Riassunti e Appunti di Introduzione al Data Mining

Emanuele Galiano
Studente di Informatica
Università di Catania

Anno Accademico 2025/2026

Indice

1 Prerequisiti Matematici Essenziali **1**

1.1 Orientamento e notazione . . . 1
1.2 Vettori e matrici . . . 1
1.2.1 Combinazioni lineari e prodotto matrice–vettore . . . 1
1.2.2 Prodotto matrice–matrice . . . 1
1.3 Distanze e similarità . . . 2
1.3.1 Norme classiche . . . 2
1.3.2 Prodotto scalare e angolo . . . 2
1.3.3 Jaccard per insiemi . . . 2
1.4 Sottospazi, basi e rango . . . 2
1.5 Proiezioni ortogonali e minimi quadrati . . . 2
1.5.1 Proiezione su una direzione . . . 2
1.5.2 Minimi quadrati in due righe . . . 2
1.6 Autovalori e autovettori . . . 3
1.7 Probabilità e statistica . . . 3
1.7.1 Attesa e varianza . . . 3
1.7.2 Covarianza e correlazione . . . 3
1.7.3 Quantili e IQR . . . 3
1.7.4 Modello di Bayes e tipi di probabilità . . . 3
1.8 Preprocessing numerico . . . 4
1.9 Combinatoria utile . . . 4
1.10 Entropia . . . 5
1.10.1 Definizione . . . 5
1.10.2 In parole più semplici . . . 5

2 Introduzione al Data Mining **7**

2.1 Definizione e finalità . . . 7
2.2 Caratteristiche dei pattern . . . 7
2.3 Metodi di data mining . . . 7
2.4 Perché fare data mining . . . 8
2.4.1 Big Data . . . 8
2.4.2 Dai dati alla conoscenza e alle comunità coinvolte . . . 8
2.5 Limiti e insidie del data mining . . . 8

2.5.1 Caso di studio: Total Information Awareness (TIA) 8
2.5.2 Esempio: co-presenza in hotel come criterio di sospetto 9
2.6 Principio di Bonferroni e test multipli 10

3 Preprocessing dei Dati **11**
3.1 Perché il preprocessing è essenziale 11
3.2 Estrazione di feature 11
3.3 Portabilità dei dati 12
3.3.1 Da numerico a categoriale: discretizzazione 12
3.3.2 Da categoriale a numerico 12
3.3.3 Da testo a numerico 12
3.4 Data cleaning 12
3.4.1 Valori mancanti 13
3.4.2 Valori errati e inconsistenze 13
3.4.3 Rilevazione di outlier con quantili 13
3.4.4 Scaling e normalizzazione 13
3.5 Riduzione dei dati 14
3.5.1 Sampling 14
3.5.2 Selezione di feature 14
3.5.3 Riduzione tramite rotazione di assi: PCA 14
3.5.4 Singular Value Decomposition (SVD) 15
3.5.5 Latent Semantic Analysis (LSA) 15
3.6 Riduzione per trasformazione dei dati 16

4 Insiemi Frequenti e Regole d'Associazione **17**
4.1 Market-basket model e definizioni 17
4.2 Regole d'associazione 17
4.2.1 Qualità di una regola 17
4.2.2 Mini-esempio (dataset giocattolo) 18
4.3 Insiemi frequenti chiusi e massimali 18
4.4 Proprietà anti-monotona e Principio di Apriori 18
4.5 Algoritmo Apriori 19
4.5.1 Apriori: esempio (minsup = 2) 19
4.5.2 Generazione dei candidati 20
4.6 Ottimizzazioni di Apriori 21
4.6.1 Hashing in bucket: PCY 21
4.6.2 Partizionamento del DB: SON 21
4.6.3 Campionamento e frontiera negativa: Toivonen 21
4.7 Perché andare oltre Apriori 21
4.8 FP-Growth: idea di base 21
4.8.1 Costruzione dell'FP-tree 22
4.8.2 Esempio di FP-Growth 22
4.9 Confronto: FP-Growth vs Apriori 24

5 Clustering **25**
5.1 Concetti generali . . . 25
5.1.1 Spazi metrici e funzioni distanza . . . 25
5.1.2 Tassonomia degli algoritmi . . . 26
5.1.3 Alta dimensionalità: equidistanza e ortogonalità . . . 26
5.2 Clustering gerarchico . . . 27
5.2.1 Distanza tra cluster (*linkage*) . . . 27
5.2.2 Dendrogramma e criteri di stop . . . 27
5.2.3 Altri criteri di combinazione . . . 27
5.2.4 Versioni divisive . . . 28
5.2.5 Complessità e ottimizzazioni . . . 28
5.3 Clustering partizionale: k-means . . . 29
5.3.1 Algoritmo base . . . 29
5.3.2 Inizializzazione . . . 29
5.3.3 Funzione obiettivo e arresto . . . 29
5.3.4 Scelta del numero di cluster k . . . 30
5.3.5 Complessità computazionale . . . 31
5.4 Clustering per densità . . . 31
5.4.1 DBSCAN . . . 31
5.4.2 OPTICS . . . 32
5.4.3 HDBSCAN . . . 35

6 Classificazione **39**
6.1 Introduzione . . . 39
6.1.1 Schema generale di un classificatore . . . 39
6.1.2 Requisiti desiderabili . . . 39
6.2 Alberi decisionali . . . 40
6.2.1 Classificazione tramite albero . . . 40
6.2.2 Costruzione top–down . . . 40
6.2.3 Splitting degli attributi . . . 41
6.2.4 Scelta dell'attributo e strategia greedy . . . 41
6.3 Misure di goodness . . . 41
6.3.1 Information Gain (ID3) . . . 41
6.3.2 Gain Ratio (C4.5) . . . 42
6.3.3 Gini Index (CART) . . . 43
6.3.4 Pruning degli alberi . . . 43
6.4 Classificatori generativi . . . 45
6.4.1 Teorema di Bayes e regola di decisione . . . 46
6.4.2 Naive Bayes . . . 46
6.4.3 Reti Bayesiane . . . 47
6.5 Classificatori discriminativi . . . 47
6.5.1 Classificazione lineare e non lineare . . . 48
6.5.2 Perceptron . . . 48

6.5.3 Support Vector Machines (SVM) 50

Capitolo 1

Prerequisiti Matematici Essenziali

1.1 Orientamento e notazione

Pensiamo a un vettore come a una freccia nello spazio (coordinate) e a una matrice come a un "trasformatore" di vettori.

- Vettori colonna in grassetto: $\mathbf{x} \in \mathbb{R}^d$. Matrici in maiuscolo: $A \in \mathbb{R}^{m \times n}$. Trasposta: $A^\top$.
- Prodotto scalare: $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y}$. Norme: $\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$, $\|\mathbf{x}\|_1 = \sum_i |x_i|$, $\|\mathbf{x}\|_\infty = \max_i |x_i|$.
- Identità: I. Vettore nullo: $\mathbf{0}$.

1.2 Vettori e matrici

1.2.1 Combinazioni lineari e prodotto matrice–vettore

Una combinazione lineare di $\mathbf{v}_1, \ldots, \mathbf{v}_k$ è $\sum_i \alpha_i \mathbf{v}_i$. Il prodotto $A\mathbf{x}$ è una combinazione delle colonne di A con pesi i componenti di $\mathbf{x}$.

Esempio. Se $A = \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix}$ e $\mathbf{x} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$, allora $A\mathbf{x} = 3 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 5 \\ -1 \end{bmatrix}$.

1.2.2 Prodotto matrice–matrice

La riga i di AB si ottiene moltiplicando la riga i di A per ogni colonna di B. Non è commutativo.

Esempio. $\begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 5 & 4 \end{bmatrix}$, ma invertendo l'ordine il risultato cambia.

1.3 Distanze e similarità

1.3.1 Norme classiche

$$\|\mathbf{x}\|_1 = \sum_i |x_i|, \qquad \|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}, \qquad \|\mathbf{x}\|_\infty = \max_i |x_i|. \tag{1.1}$$

Esempio. Per $\mathbf{x} = (3, -4)$: $\|\mathbf{x}\|_1 = 7$, $\|\mathbf{x}\|_2 = 5$, $\|\mathbf{x}\|_\infty = 4$.

1.3.2 Prodotto scalare e angolo

$$\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\|_2 \, \|\mathbf{y}\|_2 \cos\theta \;\Rightarrow\; \cos\theta = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|_2 \, \|\mathbf{y}\|_2}.$$

Esempio (cosine). $\mathbf{x} = (1, 0, 1)$, $\mathbf{y} = (1, 1, 0)$: $\mathbf{x}^\top \mathbf{y} = 1$, $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = \sqrt{2} \Rightarrow \cos\theta = 1/2$.

1.3.3 Jaccard per insiemi

Per insiemi A, B: $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. Utile con basket o set di parole.

1.4 Sottospazi, basi e rango

Lo span di $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ è l'insieme delle combinazioni lineari possibili. Una base è un insieme indipendente che genera lo spazio.

Rango. $\operatorname{rank}(A)$ conta quante direzioni indipendenti contengono le colonne (o righe) di A.

Esempio. In $\mathbb{R}^2$, $(1, 0)$ e $(2, 0)$ sono dipendenti (stessa direzione): rango 1. $(1, 0)$ e $(0, 1)$ sono indipendenti: rango 2.

1.5 Proiezioni ortogonali e minimi quadrati

1.5.1 Proiezione su una direzione

Se $\mathbf{u}$ è unitario, $\operatorname{proj}_{\mathbf{u}}(\mathbf{x}) = (\mathbf{u}^\top \mathbf{x})\, \mathbf{u}$.

Esempio (retta $y = x$). $\mathbf{u} = \frac{1}{\sqrt{2}}(1, 1)$, $\mathbf{x} = (2, 0)$. Allora $\mathbf{u}^\top \mathbf{x} = \sqrt{2}$ e la proiezione è $\sqrt{2}\, \mathbf{u} = (1, 1)$.

1.5.2 Minimi quadrati in due righe

Dato $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$ e $\mathbf{b}$, risolvi $\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2$. Le equazioni normali sono $A^\top A\, \mathbf{x} = A^\top \mathbf{b}$.

Esempio. $A = \begin{bmatrix}1\\1\\1\end{bmatrix}$, $\mathbf{b} = \begin{bmatrix}2\\3\\4\end{bmatrix}$. Qui x è lo scalare che approssima nel senso LS: $A^\top A = 3$, $A^\top \mathbf{b} = 9 \Rightarrow x = 3$.

1.6 Autovalori e autovettori

Per $A \in \mathbb{R}^{n\times n}$, $A\mathbf{v} = \lambda\mathbf{v}$ significa che $\mathbf{v}$ è una direzione lasciata invariata (a fattore λ). Se $C = C^\top$ è simmetrica:

- gli autovalori λ sono reali;
- autovettori di autovalori diversi sono ortogonali;
- esiste P ortogonale con $C = P\,\Lambda\,P^\top$ (teorema spettrale).

Esempio. $C = \begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}$: autovalori $3, 1$ con autovettori proporzionali a $(1,1)$ e $(1,-1)$.

1.7 Probabilità e statistica

1.7.1 Attesa e varianza

Per variabile discreta X: $\mathbb{E}[X] = \sum_x x\,P(X = x)$, $\mathrm{Var}(X) = \mathbb{E}[(X-\mu)^2]$. Linearità: $\mathbb{E}[aX + bY] = a\,\mathbb{E}[X] + b\,\mathbb{E}[Y]$.

Esempio. Dado equo: $\mu = 3{,}5$; $\mathrm{Var}(X) = \frac{35}{12}$.

1.7.2 Covarianza e correlazione

$$\mathrm{Cov}(X,Y) = \mathbb{E}[(X-\mu_X)(Y-\mu_Y)], \qquad \rho = \frac{\mathrm{Cov}(X,Y)}{\sigma_X\sigma_Y}.$$

Matrice di covarianza (dati centrati). Per $X \in \mathbb{R}^{n\times d}$: $C = \frac{1}{n}X^\top X$.

Esempio. Dati centrati $(1,2),(2,3),(3,4)$ su due feature: $C = \left[\begin{smallmatrix}1 & 1\\1 & 1\end{smallmatrix}\right]$.

1.7.3 Quantili e IQR

Il p-quantile è la soglia sotto cui cade una frazione p dei dati ordinati (mediana = 50%). L'IQR è $Q_3 - Q_1$. Regola outlier: $[Q_1 - 1{,}5\,\mathrm{IQR},\ Q_3 + 1{,}5\,\mathrm{IQR}]$.

Esempio. Dati ordinati $[3,5,7,8,9,10,13,15,20]$: $Q_1 \approx 6$, mediana $= 9$, $Q_3 \approx 14$, IQR $= 8$.

1.7.4 Modello di Bayes e tipi di probabilità

Il **modello di Bayes** spiega come aggiornare in modo coerente la probabilità di un'ipotesi quando osserviamo nuovi dati.

Tipi di probabilità.

- **Congiunta** $P(X, Y)$: probabilità che due eventi/variabili si verifichino insieme.
- **Marginale** $P(X)$: probabilità "totale" su X ottenuta dalla congiunta, ad es. $P(X) = \sum_y P(X, Y)$ (discreto) oppure $P(X) = \int f_{X,Y}(x, y)\, dy$ (continuo).
- **Condizionata** $P(A \mid B) = \dfrac{P(A \cap B)}{P(B)}$ (per $P(B) > 0$); in termini di variabili $P(X \mid Y) = \dfrac{P(X, Y)}{P(Y)}$.
- **Indipendenza** $X \perp Y$ se e solo se $P(X, Y) = P(X)P(Y)$ (equivale a $P(X \mid Y) = P(X)$).

Teorema di Bayes. Per ipotesi $h_1, \ldots, h_K$ e un'osservazione x:

$$P(h_k \mid x) = \frac{\underbrace{P(x \mid h_k)}_{\text{verosimiglianza}}\ \underbrace{P(h_k)}_{\text{priori}}}{\underbrace{P(x)}_{\text{evidenza}}}, \qquad P(x) = \sum_{j=1}^{K} P(x \mid h_j)P(h_j).$$

Il *posterior* $P(h_k \mid x)$ è la nostra opinione aggiornata su h_k; $P(h_k)$ è il *prior*; $P(x \mid h_k)$ è la *verosimiglianza* del dato sotto h_k; $P(x)$ è l'*evidenza* (normalizzatore). Per variabili continue, si sostituiscono le somme con integrali.

