con quattro cluster evidenti. L'algoritmo OMRk è stato eseguito su questi dati con p = 30 e k_{max} = 9. Negli otto diagrammi seguenti, la figura mostra la partizione migliore insieme alla sua qualità S per ogni k. L'algoritmo trova il valore massimo S = 0,786 a k = 5. Ciò non riflette il raggruppamento naturale (per l'occhio umano) dei punti in quattro gruppi. Nella partizione trovata per k = 4, diversi punti che dovrebbero appartenere al cluster blu vengono assegnati al cluster rosso. Questo perché k-means riduce al minimo la distanza dal punto centrale del cluster e i punti sono più vicini al centro del cluster rosso. La maggiore densità di punti nel cluster rosso non viene presa in considerazione.

L'algoritmo EM, che può approssimare la differenza di densità di punti utilizzando una distribuzione normale, ha prestazioni significativamente migliori. Come mostrato nella figura seguente, l'algoritmo EM trova quasi esattamente la stessa distribuzione naturale di cui sopra per k = 4.



Infine, dovremmo ribadire che tutti i metodi che abbiamo descritto sono solo algoritmi di ricerca greedy euristici che non esplorano l'intero spazio di tutte le partizioni. Il criterio di SW descritto è solo una stima euristica della "qualità" di una partizione. Non può esserci una misura assoluta della qualità per le partizioni perché, come abbiamo mostrato anche in un esempio bidimensionale così semplice, persone diverse possono in certi casi preferire raggruppamenti molto diversi.

Riferimenti bibliografici

[1] Mitchell, T. *Machine Learning*. McGraw Hill, 1997. http://www-2.cs.cmu.edu/tom/mlbook.html

[2] Minsky, M. e S. Papert, Perceptrons (MIT Press, Cambridge, MA, 1969)