1.8 Preprocessing numerico

Standardizzazione (z-score). $z_{ij} = \dfrac{x_{ij} - \mu_j}{\sigma_j}$. Quando usarla: feature con scale diverse.

Min–max scaling. $y_{ij} = \dfrac{x_{ij} - \min_j}{\max_j - \min_j} \in [0, 1]$. Nota: sensibile agli outlier.

Robust scaling. $r_{ij} = \dfrac{x_{ij} - \text{mediana}_j}{\text{IQR}_j}$. Pro: robusto ai valori estremi.

Codifiche categoriali. One-hot: per categorie $\{c_1, \ldots, c_g\}$, $c_i \mapsto \mathbf{e}_i \in \{0, 1\}^g$.

Esempio. Se media=50 e dev.stand.=10, il valore 65 diventa $z = 1{,}5$; con min–max su $[0, 100]$ diventa 0,65.

1.9 Combinatoria utile

Coefficienti binomiali. $\binom{n}{k} = \dfrac{n!}{k!(n-k)!}$. Esempio: 3-itemset da 100 item $= \binom{100}{3} = 161700$.

Permutazioni. $P(n, k) = n\,(n-1)\cdots(n-k+1)$: sequenze ordinate senza ripetizione.

1.10 Entropia

1.10.1 Definizione

Sia X una variabile aleatoria discreta che assume valori in un insieme finito $\mathcal{X}$, con massa di probabilità $p_X(x) = \Pr[X = x]$. L'**entropia (di Shannon)** di X è

$$H_b(X) \;=\; -\sum_{x\in\mathcal{X}} p_X(x)\,\log_b p_X(x),$$

dove b è la base del logaritmo (tipicamente $b = 2$, quindi l'unità di misura è il *bit*; se $b = e$ l'unità è il *nat*). Per convenzione $0 \log 0 := 0$.

Proprietà essenziali.

$0 \le H_b(X) \le \log_b |\mathcal{X}|, \qquad H_b(X) = 0 \;\Leftrightarrow\; X$ è deterministica, $\qquad H_b(X) = \log_b |\mathcal{X}| \;\Leftrightarrow\; X$ è uniforme.

In pratica, più è *dispersa* la distribuzione di X, maggiore è l'entropia.

Stima empirica. Dati n campioni $x_1, \ldots, x_n$ e le frequenze $n_x = |\{i : x_i = x\}|$, la stima plug–in è

$$\widehat{H}_b(X) \;=\; -\sum_{x\in\mathcal{X}} \frac{n_x}{n}\,\log_b\!\Big(\frac{n_x}{n}\Big).$$

Nel caso di etichette di classe Y, $\widehat{H}_2(Y)$ misura l'*impurità* del dataset ed è usata in molti criteri di splitting (es. *information gain*).

1.10.2 In parole più semplici

L'entropia quantifica **quanta incertezza** c'è in un fenomeno casuale: è alta quando i risultati sono imprevedibili e tutti più o meno uguali in probabilità, è bassa quando un risultato è quasi certo.

Esempio (moneta). Se una moneta è equa, testa/croce valgono 50% ciascuno: $H_2 = 1$ bit (serve, in media, 1 bit per descrivere l'esito). Se la moneta è sbilanciata, ad esempio $\Pr[\text{testa}] = 0.99$, l'entropia scende a

$$H_2 \approx -0.99 \log_2 0.99 - 0.01 \log_2 0.01 \approx 0.080 \text{ bit},$$

perché l'esito è quasi sempre testa: c'è poca incertezza.

Capitolo 2

Introduzione al Data Mining

2.1 Definizione e finalità

Il *Data Mining* consiste nello scoprire *pattern* (modelli e regolarità) interessanti e possibilmente inattesi all'interno di un insieme di dati. Le conoscenze estratte possono essere impiegate per supportare decisioni, formulare previsioni o fungere da base per ulteriori attività (ad es. profilazione di utenti).

- **Data cleaning (pre-processing)**: individuare e rimuovere artefatti e dati fittizi o rumorosi, armonizzare formati, gestire valori mancanti.
- **Visualizzazione**: comunicare in modo efficace i risultati del processo di data mining.

2.2 Caratteristiche dei pattern

I pattern da estrarre dovrebbero essere:

- **Validi**: veri (con alta probabilità) anche su dati nuovi non visti.
- **Utili**: capaci di suggerire o guidare azioni concrete.
- **Inattesi**: non banali, non ovvi.
- **Comprensibili**: interpretabili da esseri umani.

2.3 Metodi di data mining

Gli algoritmi di data mining si distinguono in:

- **Descrittivi**: mirano a rappresentare e *descrivere* la struttura dei dati (es. clustering, regole di associazione, analisi di similarità).
- **Predittivi**: usano alcune variabili per *predire* valori sconosciuti o futuri (es. classificazione, regressione, sistemi di raccomandazione).

2.4 Perché fare data mining

Negli ultimi anni la quantità di dati disponibili è esplosa. Le principali sorgenti includono:

- **Business**: web, e-commerce, transazioni, mercati finanziari, log applicativi.
- **Multimedia**: video, audio, testo, immagini.
- **Scienza**: telerilevamento, medicina, bioinformatica.
- **Società**: news, e-mail, social network, forum.

2.4.1 Big Data

I dati moderni sono spesso:

- **Grandi** (*volume*);
- **Ad elevata dimensionalità** (*varietà* di attributi);
- **Complessi** (relazionali, temporali, eterogenei).

La sola disponibilità di molti dati non si traduce automaticamente in conoscenza: servono metodi e strumenti per analizzarli in modo efficace.

2.4.2 Dai dati alla conoscenza e alle comunità coinvolte

L'enorme quantità di dati non diventa automaticamente conoscenza: occorre trasformarla con tecniche appropriate. Il data mining è al crocevia di più comunità scientifiche (apprendimento automatico, pattern recognition, visualizzazione, algoritmi, ...).

2.5 Limiti e insidie del data mining

Un'idea intuitiva (ma pericolosa) è: "raccogliamo quanti più dati possibile e troveremo pattern affidabili". In realtà, al crescere della dimensione aumenta anche la probabilità di osservare regolarità *spurie*, cioè non realmente significative. Consideriamo un caso di studio.

2.5.1 Caso di studio: Total Information Awareness (TIA)

Dopo gli attentati dell'11 settembre 2001, il Dipartimento della Difesa degli Stati Uniti propose il programma *Total Information Awareness* (TIA), volto a raccogliere in modo massivo informazioni su persone (ricevute di pagamento, spostamenti, ecc.) per prevenire attacchi terroristici. Al di là delle criticità etiche e di privacy, un rischio metodologico è la generazione di moltissimi *falsi positivi*: attività apparentemente anomale ma statisticamente spiegabili.

2.5.2 Esempio: co-presenza in hotel come criterio di sospetto

Vogliamo identificare potenziali coppie di malfattori assumendo che essi si riuniscano periodicamente nello stesso hotel. Sui dati osservati cerchiamo tutte le coppie di persone che, in *due* giorni distinti, risultano nello *stesso* hotel.

Dati di partenza.

- Numero di persone tracciate: $N = 10^9$.
- Orizzonte temporale: $D = 1000$ giorni.
- Numero di hotel: $H = 10^5$.
- Capacità per hotel e per giorno: $C = 100$ persone.

Ipotesi nulla (random). Ogni persona, in ciascun giorno, sceglie in modo casuale e indipendente se (e dove) soggiornare; in particolare, per un dato hotel in un dato giorno la probabilità che una persona vi alloggi è

$$P(\text{persona in un hotel specifico in un giorno}) \;=\; \frac{C}{N} \;=\; \frac{100}{10^9} \;=\; 10^{-7}.$$

Calcoli numerici.

1. **Stesso hotel in un giorno fissato.** Per due persone specifiche p, q, la probabilità di trovarle nello stesso hotel in un *giorno specifico* è

$$P_1 \;=\; H \cdot \left(\frac{C}{N}\right)^2 \;=\; 10^5 \cdot (10^{-7})^2 \;=\; 10^{-9}.$$

2. **Due giorni distinti non specificati.** Le coppie di giorni distinti sono $\binom{D}{2} = \frac{D(D-1)}{2} \approx \frac{1000 \cdot 999}{2} \approx 5 \cdot 10^5$. Supponendo indipendenza tra i giorni, la probabilità che p, q si trovino nello stesso hotel in *entrambi* i due giorni è

$$P_2 \;=\; \binom{D}{2} \cdot P_1^2 \;=\; \left(5 \cdot 10^5\right) \cdot (10^{-9})^2 \;=\; 5 \cdot 10^{-13}.$$

3. **Numero atteso di coppie sospette.** Le coppie distinte di persone sono $\binom{N}{2} \approx \frac{10^9(10^9-1)}{2} \approx 5 \cdot 10^{17}$. Il numero atteso di coppie candidate è dunque

$$\mathbb{E}[\#\text{coppie}] \;=\; P_2 \cdot \binom{N}{2} \;\approx\; \left(5 \cdot 10^{-13}\right) \cdot \left(5 \cdot 10^{17}\right) \;=\; 2{,}5 \cdot 10^5.$$

Considerazioni. Verificare manualmente $\sim$ 250,000 coppie è impraticabile, specie a fronte di un numero reale di coppie colpevoli verosimilmente molto più basso. L'esempio mostra come, su dati enormi, criteri apparentemente sensati possano generare moltissimi falsi positivi *anche in assenza di segnale.*

2.6 Principio di Bonferroni e test multipli

Principio di Bonferroni. Se il numero atteso di occorrenze dell'evento cercato (sotto l'ipotesi di casualità dei dati) è significativamente *maggiore* del numero di istanze che ci si aspetta di trovare nella realtà, allora qualsiasi "pattern" osservato è più verosimilmente un *artefatto* che non un'evidenza.

Interpretazione operativa. Quando formuliamo molte ipotesi/ricerche sui dati (*multiple testing*), è necessario *correggere* il livello di significatività per tenere sotto controllo i falsi positivi. Una correzione conservativa è la *correzione di Bonferroni*: se eseguiamo m test, imponiamo che il p–value di ciascun test sia $< \alpha/m$ per mantenere il Family-Wise Error Rate al di sotto di α.

Quando applicarlo. In scenari esplorativi su grandi basi dati (come nel caso sopra), prima di agire sui "pattern" trovati occorre verificare che il loro numero sia compatibile con quanto ci si aspetterebbe per puro caso. In caso contrario, i pattern vanno trattati con sospetto e sottoposti a verifica indipendente (es. dati di conferma, A/B test, validazione su hold-out).

Capitolo 3

Preprocessing dei Dati

3.1 Perché il preprocessing è essenziale

Prima di applicare qualsiasi algoritmo di data mining, i dati grezzi devono essere resi *analizzabili*. In pratica, il preprocessing riduce la distanza tra "dati come sono" e "dati come l'algoritmo li richiede". In assenza di un'adeguata preparazione, anche il miglior algoritmo può fallire o produrre risultati fuorvianti (es. feature mal estratte, scale non confrontabili, rumore e outlier che dominano l'analisi).

Quattro obiettivi chiave.

1. **Estrazione di feature**: ricavare attributi informativi a partire da dati grezzi (log, segnali, immagini, testo, traffico di rete, ...), ed eventualmente integrare sorgenti eterogenee in un unico dataset.
2. **Portabilità/trasformazione del tipo di dato**: mappare tipi di dato verso rappresentazioni adatte agli algoritmi (es. numerico $\leftrightarrow$ categoriale, testo $\rightarrow$ vettori).
3. **Data cleaning**: trattare valori mancanti, errori, inconsistenze e outlier; riportare gli attributi su scale confrontabili (scaling/normalizzazione).
4. **Riduzione dei dati**: diminuire dimensionalità o volume preservando l'informazione utile (selezione/trasformazione di feature, sampling, PCA/SVD/LSA).

3.2 Estrazione di feature

L'estrazione di feature costruisce un set di attributi più adatti al problema. È fortemente dipendente dal dominio e incide in modo determinante sulla qualità dell'analisi.

Esempi tipici.

- **Dati sensoriali**: dopo il cleaning, si possono utilizzare direttamente i campioni o trasformarli (es. trasformata di Fourier) per ottenere feature spettrali.
- **Immagini**: matrici di pixel, istogrammi di colore, *edge features*.

- **Web logs**: record testuali formattati da cui si estraggono feature di sessione, frequenze, pattern di navigazione.
- **Traffico di rete**: conteggi, tassi, durate; utili per intrusion detection.
- **Documenti**: rappresentazione *bag-of-words* o pesi TF–IDF (§3.3.3).

3.3 Portabilità dei dati

Gli algoritmi spesso richiedono tipologie di input specifiche; la *portabilità* converte i dati nella forma necessaria. Talvolta la conversione è *lossy* (perdita di informazione): è buona norma esplicitarne il compromesso.

3.3.1 Da numerico a categoriale: discretizzazione

La discretizzazione suddivide il dominio numerico in intervalli, assegnando a ciascun valore l'etichetta dell'intervallo corrispondente.

Equi-width Intervalli di uguale ampiezza: $[a, a+w), [a+w, a+2w), \ldots$.

Equi-depth Intervalli con uguale numerosità attesa (*quantile binning*).

Equi-log Intervalli di uguale differenza su scala logaritmica.

Esempio. Età $\in [0, 80]$ in 4 *intervalli* equi-distanti da 20 anni: $[0, 20), [20, 40), [40, 60), [60, 80]$. L'età $23 \mapsto$ intervallo 2.

3.3.2 Da categoriale a numerico

Label encoding assegna un intero a categoria, ma introduce un ordine fittizio. Per modelli sensibili all'ordinalità (NN, distanze), si preferisce l'*one-hot encoding*.

Definizione (one-hot). Per g categorie si rappresenta la categoria c_i con $\mathbf{e}_i \in \{0,1\}^g$, dove $(\mathbf{e}_i)_i = 1$ e altrove 0.

Esempio. Colore $\in \{\texttt{rosso}, \texttt{verde}, \texttt{blu}\}$: rosso $\to (1,0,0)$; blu $\to (0,0,1)$.

3.3.3 Da testo a numerico

Una collezione di n documenti e d termini può essere rappresentata da una matrice *termine-documento* $X \in \mathbb{R}^{d \times n}$, con x_{tj} frequenze (o pesi TF–IDF) del termine t nel documento j. La matrice è tipicamente sparsa; tecniche di riduzione (§3.5) sono utili.

3.4 Data cleaning

Il cleaning gestisce mancanze, errori e inconsistenze, e riporta le feature su scale comparabili.

3.4.1 Valori mancanti

- **Eliminazione di record** con mancanti: semplice, ma rischiosa se i mancanti sono frequenti o non *MCAR* (*Missing Completely At Random*).
- **Imputazione**: media/mediana per attributi numerici; categoria/moda per categoriali; metodi più accurati includono k-NN, regressione, EM, modelli generativi. *Nota*: imputazioni errate possono introdurre bias.
- **Gestione nativa**: alcuni algoritmi tollerano i mancanti (es. alberi decisionali con surrogate splits).

Mini-esempio (serie temporale). Per dati temporali/spaziali è comune l'interpolazione (lineare, spline) per colmare piccole lacune.

3.4.2 Valori errati e inconsistenze

- **Deduplicate/record linkage**: rilevazione e fusione di duplicati (spesso in integrazione di sorgenti).
- **Vincoli di dominio**: coerenza semantica (es. Nazione=`USA` $\Rightarrow$ Città $\neq$ `Roma`).
- **Outlier**: punti anomali da investigare; non sempre errori, ma potenziali driver del fenomeno.

3.4.3 Rilevazione di outlier con quantili

Sia S un insieme di valori. Il quantile d'ordine $x \in [0,1]$ è q_x tale che $x\%$ di S è minore di q_x e $(1-x)\%$ maggiore o uguale.

Interquantile range (IQR). $\mathrm{IQR} = Q_3 - Q_1$. La regola classica assegna come outlier i valori

$$x < Q_1 - 1.5\,\mathrm{IQR} \quad \text{oppure} \quad x > Q_3 + 1.5\,\mathrm{IQR}.$$

Mini-esempio. Con $Q_1 = 10$, $Q_3 = 22$, $\mathrm{IQR} = 12$: soglia bassa $= 10 - 18 = -8$, soglia alta $= 22 + 18 = 40$. Valori > 40 o < -8 sono outlier.

3.4.4 Scaling e normalizzazione

Scale diverse rendono gli attributi *non confrontabili* (es. Età vs Salario). La distanza euclidea è dominata dalla scala maggiore. Due standard interventi:

Standardizzazione (z-score). Per un attributo A_j con media μ_j e deviazione standard $\sigma_j > 0$, ogni valore x_{pj} è trasformato in

$$z_{pj} = \frac{x_{pj} - \mu_j}{\sigma_j}, \quad \mu_j = \frac{1}{n}\sum_{p=1}^{n} x_{pj}, \quad \sigma_j^2 = \frac{1}{n}\sum_{p=1}^{n}(x_{pj} - \mu_j)^2.$$

Min–max scaling. Con $\min_j, \max_j$ valori minimo/massimo osservati di A_j,

$$y_{pj} = \frac{x_{pj} - \min_j}{\max_j - \min_j} \in [0, 1].$$

Nota: il min–max è sensibile agli outlier; lo z-score è più robusto ma non limita l'intervallo.

3.5 Riduzione dei dati

Ridurre volume e/o dimensionalità accelera l'analisi e migliora la generalizzazione, a parità d'informazione.

3.5.1 Sampling

Costruire un sottoinsieme rappresentativo dei dati.

- **Biased (guidato)**: privilegia porzioni ritenute più informative (es. record recenti in serie temporali).
- **Stratificato**: partiziona il dataset in strati omogenei e campiona proporzionalmente in ogni strato.

3.5.2 Selezione di feature

Rimuovere attributi irrilevanti, ridondanti o rumorosi.

- **Unsupervised**: criteri di ridondanza/varianza, filtri (es. varianza bassa) e metodi embedded.
- **Supervised**: per classificazione, scegliere feature predittive della classe (filtri basati su correlazione/*mutual information*, wrapper, embedded tipo LASSO).

3.5.3 Riduzione tramite rotazione di assi: PCA

La *Principal Component Analysis* cerca una base ortogonale in cui poche dimensioni catturano la maggior parte della varianza.

Matrice di covarianza. Dato un dataset $D \in \mathbb{R}^{M \times N}$ (righe=record, colonne=attributi), centrato per colonna, la covarianza è $C = \frac{1}{M} D^\top D \in \mathbb{R}^{N \times N}$.

Autodecomposizione. $C = P \, \Lambda \, P^\top$, con colonne di P autovettori ortonormali (componenti principali) e $\Lambda = \mathrm{diag}(\lambda_1 \geq \cdots \geq \lambda_N)$ varianze spiegate.

Trasformazione. Scegliendo P componenti $P_p = [\mathbf{p}_1 \cdots \mathbf{p}_P]$, i dati proiettati sono

$$D' = D \, P_p.$$

Scelta del numero di componenti. Si usa la frazione di varianza spiegata $\sum_{i=1}^{P} \lambda_i / \sum_{i=1}^{N} \lambda_i$ (*es.* soglia 90%), oppure un grafico a gomito (*scree plot*).

Mini-esempio (intuitivo). Con due attributi fortemente correlati (es. altezza e apertura delle braccia), la prima componente è circa la bisettrice che massimizza la varianza proiettata; la seconda è ortogonale e a bassa varianza.

3.5.4 Singular Value Decomposition (SVD)

La SVD decompone una matrice $M \in \mathbb{R}^{n\times d}$ come

$$M = U\,\Sigma\,V^\top,$$

con $U \in \mathbb{R}^{n\times n}$ e $V \in \mathbb{R}^{d\times d}$ ortogonali, e $\Sigma \in \mathbb{R}^{n\times d}$ diagonale non negativa. I valori singolari non nulli sono radici degli autovalori di $M^\top M$; il loro numero è il rango di M.

Interpretazione geometrica. In $\mathbb{R}^2$, un cerchio unitario è ruotato (da V), scalato lungo i semiassi (da Σ) e ri-ruotato (da U) in un'ellisse.

Relazione con PCA. Con dati centrati, PCA e SVD coincidono (sulle direzioni delle colonne). PCA massimizza varianza spiegata; SVD massimizza la norma di Frobenius catturata dall'approssimazione rank-k.

Versioni alternative.

- **Thin SVD** (o economy): rimuove le *colonne* di U e le *righe* di Σ in eccesso rispetto alla dimensione minima. Per $M \in \mathbb{R}^{m\times n}$ con $m \geq n$: $U_n \in \mathbb{R}^{m\times n}$, $\Sigma_n \in \mathbb{R}^{n\times n}$, $V^\top \in \mathbb{R}^{n\times n}$. Decomposizione *esatta* ($M = U_n\,\Sigma_n\,V^\top$), ma con meno memoria. *Uso tipico*: quando $m \gg n$ (o viceversa) e vuoi la SVD senza parti sicuramente nulle.
- **Compact SVD**: elimina *tutte e sole* le righe/colonne di Σ associate a valori singolari *nulli* e, di conseguenza, le colonne di U e le righe di $V^\top$ corrispondenti. Se $\mathrm{rank}(M) = r < \min(m,n)$: $U_r \in \mathbb{R}^{m\times r}$, $\Sigma_r \in \mathbb{R}^{r\times r}$, $V_r^\top \in \mathbb{R}^{r\times n}$. Decomposizione ancora *esatta*. *Uso tipico*: matrici a rango basso (colonne/righe dipendenti), per togliere zeri inutili.
- **Truncated SVD** (rank-k): mantiene solo i *primi* k valori singolari ($k < r$) e tronca il resto: $U_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, $V_k^\top \in \mathbb{R}^{k\times n}$. Decomposizione *non esatta*: $M \approx U_k\,\Sigma_k\,V_k^\top$, migliore approssimazione a rango k (in norma 2/Frobenius). *Uso tipico*: riduzione dimensionale, denoising, LSA/LSI, raccomandazione (si conserva la maggior parte della "energia").

3.5.5 Latent Semantic Analysis (LSA)

LSA applica la SVD alla matrice termine-documento (§3.3.3). Troncando ai primi k valori singolari si ottiene uno spazio semantico di bassa dimensione che attenua sinonimia/rumore e migliora il recupero di informazione.

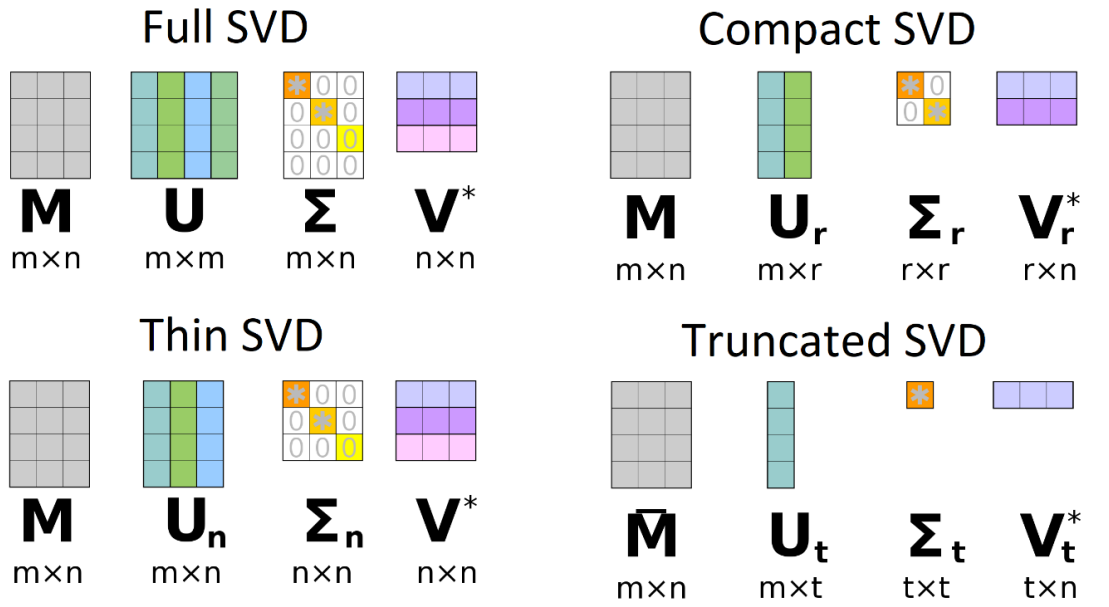

Figura 3.1: Confronto tra varianti della SVD. Full: $U \in \mathbb{R}^{m \times m}$, $\Sigma \in \mathbb{R}^{m \times n}$, $V^\top \in \mathbb{R}^{n \times n}$. Compact/Thin: $U_r \in \mathbb{R}^{m \times r}$, $\Sigma_r \in \mathbb{R}^{r \times r}$, $V_r^\top \in \mathbb{R}^{r \times n}$. Truncated: $U_t \in \mathbb{R}^{m \times t}$, $\Sigma_t \in \mathbb{R}^{t \times t}$, $V_t^\top \in \mathbb{R}^{t \times n}$ con $t < r$.

3.6 Riduzione per trasformazione dei dati

Ridurre dimensionalità trasformando i dati in forme più compatte e *strutturate*:

- **Serie temporali**: trasformata di Fourier, trasformata wavelet di Haar.
- **Grafi**: embedding spettrali/*multidimensional scaling* per preservare distanze/similarità tra nodi.

Capitolo 4

Insiemi Frequenti e Regole d'Associazione

4.1 Market-basket model e definizioni

Nel *market-basket model* ogni transazione (*basket*) è un insieme di oggetti (*item*). L'obiettivo è individuare **itemset frequenti**, cioè insiemi di item che compaiono assieme in molte transazioni, e derivarne **regole d'associazione** utili per descrivere co-occorrenze interessanti.

Supporto. Sia $\mathcal{D}$ l'insieme dei basket (con $|\mathcal{D}| = N$) e sia $I = \{i_1, \ldots, i_k\}$ un itemset. Il **supporto** assoluto di I è

$$\text{supp}(I) = |\{T \in \mathcal{D} \,:\, I \subseteq T\}|, \qquad \text{supp}_{\text{rel}}(I) = \frac{\text{supp}(I)}{N}.$$

Dato un valore soglia σ (*min-sup*), I è detto **frequente** se $\text{supp}(I) \geq \sigma$.

Soglia di supporto: trade-off. Una soglia troppo alta può eliminare pattern interessanti; una troppo bassa produce un'esplosione di candidati difficili da analizzare e validare.

4.2 Regole d'associazione

Una **regola d'associazione** è un'implicazione $X \to j$, con X itemset e j un singolo item con $j \notin X$. Si estende naturalmente a $X \to Y$ con $X \cap Y = \varnothing$.

4.2.1 Qualità di una regola

Confidenza. Con $\text{supp}(\cdot)$ definito sopra, la confidenza di $X \to j$ è

$$\text{conf}(X \to j) = \frac{\text{supp}(X \cup \{j\})}{\text{supp}(X)} \;=\; P(j \mid X).$$

Coverage. $\text{supp}(X)$ è detto anche *coverage*: misura quanto spesso è applicabile la regola.

Interesse (o *interest*). Quantifica l'influenza di X su j come scostamento dalla prevalenza marginale di j:

$$\mathrm{int}(X \to j) = \mathrm{conf}(X \to j) - \frac{\mathrm{supp}(\{j\})}{N}.$$

Lift. Confronta la co-occorrenza osservata con quella attesa in caso di indipendenza:

$$\mathrm{lift}(X \to j) = \frac{N \cdot \mathrm{supp}(X \cup \{j\})}{\mathrm{supp}(X)\,\mathrm{supp}(\{j\})} \;=\; \frac{\mathrm{conf}(X \to j)}{\mathrm{supp}(\{j\})/N}.$$

Valori > 1 indicano associazione positiva, < 1 negativa.

Nota. Supporto e confidenza alti non implicano necessariamente interesse: regole ovvie (es. pasta, pomodoro $\to$ pasta) possono essere poco informative.

4.2.2 Mini-esempio (dataset giocattolo)

Sia $N = 8$ e consideriamo item $\{b, c, j, m, p\}$. Supponiamo che $\mathrm{supp}(\{b\}) = 6$, $\mathrm{supp}(\{c\}) = 5$, $\mathrm{supp}(\{j\}) = 4$, $\mathrm{supp}(\{m\}) = 5$, $\mathrm{supp}(\{p\}) = 2$ e, tra le coppie, $\mathrm{supp}(\{b, c\}) = 4$, $\mathrm{supp}(\{c, j\}) = 3$, $\mathrm{supp}(\{c, m\}) = 2$, $\mathrm{supp}(\{m, p\}) = 2$, ecc. Per la regola $\{c, m\} \to b$ si ha

$$\mathrm{conf} = \tfrac{\mathrm{supp}(\{b,c,m\})}{\mathrm{supp}(\{c,m\})} = \tfrac{2}{2} = 1.0, \quad \mathrm{lift} = \frac{8 \cdot 2}{2 \cdot 6} = 1.33\,.$$

4.3 Insiemi frequenti chiusi e massimali

Sia I frequente.

- I è **chiuso** se nessun suo super-insieme ha lo *stesso* supporto di I.
- I è **massimale** se nessun suo super-insieme è frequente.

Gli insiemi massimali sono (per definizione) chiusi; gli insiemi chiusi sono un sottoinsieme degli insiemi frequenti e consentono una rappresentazione più compatta senza perdere il supporto degli insiemi chiusi stessi.

4.4 Proprietà anti-monotona e Principio di Apriori

Per itemset $S \subseteq I$ vale l'**anti-monotonicità del supporto**:

$$\mathrm{supp}(I) \leq \mathrm{supp}(S).$$

Da cui il **Principio di Apriori**: se un itemset I è frequente, ogni suo sottoinsieme è frequente; equivalentemente, se I non è frequente, nessun suo super-insieme può esserlo. Questa proprietà consente un *pruning* efficace dello spazio dei candidati.

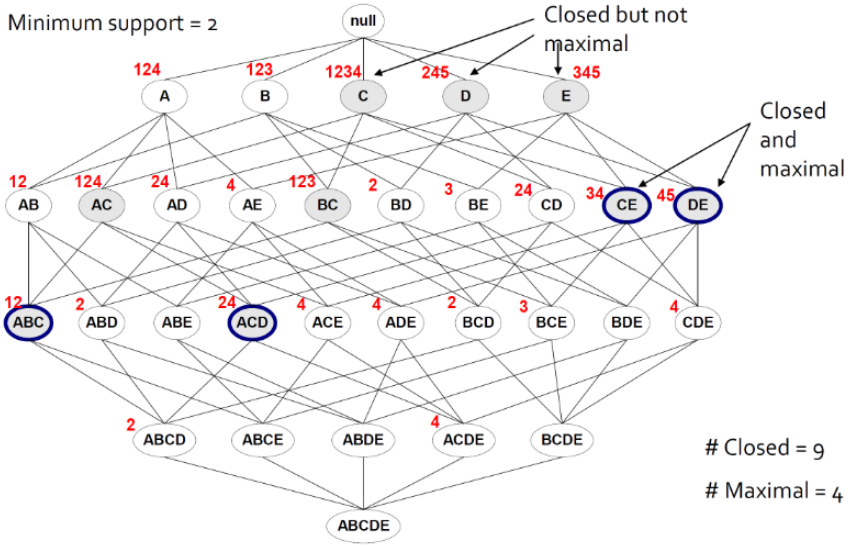

Figura 4.1: Grafo degli itemset con minsup = 2. Un itemset è *frequente* se il suo supporto è ≥ 2; è *chiuso* se nessun superinsieme ha lo stesso supporto; è *massimale* se nessun superinsieme è frequente. Nell'esempio: alcuni chiusi (es. CE) e chiusi–massimali (CE, DE); conteggi indicati: #closed = 9, #maximal = 4.

4.5 Algoritmo Apriori

Ricerca bottom-up per cardinalità crescente.

1. Calcola l'insieme L_1 degli item singoli frequenti.
2. Per $k = 1, 2, \ldots$:
   (a) **Join**: genera C_{k+1} (candidati di taglia k+1) con self-join di L_k.
   (b) **Prune**: elimina da C_{k+1} gli itemset che contengono sottoinsiemi di taglia k non frequenti (per §4.4).
   (c) **Conteggio**: calcola supp$(\cdot)$ dei candidati scorrendo il DB e costruisci $L_{k+1} = \{c \in C_{k+1} : \text{supp}(c) \geq \sigma\}$.
3. Arresta quando $C_{k+1} = \varnothing$.

4.5.1 Apriori: esempio (minsup = 2)

Consideriamo $N = 8$ basket e gli item $\{b, c, j, m, p\}$. I supporti degli item singoli sono:

Item	b	c	j	m	p
supp$(\cdot)$	6	5	4	5	2

Con minsup = 2, tutti e cinque gli item sono frequenti, quindi $L_1 = \{b, c, j, m, p\}$. *(Dati come nelle slide).*

Passo $k = 1 \to 2$: generazione C_2 e pruning. C_2 si ottiene con self-join di L_1 e contiene tutte le coppie possibili:

$$\{b,c\},\{b,j\},\{b,m\},\{b,p\},\{c,j\},\{c,m\},\{c,p\},\{j,m\},\{j,p\},\{m,p\}.$$

Dai conteggi nel DB (come in tabella delle slide) si ottengono i supporti:

Itemset	$\{b,c\}$	$\{b,j\}$	$\{b,m\}$	$\{b,p\}$	$\{c,j\}$	$\{c,m\}$	$\{c,p\}$	$\{j,m\}$	$\{j,p\}$	$\{m,p\}$
supp$(\cdot)$	4	2	4	1	3	2	0	2	1	2

Applicando minsup, otteniamo $L_2 = \{\{b,c\},\{b,j\},\{b,m\},\{c,j\},\{c,m\},\{j,m\},\{m,p\}\}$.

Passo $k = 2 \to 3$: generazione C_3 da L_2 (self-join) e pruning. Si combinano coppie con i primi $k-1$ item uguali (ordine lessicografico) e si rimuovono i candidati che hanno qualche sottoinsieme di taglia 2 non in L_2 (Principio di Apriori, §4.4). I candidati che restano sono:

$$C_3 = \{\{b,c,j\},\{b,c,m\},\{b,j,m\},\{c,j,m\}\}.$$

Conteggiando i supporti sul DB (slide):

Itemset	$\{b,c,j\}$	$\{b,c,m\}$	$\{b,j,m\}$	$\{c,j,m\}$
supp$(\cdot)$	2	2	1	1

Quindi $L_3 = \{\{b,c,j\},\{b,c,m\}\}$.

Passo $k = 3 \to 4$: generazione C_4 e arresto. L'unico candidato unibile è $\{b,c,j,m\}$, ma il suo supporto vale $1 < 2$, dunque non è frequente e $L_4 = \varnothing$. L'algoritmo termina.

Riassunto dell'esempio.

$$L_1 = \{b,c,j,m,p\}, \quad L_2 = \{\{b,c\},\{b,j\},\{b,m\},\{c,j\},\{c,m\},\{j,m\},\{m,p\}\},$$

$$L_3 = \{\{b,c,j\},\{b,c,m\}\}, \quad L_4 = \varnothing.$$

L'anti-monotonicità del supporto permette il *pruning* efficace a ogni livello, riducendo drasticamente i candidati da contare.

4.5.2 Generazione dei candidati

Se gli item sono ordinati, due insiemi $A = (a_1, \ldots, a_{k-1}, x)$ e $B = (a_1, \ldots, a_{k-1}, y)$ in L_k con $x < y$ producono il candidato $(a_1, \ldots, a_{k-1}, x, y)$. Il passo di *prune* scarta i candidati che hanno almeno un sottoinsieme di taglia k non presente in L_k.

Esempio (schema). Da $L_2 = \{\{b,c\},\{b,j\},\{b,m\},\{c,j\},\{c,m\},\{j,m\},\{m,p\}\}$ si generano candidati di taglia 3 come $\{b,c,j\}$, $\{b,c,m\}$, $\{b,j,m\}$, $\{c,j,m\}$, ecc., poi si eliminano quelli che contengono coppie non frequenti.

4.6 Ottimizzazioni di Apriori

4.6.1 Hashing in bucket: PCY

Alla prima passata si contano i singoli item e, parallelamente, si proiettano tutte le coppie in bucket tramite una funzione hash. I bucket con supporto sotto soglia vengono marcati come non frequenti: alla seconda passata, una coppia (i, j) è candidata solo se *entrambi* gli item sono frequenti e il bucket hash di (i, j) è frequente. Ciò riduce notevolmente $|C_2|$.

4.6.2 Partizionamento del DB: SON

Divide il dataset in partizioni; su ciascuna partizione si esegue Apriori con min-sup scalato (proporzionale alla frazione di transazioni della partizione). L'unione degli insiemi frequenti locali fornisce i candidati globali, che vengono poi verificati su tutto il DB. L'algoritmo è adatto a calcolo distribuito.

4.6.3 Campionamento e frontiera negativa: Toivonen

Si esegue Apriori su un campione casuale S con soglia più bassa (σ'); si ottiene un insieme di itemset frequenti in S e la *frontiera negativa*: insiemi non frequenti in S i cui *immediati* sottoinsiemi sono frequenti. Se nessun elemento della frontiera negativa risulta frequente sull'intero DB, i frequenti di S sono la risposta; altrimenti si ripete con un nuovo campione (per evitare falsi negativi), regolando σ'.

4.7 Perché andare oltre Apriori

Apriori richiede (i) generare esplicitamente i candidati C_k a ogni livello e (ii) più passate sul database per calcolare i supporti. Con soglie basse o molti pattern, il numero di candidati esplode e le scansioni diventano costose. **FP-Growth** evita entrambi: rappresenta il DB in modo compatto (*FP-tree*) e *fa crescere* i pattern frequenti senza generare C_k.

4.8 FP-Growth: idea di base

1. **Costruzione FP-tree** (*Frequent Pattern tree*): scansiona il DB per ottenere i supporti degli item, scarta quelli con supporto $< \sigma$, ordina gli item per supporto decrescente e inserisci le transazioni nell'albero condividendo i prefissi comuni. Mantieni una *header table* con link ai nodi per item.
2. **Pattern-growth**: per ogni item x (dall'ultimo al primo nell'ordine per supporto) estrai la *pattern base condizionale* di x dai cammini che portano a x, costruisci l'*FP-tree condizionale* e ripeti ricorsivamente. I pattern trovati si concatenano con x.

Servono in genere **due passate** sul DB (una per i conteggi degli item, una per costruire l'albero); poi si lavora su strutture in memoria.

4.8.1 Costruzione dell'FP-tree

1. **Prima passata**: calcola $\text{supp}(i)$ per ogni item; elimina gli item con $\text{supp}(i) < \sigma$.
2. **Ordina** gli item per supporto decrescente (tie-break fisso) e **riordina** ogni transazione seguendo lo stesso ordine.
3. **Inserisci** ciascuna transazione nell'albero a partire dalla radice: percorri/crea i nodi lungo il prefisso ordinato, incrementando i contatori dei nodi e aggiornando i *node link* nella header table.

Proprietà: l'FP-tree conserva l'informazione necessaria a ricostruire i supporti dei pattern frequenti ed è molto compatto se molte transazioni condividono prefissi.

4.8.2 Esempio di FP-Growth

Soglia $\sigma = 3$. Dalla prima passata otteniamo gli item frequenti (con supporto) in ordine decrescente:

$$f : 4, \quad c : 4, \quad a : 3, \quad b : 3, \quad m : 3, \quad p : 3.$$

Ogni transazione viene **riordinata** secondo l'ordine $f \succ c \succ a \succ b \succ m \succ p$ ed **inserita** nell'FP-tree, aggregando i prefissi per incrementare i contatori.

Header table iniziale.

Item	f	c	a	b	m	p
$\text{supp}(\cdot)$	4	4	3	3	3	3

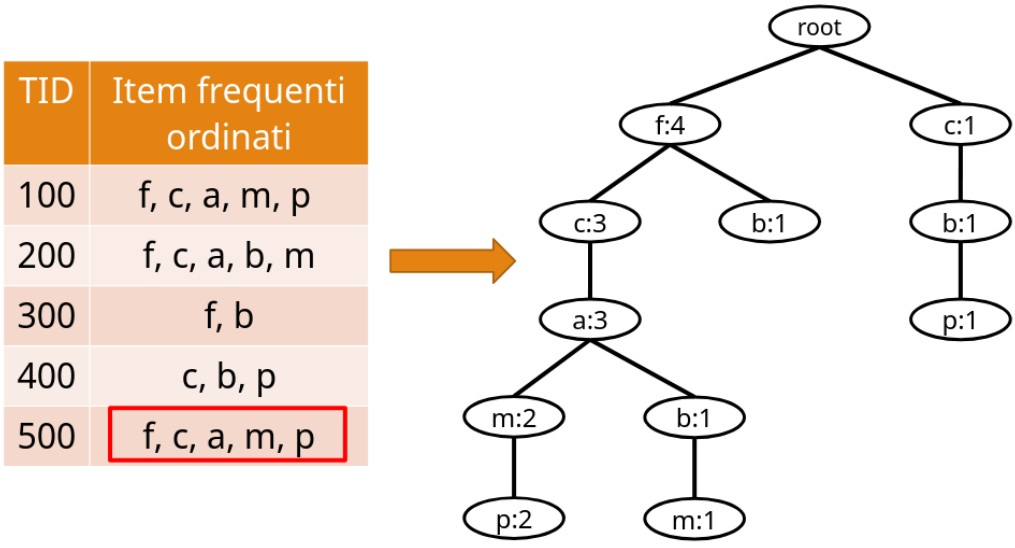

Figura 4.2: Costruzione dell'FP-tree: a sinistra le transazioni con gli item frequenti ordinati; a destra l'albero ottenuto condividendo i prefissi e incrementando i contatori dei nodi. L'ultima transazione (TID 500) segue il percorso f $\rightarrow$ c $\rightarrow$ a $\rightarrow$ m $\rightarrow$ p e aggiorna i relativi nodi.

Visita per pattern-growth. Si processano gli item *dal meno frequente al più frequente* nell'ordine della header table (a parità, dall'ultimo al primo):

$$p \to m \to b \to a \to c \to f.$$

Come si espande un item x (pattern-growth).

1. **Pattern base condizionale di x.** Segui i *node link* di x e, per ogni nodo x, prendi il cammino dalla radice al *genitore* di x (escludi x). Assegna a quel cammino un *peso* uguale al contatore del nodo x.
2. **FP-tree condizionale di x.** Dai cammini pesati: (i) somma i pesi per ogni item e *rimuovi* quelli con supporto $< \sigma$; (ii) ordina gli item per supporto decrescente; (iii) inserisci i cammini (con pesi) costruendo l'albero T_x.
3. **Ricorsione e output.** I pattern frequenti che *contengono* x sono $\{x\}$ unito a ciascun pattern frequente trovato in T_x. *Caso speciale (cammino unico)*: se T_x è una sola path, tutte le combinazioni dei suoi nodi sono frequenti; il supporto è il *minimo* dei contatori lungo la combinazione.

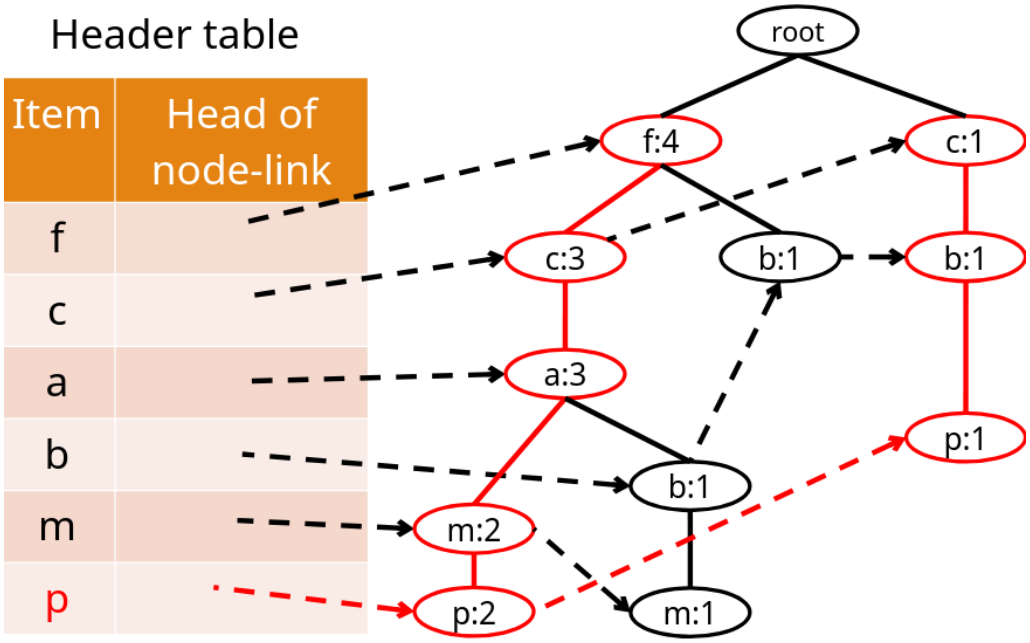

Figura 4.3: Header table e node-link per l'item p: i puntini tratteggiati collegano le occorrenze di p nell'FP-tree. Seguendo i node-link si raccolgono i cammini verso la radice (senza p) con i rispettivi contatori: questa è la pattern base condizionale di p, da cui si costruisce l'FP-tree condizionale T_p.

Esempio 1: item p. Supponiamo che, seguendo i *node link* di p, si incontrino i cammini verso radice:

$$\langle f, c, a, m \rangle : 2 \quad \text{e} \quad \langle c, b \rangle : 1.$$

La **base condizionale** di p è quindi $\{\langle f, c, a, m\rangle$ con peso 2, $\langle c, b\rangle$ con peso $1\}$. Con $\sigma = 3$ nessun sotto-pattern che include p raggiunge la soglia (pesi massimi 2 e 1), dunque *nessun* pattern frequente contiene p.

Esempio 2: item m. Cammini verso m (esempio coerente con le slide):

$$\langle f, c, a\rangle : 2, \quad \langle f, c\rangle : 1.$$

La base condizionale di m è $\{\langle f, c, a\rangle : 2,\ \langle f, c\rangle : 1\}$. Frequenze condizionali:

$$\mathrm{supp}_{\mathrm{cond}}(f) = 3,\ \mathrm{supp}_{\mathrm{cond}}(c) = 3,\ \mathrm{supp}_{\mathrm{cond}}(a) = 2.$$

Con $\sigma = 3$ risultano frequenti i pattern $\{m, f\}$, $\{m, c\}$ e, proseguendo, $\{m, f, c\}$ con supporto 3 (intersezione dei cammini).

Esempio 3: item b. Cammini verso b:

$$\langle f, c, a\rangle : 2, \quad \langle c\rangle : 1.$$

Base condizionale di b: $\{\langle f, c, a\rangle : 2,\ \langle c\rangle : 1\}$. Frequenze condizionali:

$$\mathrm{supp}_{\mathrm{cond}}(c) = 3,\ \mathrm{supp}_{\mathrm{cond}}(f) = 2,\ \mathrm{supp}_{\mathrm{cond}}(a) = 2.$$

Con $\sigma = 3$ si ottiene $\{b, c\}$ frequente; combinazioni con f o a non superano la soglia.

4.9 Confronto: FP-Growth vs Apriori

Aspetto	**Apriori**	**FP-Growth**
Generazione candidati	Sì: crea C_k a ogni livello (rischio di esplosione combinatoria)	No: crescita diretta dei pattern dall'FP-tree
Accessi al DB	Molte passate (una per ogni k)	Tipicamente 2 passate, poi si lavora in memoria
Strutture dati principali	Liste di candidati e conteggi	FP-tree + header table (node-link)
Quando preferirlo	DB piccoli/sparsi, soglie alte, ambienti distribuiti molto semplici	DB densi, soglie basse, molti prefissi condivisi (compressione efficace)
Note pratiche	Pruning con principio di Apriori, implementazione semplice	Evita i candidati; molto veloce se l'FP-tree è compatto

Tabella 4.1: Confronto sintetico tra Apriori e FP-Growth.

Capitolo 5

Clustering

5.1 Concetti generali

Il **clustering** raggruppa oggetti in *cluster* tali che i punti nello stesso cluster siano tra loro simili, mentre punti in cluster diversi siano dissimili. È un compito *unsupervised*: non si conoscono etichette a priori. La *classificazione* è invece *supervised* e richiede classi note per l'addestramento.

5.1.1 Spazi metrici e funzioni distanza

Si assume uno **spazio metrico** $(\mathcal{S}, D)$, con D che soddisfa:

$$\begin{aligned} \textbf{Non negatività} \quad & D(x,y) \geq 0 && \forall\, x, y \in S, \\ \textbf{Simmetria} \quad & D(x,y) = D(y,x) && \forall\, x, y \in S, \\ \textbf{Disuguaglianza triangolare} \quad & D(x,y) + D(y,z) \geq D(x,z) && \forall\, x, y, z \in S. \end{aligned}$$

Distanze in spazi euclidei. Siano $\mathbf{x} = (x_1, \ldots, x_n)$ e $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$.

$$\textbf{Distanza euclidea:} \quad D_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

$$\textbf{Distanza di Manhattan:} \quad D_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

$$\textbf{Norma } L_r: \quad D_r(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^r \right)^{1/r}, \quad r \geq 1$$

$$\textbf{Norma } L_\infty: \quad D_\infty(\mathbf{x}, \mathbf{y}) = \max_{1 \leq i \leq n} |x_i - y_i|$$

$$\textbf{Distanza del coseno (angolare):} \quad D_\angle(\mathbf{x}, \mathbf{y}) = \arccos\left(\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\, \sqrt{\sum_{i=1}^{n} y_i^2}} \right), \qquad \mathbf{x} \neq \mathbf{0},\ \mathbf{y} \neq \mathbf{0}.$$

Spazi non euclidei. Per oggetti-insiemi o stringhe il centroide può non avere senso: si usa il **medoide** (elemento del dataset che minimizza la somma delle distanze agli altri). Esempi di metriche:

$$D_{\text{Jac}}(S,T) = 1 - \frac{|S \cap T|}{|S \cup T|} \quad \text{(Jaccard)},$$

Altri esempi di distanze sono:

1. **Distanza di Edit**: il minimo numero di operazioni di cancellazione o inserzioni di caratteri da effettuare partendo da una stringa A per ottenere la stringa B. (es. A = abcde, B = acfdeg ⇒ D(A, B) = 3).
2. **Distanza di Hamming**: dati A, B vettori, il numero di componeneti in corrispondenza delle quali differiscono (es. A = (1, 0, 1, 0, 1), B = (1, 1, 1, 1, 0) ⇒ D(A, B) = 3).

5.1.2 Tassonomia degli algoritmi

Tre famiglie principali:

1. **gerarchici** (agglomerativi/divisivi);
2. **partizionali** (es. k-means);
3. **a densità** (DBSCAN/OPTICS/HDBSCAN).

Bontà di un algoritmo. Dipende da: *scalabilità*, supporto a attributi eterogenei, capacità di cogliere *forme diverse* di cluster, *robustezza* a outlier/rumore e dati mancanti, *stabilità* all'aggiunta di nuovi dati, e *interpretabilità* dei risultati. La scelta pratica è un compromesso tra qualità e costi computazionali.

5.1.3 Alta dimensionalità: equidistanza e ortogonalità

Spazi euclidei ad elevata dimensionalità soffrono del *problema della dimensionalità*:

- quasi tutte le coppie di punti risultano **equidistanti** e lontane tra loro;
- quasi tutte le coppie di vettori sono quasi **ortogonali**.

Equidistanza dei punti. Sia $D(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2$ con $\mathbf{x},\mathbf{y} \in [0,1]^n$ indipendenti. Quando n è grande, con alta probabilità:

$$\underbrace{1}_{\text{limite inferiore}} \lesssim D(\mathbf{x},\mathbf{y}) \lesssim \underbrace{\sqrt{n}}_{\text{limite superiore}}$$

e *solo una frazione trascurabile* di coppie è vicina ai due limiti. La **maggior parte** delle coppie ha una distanza vicina alla media, circa $\sqrt{n}/3$ (concentrazione della distanza). Inoltre i prodotti scalari tendono a 0, così gli angoli sono prossimi a 90° (quasi ortogonalità).

Conseguenze pratiche. Distinguerre "vicini" da "lontani" diventa difficile; conviene standardizzare le feature, ridurre la dimensionalità (es. PCA) o usare metriche più adatte (cosine/angolare), soprattutto in presenza di dati sparsi.

5.2 Clustering gerarchico

Schema agglomerativo. (a) inizializza: ogni punto è un cluster; (b) ripeti: fonde i due cluster più vicini secondo una *distanza tra cluster*; (c) termina con un criterio (numero desiderato di cluster o qualità).

5.2.1 Distanza tra cluster (*linkage*)

- **Single-link**: $\min\{D(x,y) : x \in C_i,\ y \in C_j\}$ (tende a catene).
- **Complete-link**: $\max\{D(x,y) : x \in C_i,\ y \in C_j\}$ (favorisce cluster compatti).
- **Average-link**: media delle distanze su tutte le coppie $x \in C_i$, $y \in C_j$ (compromesso).
- **Centroid/medoid**: distanza tra centroidi (euclideo) o tra medoidi (generale).

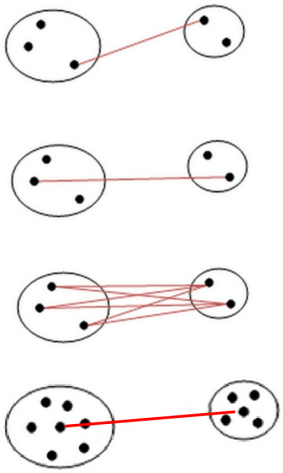

Figura 5.1: Esempi grafici delle diverse nozioni di distanza tra cluster.

5.2.2 Dendrogramma e criteri di stop

Il **dendrogramma** registra le fusioni; tagliandolo a una certa altezza si ottiene la partizione. Criteri di terminazione: (i) fermarsi a k cluster prefissati; (ii) fermarsi quando l'unione successiva degrada troppo la qualità (es. aumento del diametro o della distanza media intra-cluster).

5.2.3 Altri criteri di combinazione

Si può fondere la coppia che massimizza la *qualità* del cluster risultante. Definizioni utili: *raggio* $= \max_{x \in C} D(x, \text{centroide}(C))$; *diametro* $= \max_{x,y \in C} D(x,y)$.

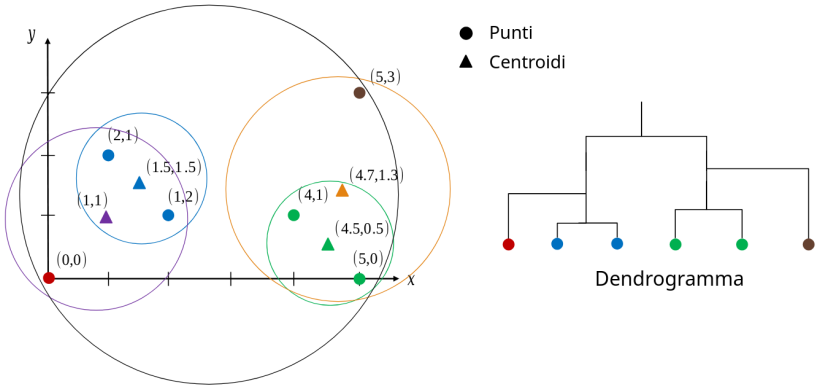

Figura 5.2: A sinistra: punti nel piano con centroidi (triangoli) e cerchi che schematizzano la coesione dei gruppi; i colori indicano i cluster. A destra: dendrogramma agglomerativo che mostra l'ordine di fusione e l'altezza (distanza di linkage). Un taglio orizzontale del dendrogramma determina il numero di cluster.

5.2.4 Versioni divisive

Approccio **top–down**: un'altra versione dove si parte da un unico cluster e lo si *divide* iterativamente scegliendo il miglior *taglio*. Le stesse metriche di distanza/qualità si applicano in modo duale.

5.2.5 Complessità e ottimizzazioni

Analisi *naive*. Al primo passo si valuta la distanza per ogni coppia di cluster e si sceglie la migliore: costo $\Theta(n^2)$. Dopo ogni fusione i cluster diminuiscono di uno, quindi i passi successivi costano, nell'ordine, $(n-1)^2, (n-2)^2, \ldots, 2^2$.

$$T_{\text{naive}} = \sum_{k=2}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} - 1 = \Theta(n^3).$$

(Spazio tipico: matrice delle distanze $O(n^2)$.)

Ottimizzazione con *coda di priorità*. Usando una coda di priorità (min-heap) sulle distanze tra cluster:

- accesso al minimo (*peek*) in $O(1)$; inserimenti e cancellazioni in $O(\log n)$;
- ad ogni fusione si *rimuovono* al più $2(n-1)$ distanze (quelle dai due cluster che si fondono): $O(n \log n)$;
- si *calcolano e inseriscono* le distanze tra il nuovo cluster e gli altri (al più $n-2$): $O(n \log n)$.

Su $n-1$ fusioni:

$$T_{\text{heap}} = O(n \cdot (n \log n)) = O(n^2 \log n).$$

Risultato: la complessità scende da $O(n^3)$ a circa $O(n^2 \log n)$ mantenendo la matrice (o la coda) aggiornata a ogni iterazione.

5.3 Clustering partizionale: k-means

Metodi per spazi euclidei che partizionano i dati in k cluster minimizzando la somma delle distanze al quadrato dai centroidi.

5.3.1 Algoritmo base

1. **Inizializza** k centroidi (idealmente separati).
2. **Assegna** ogni punto al centroide più vicino (distanza euclidea).
3. **Aggiorna** ogni centroide come media dei punti assegnati.
4. **Ripeti** finché i centroidi si stabilizzano o il miglioramento è sotto soglia.

Converge in pochi round, ma solo a un ottimo *locale*.

5.3.2 Inizializzazione

Scelta *greedy*:

1. Si sceglie il primo punto in maniera casuale o lo si aggiunge all'insieme S dei punti già selezionati, inizialmente vuoto.
2. Si calcola la massima distanza minima dai centroidi scelti.
3. Si ripete il passo 2. finché $|s| < k$.

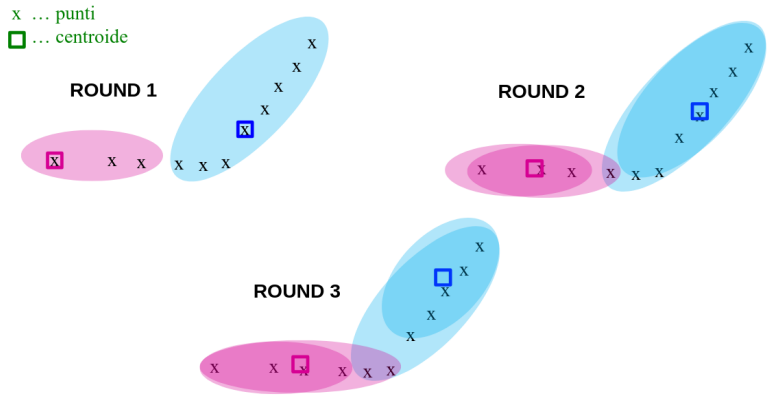

Figura 5.3: K-means: evoluzione in tre round. Round 1: inizializzazione e prime assegnazioni ai centroidi (quadrati). Round 2: ricalcolo dei centroidi e riassegnazione dei punti (X). Round 3: i centroidi si stabilizzano e i cluster (ellissi colorate) convergono.

5.3.3 Funzione obiettivo e arresto

Con partizione $C_1, \ldots, C_k$ e centroidi μ_r:

$$J = \sum_{r=1}^{k} \sum_{x \in C_r} \|\mathbf{x} - \mu_r\|_2^2.$$

Arresto quando ΔJ tra iterazioni consecutive è sotto soglia o quando non cambia l'assegnazione. Con metriche diverse da euclidea il centroide non è il minimizzatore naturale.

5.3.4 Scelta del numero di cluster k

Poiché k non è noto a priori, si esegue il metodo per più valori e si seleziona quello che ottimizza una metrica di qualità interna.

Funzione obiettivo. Per k cluster $C_1, \dots, C_k$ con centroidi $\mathbf{c}_1, \dots, \mathbf{c}_k$, la funzione standard è

$$W(k) = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{c}_i\|_2^2, \qquad \bar{W}(k) = \frac{W(k)}{n} \text{ (distanza media al centroide).}$$

$W(k)$ è decrescente in k.

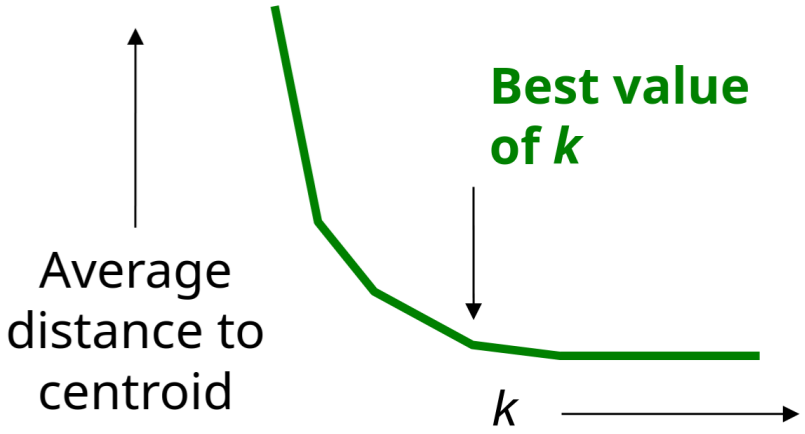

Figura 5.4: Metodo (*elbow*). Si traccia la distanza media dal centroide (o WCSS/n) al variare di k; il valore "ottimo" è nel punto di flesso, dove l'aumento di k porta benefici marginali trascurabili.

Metodo *elbow*. Si calcola $\bar{W}(k)$ per $k = k_{\min}, \dots, k_{\max}$ e si sceglie il k per cui il calo di $\bar{W}$ passa da "ripido" a "lento" (punto di flesso).

- *Procedura pratica:* si valuta $\bar{W}(k)$ su una griglia di valori e si ispeziona il grafico $\bar{W}$ vs k.
- *Variante a ricerca binaria:* fissati due estremi $x < y$, si prende $z = \lfloor (x+y)/2 \rfloor$, si calcola $\bar{W}(z)$ e si sostituisce l'estremo *più vicino* a $\bar{W}(z)$ con z; si ripete finché l'intervallo è piccolo. Il k finale approssima il gomito.

Nota: se la curva non mostra un gomito netto, l'elbow diventa ambiguo e conviene affiancarlo a silhouette/stabilità.

Metodo *silhouette.* Per ogni punto $\mathbf{x}$ assegnato al cluster C_i:

$$a(\mathbf{x}) = \frac{1}{|C_i| - 1} \sum_{\mathbf{y} \in C_i,\ \mathbf{y} \neq \mathbf{x}} \|\mathbf{x} - \mathbf{y}\|, \qquad b(\mathbf{x}) = \min_{j \neq i} \ \frac{1}{|C_j|} \sum_{\mathbf{y} \in C_j} \|\mathbf{x} - \mathbf{y}\|.$$

Lo *score di silhouette* del punto è

$$s(\mathbf{x}) = \frac{b(\mathbf{x}) - a(\mathbf{x})}{\max\{a(\mathbf{x}),\, b(\mathbf{x})\}} \in [-1, 1].$$

Valori vicini a 1 indicano assegnazioni "pulite", vicini a 0 punti al confine, negativi assegnazioni sbagliate. Si sceglie

$$k^\star = \arg\max_k \ \frac{1}{n} \sum_{r=1}^{n} s(\mathbf{x}_r).$$

Regole d'uso. Calcolare gli indici su più esecuzioni (inizializzazioni diverse) e riportare media/deviazione; standardizzare le feature prima del confronto; evitare k troppo grandi che trivialiscono $\bar{W}$ ma peggiorano la silhouette.

5.3.5 Complessità computazionale

Ogni iterazione del k-means ha due passi:

1. **Assegnamento** (nearest–centroid): per ciascun punto si valuta la distanza verso i k centroidi. Costo $O(nkd)$ in $\mathbb{R}^d$ (spesso si sottintende d, scrivendo $O(nk)$).
2. **Aggiornamento dei centroidi**: si ricalcolano le medie dei cluster. Costo $O(nd)$.

Con t iterazioni totali:

$$T(n, k, d, t) = O(t\, n\, k\, d) \quad \text{(nelle slide: } O(tkn)\text{)}.$$

5.4 Clustering per densità

5.4.1 DBSCAN

DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) definisce i cluster come regioni con densità elevata separate da regioni a bassa densità. Richiede due parametri:

$$\varepsilon > 0 \quad \text{(raggio dell'intorno)} \qquad \text{e} \qquad \texttt{MinPts} \in \mathbb{N} \quad \text{(soglia di densità)}.$$

Definizioni. Dato un punto p e una metrica $D(\cdot, \cdot)$:

- **ε-intorno**: $N_\varepsilon(p) = \{x : D(x, p) \leq \varepsilon\}$.
- **Core point**: p è *core* se $|N_\varepsilon(p)| \geq \texttt{MinPts}$.
- **Directly density-reachable**: q è *direttamente raggiungibile per densità* da p se $q \in N_\varepsilon(p)$ e p è core.

- **Density-reachable**: q è *raggiungibile per densità* da p se esiste una catena $p = x_0, x_1, \ldots, x_m = q$ in cui ogni x_{i+1} è direttamente raggiungibile da x_i.
- **Density-connected**: p e q sono *connessi per densità* se esiste o tale che p e q sono entrambi raggiungibili per densità da o.

Cluster C = insieme massimale di punti *density-connected*; i punti non assegnati sono *rumore* (outlier). Punti non-core inclusi in un cluster sono detti *border*.

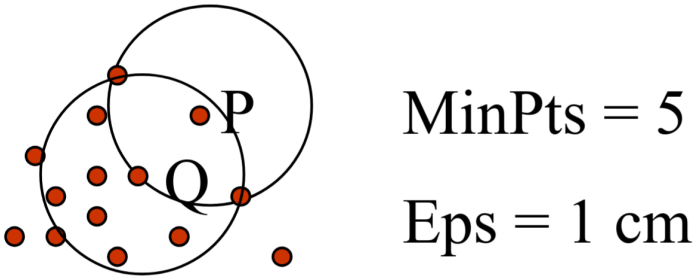

Figura 5.5: DBSCAN: esempi di *core*, *border* e *noise* con parametri ε e `MinPts`.

Algoritmo.

1. Visita un punto non ancora etichettato p e calcola $N_\varepsilon(p)$.
2. Se p è *core*, crea un nuovo cluster e *espandilo*: aggiungi ricorsivamente tutti i punti direttamente raggiungibili, iterando finché possibile (tutti i raggiungibili per densità da p).
3. Se p non è core e non è stato assegnato come *border*, etichettalo come *rumore*.
4. Ripeti finché tutti i punti sono visitati.

Scelta dei parametri. Usare `MinPts` $\approx d{+}1$ come minimo (con d dimensioni) e valori più alti con dataset grandi o rumorosi. Per ε si usa il *k-distance plot*: per ogni punto si considera la distanza dal k-esimo vicino ($k =$ `MinPts`), si ordina in senso decrescente e si cerca il *gomito* della curva.

Complessità. Il costo è dominato dalle ricerche di vicinato. Con appropriate strutture (R-tree) il costo è $O(n \log n)$. Senza questo tipo di strutture, il costo è $O(n^2)$.

Pro e contro. *Pro*: non richiede k, trova cluster di forma arbitraria, gestisce il rumore, poco sensibile all'ordine di scansione. *Contro*: scelta di (ε, `MinPts`) non banale; difficile con densità molto diverse o in alta dimensione.

5.4.2 OPTICS

OPTICS (*Ordering Points To Identify the Clustering Structure*) estende DBSCAN per gestire **densità variabili**. Invece di una singola partizione, produce un *ordinamento* dei

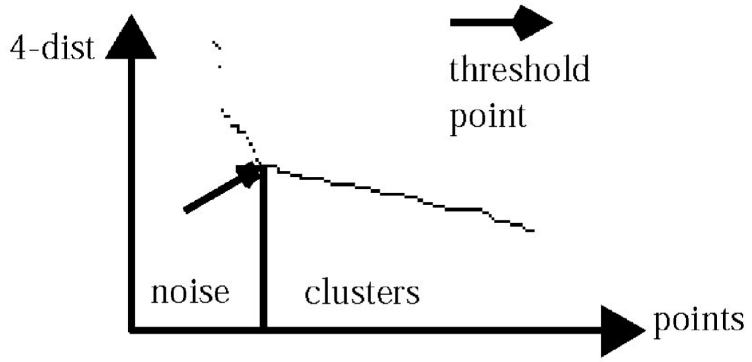

Figura 5.6: k-distance plot: il gomito suggerisce il valore di ε.

punti con associata una misura di "raggiungibilità" che riassume la struttura di densità a più scale.

Core–distance e reachability (OPTICS). Fissiamo `MinPts` $= k$ e una metrica D.

***Core–distance* di p:** è il raggio minimo che rende p un punto *core*. In pratica è la distanza dal k-esimo vicino di p:

$$\text{core\_dist}_k(p) = d_k(p).$$

Se p ha meno di k vicini, non è core e si pone $\text{core\_dist}_k(p) = +\infty$ (non definita).

***Reachability–distance* di o da p:** misura quanto è "raggiungibile" o partendo da p:

$$\text{reach\_dist}_k(o \,|\, p) = \max\{\text{core\_dist}_k(p),\, D(p, o)\}.$$

Lettura immediata.

- Se p è in una regione densa ($\text{core\_dist}_k(p)$ piccola) e o è *dentro* quel raggio, allora $\text{reach\_dist}_k(o \,|\, p) = \text{core\_dist}_k(p)$ (tutti questi o "valgon lo stesso" nel plot).
- Se o è *più lontano* del raggio denso di p, $\text{reach\_dist}_k(o \,|\, p) = D(p, o)$ (serve "uscire" dalla zona densa).

Mini-esempio. Con $k = 5$ e $\text{core\_dist}_5(p) = 0.8$: un vicino a distanza 0.6 ha reach_dist $=$ 0.8; un punto a distanza 1.2 ha reach_dist $= 1.2$.

Risultato: ordering e *reachability plot*. OPTICS visita iterativamente i punti scegliendo, tramite una coda di priorità, quello con *reach_dist* minima; registra per ciascun punto l'ordine di visita e la sua *reachability*. Plottando le *reachability* nell'ordine si ottiene il **reachability plot**: le *valli* corrispondono a cluster, le creste a separazioni.

Estrazione dei cluster. Si possono ottenere partizioni:

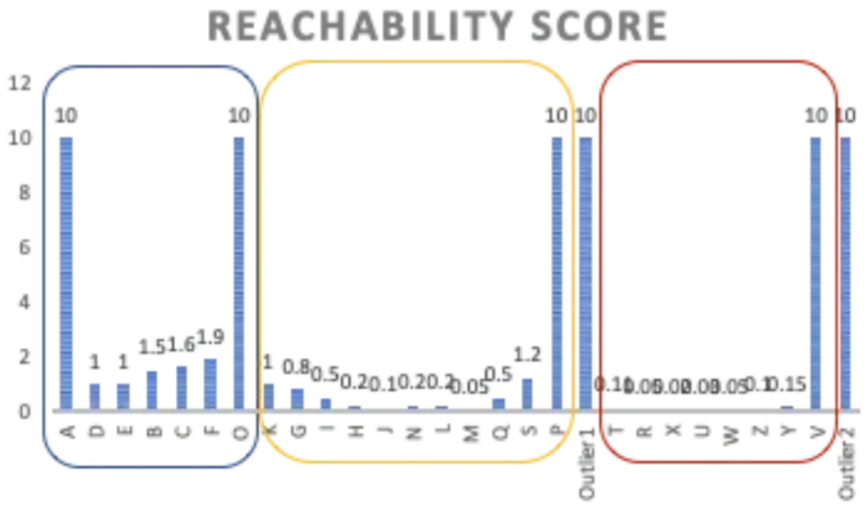

Figura 5.7: OPTICS: esempio di *reachability plot*. Le zone basse (valli) indicano cluster densi; le zone alte (creste) indicano separazioni. (immagine da libro/slide)

- applicando un *cut* orizzontale sul plot (equivalente a DBSCAN a un dato ε);
- individuando automaticamente valli significative (metodi di *valley picking* o soglie relative).

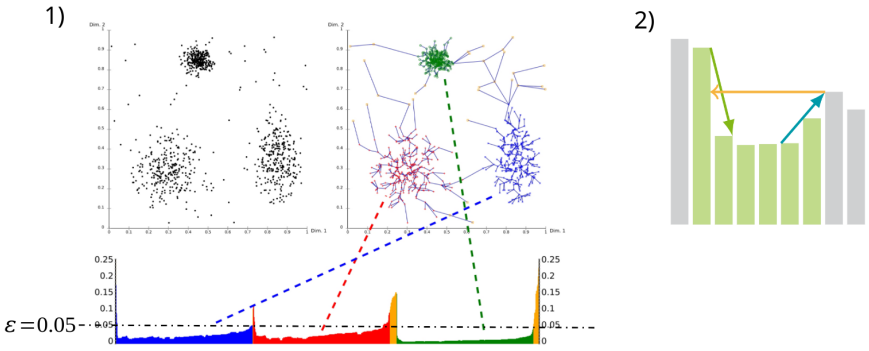

Figura 5.8: OPTICS. (1) A sinistra: punti nel piano e, sotto, *reachability plot*; le valli (segmenti colorati) corrispondono a regioni dense/cluster, mentre la linea orizzontale tratteggiata indica una soglia ϵ che produce un taglio in stile DBSCAN. Le linee tratteggiate collegano ogni gruppo nel piano al suo intervallo nel plot. (2) A destra: regola *steep down/up* per l'estrazione automatica dei cluster dal reachability plot (si entra quando la reachability scende bruscamente e si esce quando risale).

Costo. Con le adeguate strutture: $O(n \log n)$; altrimenti $O(n^2)$.

5.4.3 HDBSCAN

HDBSCAN (Hierarchical DBSCAN) estende DBSCAN costruendo una *gerarchia di cluster per densità* e selezionando automaticamente i cluster più *stabili*. L'unico parametro concettuale è `MinPts` (soglia minima di densità); nelle librerie può comparire anche `min_cluster_size` (qui lo assimiliamo a `MinPts` come nelle slide).

Idea.

1. Definisco una distanza che "appiattisce" le regioni rade e rende confrontabili densità diverse.
2. Costruisco l'MST su tale distanza e, facendo crescere la densità minima $\lambda = 1/\varepsilon$, ottengo un *cluster tree*.
3. Condenso l'albero tenendo solo rami sostenuti da almeno `MinPts` punti e scelgo i cluster più *stabili*.

Core distance di X. core_dist(p) = distanza dal `MinPts` più vicino.

Distanza di *mutual reachability*. Sia core_dist(p) la distanza dal `MinPts` *punto* più vicino di p. La distanza

$$d_{\text{mreach}}(p, q) = \max\{\text{core\_dist}(p),\ \text{core\_dist}(q),\ D(p, q)\}$$

"*dilata*" le regioni poco dense (aumentando le loro distanze interne) e *comprimi* quelle dense: in questo modo cluster a densità diverse risultano separabili con un unico parametro.

Mutual Reachability graph G_{MinPts}. grafo pesato in cui i nodi sono gli oggetti dello spazio e, per ogni coppia di oggetti X, Y, viene inserito un arco il cui peso coincide con la Mutual Reachability distance tra X e Y.

Costruzione del dendogramma

Per la costruzione del dendogramma si procede così:

Costruzione del MST Si costruisce il Minimum Spanning Tree (MST) di G_{MinPts}, ovvero il sottografo aciclico di di G il cui peso totale sugli archi è minimo e che garantisce la connessione di G.

Rimozione degli archi Si ordinano gli archi del MST in ordine decrescente di peso e si rimuovono progressivamente, partendo da quello più pesante. Ad ogni rimozione si ottiene una partizione dei punti in cluster (componenti connesse del grafo rimanente). Questa struttura, la chiamiamo *cluster tree*.

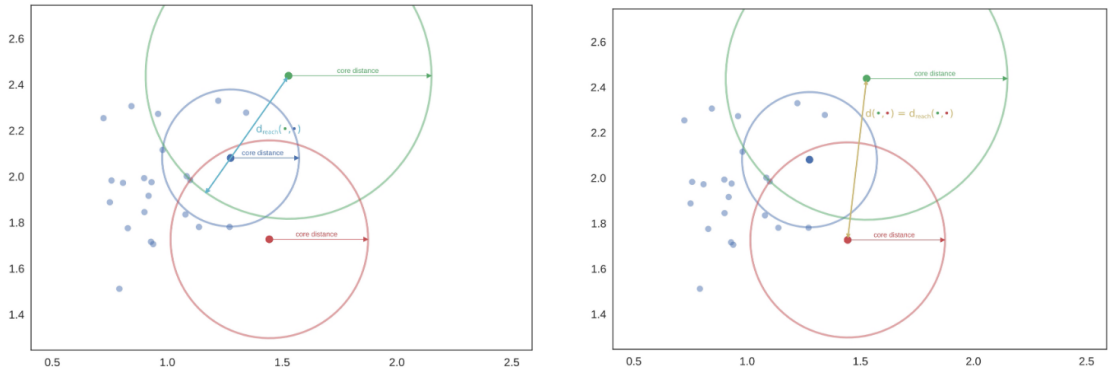

Figura 5.9: HDBSCAN: illustrazione di *core_dist*, *reach_dist* e *mutual reachability*. Ogni cerchio centrato in un punto ha raggio pari alla sua core_dist$(\cdot)$ (distanza dal `MinPts`-esimo vicino). **Sinistra**: per i punti p (verde) e o (blu) la distanza di raggiungibilità da p a o è reach_dist$(o \mid p) = \max\{\text{core_dist}(p), D(p,o)\}$; poiché il raggio di p domina, vale reach_dist$(o \mid p) = $ core_dist(p). **Destra**: tra p (verde) e q (rosso) prevale la distanza euclidea, quindi la *mutual reachability* risulta $d_{\text{mreach}}(p,q) = \max\{\text{core\_dist}(p), \text{core\_dist}(q), D(p,q)\} = D(p,q)$. L'uso di d_{mreach} "gonfia" le regioni rade e rende più separabili cluster con densità diverse.

Condensed tree e selezione dei cluster significativi

Condensed tree. A partire dal cluster tree si *condensa* mantenendo solo i rami che, per qualche intervallo di densità λ, hanno almeno `MinPts` punti (gli altri rami vengono collassati sul padre). Ogni nodo C del condensed tree porta:

$$\begin{aligned} \lambda_{\min}(C) &\quad \text{(densità alla nascita)} \\ \lambda_{\max}(C) &\quad \text{(densità alla scomparsa/split)} \\ |C|(\lambda) &\quad \text{(numero di punti in } C \text{ alla densità } \lambda) \end{aligned}$$

Qui $|C|(\lambda)$ indica quanti punti appartengono al cluster C per una data densità λ; questo valore viene usato per decidere se mantenere o collassare un ramo nel *condensed tree*.

Stabilità (persistenza). La stabilità misura "quanto a lungo" un cluster esiste al crescere di λ. Per un nodo C del condensed tree:

$$\text{stab}(C) = \sum_{p \in C} (\lambda_p^{\max}(C) - \lambda^{\min}(C)),$$

dove $\lambda_p^{\max}(C)$ è la densità alla quale il punto p lascia C (per split o perché C muore). Equivalente: è l'"area sotto la curva" del numero di punti di C lungo λ nel condensed tree.

Estrazione dei cluster significativi (dal *condensed tree*). *Input:* albero condensato in cui ogni nodo C porta $\lambda_{\text{birth}}(C)$, $\lambda_{\text{death}}(C)$ e la funzione $|C|(\lambda)$ (# punti vivi in C alla densità λ).

1. **Calcolo della stabilità (bottom–up).** Per ogni nodo C si valuta la *persistenza* del cluster lungo l'intervallo in cui esiste come:

$$\mathrm{stab}(C) \;=\; \sum_{p \in C} \left(\lambda_p^{\max}(C) - \lambda^{\min}(C)\right),$$

dove $\lambda^{\min}(C)$ è la densità alla *nascita* di C e $\lambda_p^{\max}(C)$ è la densità alla quale il punto p *esce* da C (per split o scomparsa). Si procede dal basso verso l'alto in modo da avere già disponibili le stabilità dei figli quando si valuta il padre.

2. **Regola di selezione (top–down, senza sovrapposizioni).** Visitando un nodo C con figli $C_1, \ldots, C_m$, si confronta la stabilità del padre con la somma di quelle dei figli:

$$S_{\text{figli}} = \sum_{i=1}^{m} \mathrm{stab}(C_i).$$

   - Se $\mathrm{stab}(C) \geq S_{\text{figli}}$: **seleziona** C come cluster e *non* scendere oltre in quel ramo.
   - Altrimenti: **non** selezionare C e **scendere** ricorsivamente, applicando la stessa regola ai figli e selezionando quelli con stabilità > 0.

3. **Output.** I nodi selezionati sono disgiunti e costituiscono i *cluster significativi* massimizzando la stabilità complessiva; in caso di parità si preferisce il padre.

Assegnazione dei border (opzionale). I punti non-core vicini a più cluster possono essere assegnati al cluster selezionato con λ_{birth} più basso o tramite un punteggio di *membership* proporzionale al tempo di permanenza del punto nel cluster nel condensed tree.

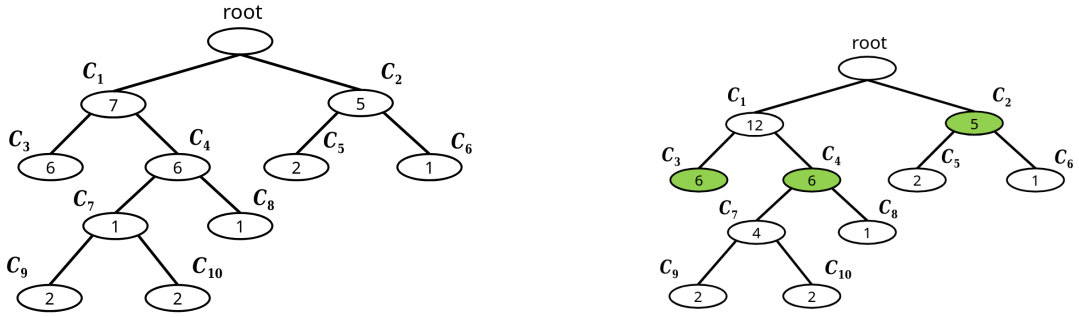

Figura 5.10: HDBSCAN. Sinistra: MST costruito sulla *mutual reachability distance* (archi più pesanti vengono tagliati al crescere di λ). Destra: *condensed tree*; in evidenza i cluster scelti massimizzando la stabilità.

Costo. Dominato dalla costruzione del MST: con strutture adeguate $O(n \log n)$; altrimenti $O(n^2)$.

Capitolo 6

Classificazione

6.1 Introduzione

La **classificazione** suddivide un insieme di dati in *classi* note a priori (etichette), apprendendo da esempi etichettati come assegnare la classe a nuove tuple. È quindi *apprendimento supervisionato*. Al contrario, il *clustering* non parte da etichette (*unsupervised*) e scopre gruppi per similarità.

Predizione (regressione). Quando il target è *numerico continuo*, il compito è di *predire* un valore reale (apprendimento supervisionato *continuo*), cercando una funzione che approssimi il target, non un confine tra classi.

6.1.1 Schema generale di un classificatore

1. **Costruzione del modello** (training): si apprende da un *training set* etichettato.
2. **Validazione/valutazione** (test): si misura la bontà su un *test set* etichettato.
3. **Uso** (deploy): si applica il modello a nuove tuple per predirne la classe.

Overfitting. L'overfitting si verifica quando un modello "impara a memoria" il training, compreso il rumore: va molto bene sui dati visti ma generalizza male su dati nuovi. In pratica è un segnale che il modello è troppo complesso rispetto alle informazioni disponibili. Per ridurlo, si separano chiaramente i dati per la verifica e si preferiscono soluzioni più semplici quando offrono prestazioni simili.

6.1.2 Requisiti desiderabili

- **Accuratezza**: corretta predizione delle classi (o del valore, per i predittori).
- **Velocità**: tempi di training e di classificazione contenuti.
- **Robustezza**: tolleranza a rumore e dati mancanti.
- **Scalabilità**: efficienza su dataset di grandi dimensioni.

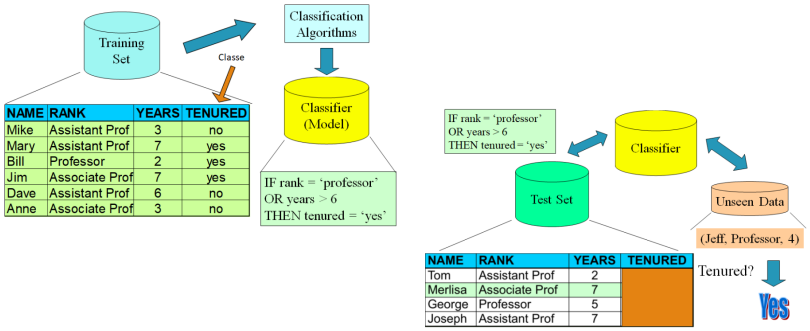

Figura 6.1: Schema a blocchi di un classificatore: addestramento, validazione e uso.

6.2 Alberi decisionali

Gli **alberi decisionali** classificano applicando test su attributi lungo i nodi interni; le *foglie* portano le etichette di classe.

6.2.1 Classificazione tramite albero

La classe di una tupla q si ottiene seguendo il cammino radice→foglia guidato dai test. Ogni cammino implementa una regola `IF-THEN` (le condizioni interne sono congiunte in AND). L'insieme di regole è *esaustivo* e *mutuamente esclusivo* (ogni tupla è coperta da una sola regola).

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

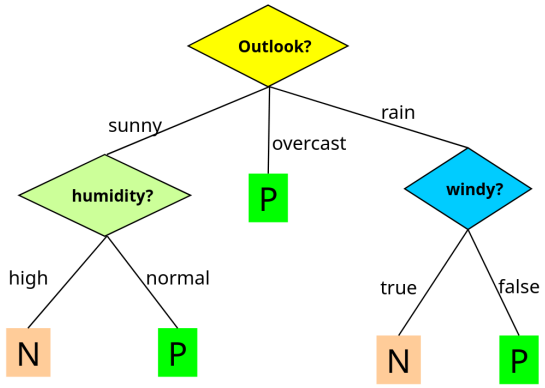

Figura 6.2: Dataset *weather* (a sinistra) e albero decisionale appreso (a destra). La tabella contiene 14 esempi con quattro attributi descrittivi (`Outlook`, `Temperature`, `Humidity`, `Windy`) e la classe binaria `P/N`. L'albero (stile ID3/C4.5) sceglie come radice `Outlook`; il ramo `overcast` porta direttamente alla classe `P`, mentre per `sunny` si testa `Humidity` e per `rain` si testa `Windy`. L'esempio illustra il passaggio da dati tabellari a regole interpretabili.

6.2.2 Costruzione top–down

Costruzione ricorsiva dalla radice:

1. Se tutte le tuple del nodo X hanno la *stessa* classe C, crea una foglia C.
2. Altrimenti scegli un attributo A (non ancora usato) e *ramifica* X (*splitting*) secondo i valori/soglia di A; crea i figli.
3. Per ogni figlio X_i: se puro, fermati; se impuro, ripeti ricorsivamente.

Pruning. Se le tuple nel nodo sono poche o la profondità è elevata, si può fermare prima e rendere il nodo una foglia (vedere figura 6.2 con l'attributo "overcast").

6.2.3 Splitting degli attributi

- **Booleani/numerici**: split *binario* su soglia t ("$\leq t$" a sinistra, "$> t$" a destra).
- **Categoriali**: split *binario* definendo un sottoinsieme non vuoto di valori (a sinistra se il valore *non* appartiene al sottoinsieme, a destra altrimenti).

6.2.4 Scelta dell'attributo e strategia greedy

L'albero minimale è un problema *NP-hard*; si usa una strategia *greedy* che, ad ogni passo, seleziona l'attributo con massima *goodness* (partizioni più pure), costruendo l'albero "più compatto" possibile.

6.3 Misure di goodness

La scelta dell'attributo si basa su misure di *goodness*, che variano da algoritmo ad algoritmo.

6.3.1 Information Gain (ID3)

Idea. L'**Information gain** è un algoritmo che si basa sull'idea di selezionare l'attributo che massimizza la riduzione dell'entropia riguardo alla classe delle tuple dopo lo split. Questo perché:

- **Entropia massima**: si ha quando le classi sono equamente distribuite (massima incertezza).
- **Entropia minima**: si ha quando tutte le tuple appartengono alla stessa classe (certezza completa).

Sia S_X l'insieme di tuple al nodo X, con due classi P e N; si indichino con p e n le rispettive numerosità. L'**entropia** di S_X è

$$H(S_X) \;=\; -\frac{p}{p+n}\log_2\frac{p}{p+n} \;-\; \frac{n}{p+n}\log_2\frac{n}{p+n}.$$

Sia A un attributo con k valori distinti, che induce la partizione $S_X \to S_1, \ldots, S_k$. Se S_i contiene p_i e n_i elementi, allora

$$H(S_i) \;=\; -\frac{p_i}{|S_i|}\log_2\frac{p_i}{|S_i|} \;-\; \frac{n_i}{|S_i|}\log_2\frac{n_i}{|S_i|}$$

Possiamo anche calcolare l'**entropia media** dopo lo split su A:

$$\overline{H}_A(S_X) \;=\; \sum_{i=1}^{k} \frac{|S_i|}{|S_X|}\, H(S_i).$$

L'**information gain** è definito come la riduzione di entropia ottenuta dal partizionamento S_x scegliendo l'attributo A:

$$\mathrm{Gain}(S_X, A) \;=\; H(S_X) \;-\; \overline{H}_A(S_X).$$

Si sceglie l'attributo con gain massimo.

Esempio e limitazioni

Sul dataset "weather" (Fig. 6.2) si ottengono:

$$\mathrm{Gain}(\textit{outlook}) = 0.246, \quad \mathrm{Gain}(\textit{temperature}) = 0.029, \quad \mathrm{Gain}(\textit{humidity}) = 0.151, \quad \mathrm{Gain}(\textit{windy}) = 0.048.$$

Limite noto. L'Information Gain è *sbilanciato* verso attributi con molti valori: un attributo quasi univoco (es. `ID`) produce molte partizioni piccole (foglie pure), abbattendo l'entropia media e gonfiando artificialmente , pur senza reale capacità predittiva.

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

L'attributo *outlook* è scelto come radice:

gain(outlook) = 0.246

gain(temperature) = 0.029

gain(humidity) = 0.151

gain(windy) = 0.048

Figura 6.3: Esempio di scelta della radice con Information Gain sul dataset "weather".

6.3.2 Gain Ratio (C4.5)

Il **Gain Ratio** corregge il bias dell'Information Gain verso attributi con molti valori introducendo la *split information*, che misura quanta informazione è generata dal solo atto di partizionare i dati secondo l'attributo (indipendentemente dalla classe).

Sia S l'insieme di tuple nel nodo corrente e A un attributo che induce la partizione $S = S_1 \cup \cdots \cup S_k$. Definiamo

$$\mathrm{SplitInfo}(A, S) = -\sum_{i=1}^{k} \frac{|S_i|}{|S|}\, \log_2\!\Big(\frac{|S_i|}{|S|}\Big), \qquad \mathrm{Gain}(A, S) = H(S) - \sum_{i=1}^{k} \frac{|S_i|}{|S|}\, H(S_i).$$

Il **Gain Ratio** è

$$\mathrm{GR}(A,S) = \frac{\mathrm{Gain}(A,S)}{\mathrm{SplitInfo}(A,S)}.$$

Selezione in C4.5. Per evitare divisioni spurie quando SplitInfo è piccola, C4.5 sceglie l'attributo con GR massimo tra quelli con *Gain* non inferiore (ad es.) al gain medio del nodo. In pratica:

1. calcola $\mathrm{Gain}(A,S)$ e scarta attributi con gain ≤ 0;
2. tra i rimanenti, seleziona l'attributo con GR più alto.

Nota pratica (attributi continui). Per un attributo numerico A si ordinano i valori e si valutano soglie candidate t nelle posizioni fra due valori consecutivi: $A \leq t$ vs $A > t$. Per ogni soglia si calcolano gain e gain ratio; si sceglie la soglia che massimizza la metrica.

6.3.3 Gini Index (CART)

Sia i una classe e T una tupla di classe i scelta a caso da S_x. Per ricavare il **Gini Index** si calcola la probabilità che T venga classificata erroneamente, ovvero che appartenga a una classe diversa da i: e quindi occorre considerare:

- **Probabilità che T sia di classe i**: $P(i \mid S_X)$.
- **Probabilità che T sia di una classe diversa da i**: $1 - P(i \mid S_X)$.

Dato che il ragionamento fatto vale per ogni classe, si sommano le probabilità di errore su tutte le classi:

$$\mathrm{Gini}(S_X) = \sum_{i=1}^{n} p_i(1-p_i) = \sum_{i=1}^{n}(p_i - p_i^2) = \sum_{i=1}^{n} p_i - \sum_{i=1}^{n} p_i^2 = 1 - \sum_{i=1}^{n} p_i^2$$

Con la supposizione che S_x contenga k classi e che p_i sia la probabilità che una tupla scelta a caso da S_X appartenga alla classe i, si ha che il **Gini Index** dello split è defininito come:

$$\mathrm{Gini}_{\mathrm{split}}(S_X) = 1 - \sum_{i=1}^{k} \frac{|S_i|}{|S_x|} Gini(S_i)$$

CART seleziona l'attributo/soglia che *minimizza* GiniSplit. Su attributi categoriali si cercano partizioni in due sottoinsiemi di valori; su continui, soglie come in C4.5.

6.3.4 Pruning degli alberi

Alberi molto profondi generalizzano male (generano *overfitting*). Per evitare questo, si effettua un **pruning**, ovvero si riduce la dimensione dell'albero sostituendo un sottoalbero con una foglia etichettato con la classe maggioritaria delle tuple nel sottoalbero, il pruning inserisce però un tasso di errore, si fa solo se necessario. Si usano due strategie principali:

Pre-pruning in fase di costruzione dell'albero si interrompe la crescita quando la goodness dello split è al di sopra di una *soglia*.

Post-pruning si costruisce l'albero completo e poi lo si riduce valutando l'errore su validation set o tramite stima incrociata. Generalmente è più dispensioso ma più efficace.

Pruning pessimistico (C4.5)

Confronta l'errore stimato del *sottoalbero* T radicato in X con l'errore stimato della *foglia* che sostituisce T (classe maggioritaria in X).

Sia X un nodo dell'albero con insieme di esempi S_x ($N = |S_x|$) e classe di maggioranza C. Sia T il sottoalbero radicato in X e siano $x_1, \ldots, x_k$ i figli immediati di X, con S_{x_i} gli esempi nel figlio x_i e C_i la sua classe di maggioranza. Le due quantità

$$E_p(T) = \frac{\big|\{\, t \in S_x \mid \text{class}(t) \neq C \,\}\big| + \epsilon}{|S_x|}$$

$$E'_p(T) = \frac{\displaystyle\sum_{i=1}^{k} \big|\{\, t \in S_{x_i} \mid \text{class}(t) \neq C_i \,\}\big| + k\,\epsilon}{|S_x|}$$

sono le **stime del tasso di errore** usate per decidere se fare pruning.

Che cosa misurano.

- $E_p(T)$ è l'*errore stimato* se **si pota** T sostituendo l'intero sottoalbero con *una sola foglia* etichettata con la classe di maggioranza C del nodo X. Il numeratore conta le istanze di S_x che verrebbero sbagliate da tale foglia, con una *correzione* ϵ (tipicamente $\epsilon = \frac{1}{2}$) per evitare stime troppo ottimistiche su campioni piccoli.
- $E'_p(T)$ è l'*errore stimato* se **si mantiene lo split corrente** di X nei suoi k figli, ma *troncando* ognuno di essi a foglia (ognuna etichettata con la propria maggioranza C_i). Si sommano gli errori dei k figli e si aggiunge una correzione ϵ per *ciascuna* foglia ($k\epsilon$).

Decisione di pruning. Confrontando $E_p(T)$ ed $E'_p(T)$:

- per $E_p(T) \leq E'_p(T)$, *potare* il nodo X è preferibile, poiché l'errore stimato come foglia è minore o uguale a quello del sottoalbero.
- per $E_p(T) > E'_p(T)$, conviene *mantenere* lo split, in quanto l'errore stimato del sottoalbero è inferiore a quello della singola foglia.

Il valore ϵ è una sorta di "costo fisso" per ogni foglia aggiunta all'albero. L'aggiunta di questo valore agisce da *regolarizzatore*: penalizza strutture con molte foglie, evitando che piccole fluttuazioni del campione giustifichino split inutili.

Cost–complexity pruning (CART)

Si valuta il vantaggio dello *split* di un nodo X confrontando la riduzione di *error rate* con l'aumento di complessità (nuove foglie).

Errore prima e dopo lo split. Sia S_X l'insieme dei campioni che arrivano al nodo X e C la classe maggioritaria in S_X.

$$E(X) = \frac{|\{\, t \in S_X : \ \text{class}(t) \neq C \,\}|}{|S_X|}.$$

Se X viene diviso in k figli $X_1, \ldots, X_k$ (con classi maggioritarie $C_1, \ldots, C_k$), l'errore *atteso dopo* lo split è

$$E'(X) = \frac{\sum_{i=1}^{k} |\{\, t \in S_{X_i} : \ \text{class}(t) \neq C_i \,\}|}{|S_X|}.$$

Indice di costo–complessità per lo split. Definiamo il guadagno medio per foglia aggiunta:

$$\alpha(X) = \frac{E(X) - E'(X)}{k - 1}.$$

Il valore α misura *quanto* diminuisce l'errore per ogni foglia extra introdotta dallo split. Se α è sufficientemente piccolo, ovvero quando α è minore di una soglia prefissata α_0, lo split non è conveniente e si pota il nodo X.

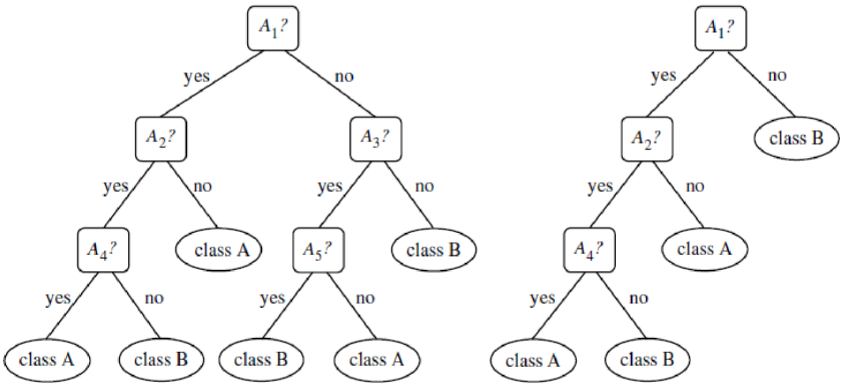

Figura 6.4: Pruning: confronto errore stimato del sottoalbero vs foglia (C4.5) e principio costo–complessità (CART).

Pro/contro degli alberi decisionali. *Pro:* interpretabili, veloci in predizione, gestiscono mix di attributi (continui/categoriali), poca preparazione dei dati. *Contro:* instabili rispetto a piccole variazioni dei dati, propensi all'overfitting, separazioni per soglie assiali (forme complesse richiedono molti nodi), accuratezza spesso inferiore a ensemble o SVM su dati ad alta dimensionalità.

6.4 Classificatori generativi

I modelli generativi producono un **modello probabilistico** a partire dai dati, predicendo la **classe** di appartenenza *più probabile* per un nuovo dato a partire dal modello sviluppato. Questi modelli si basano sul **teorema di Bayes**.

6.4.1 Teorema di Bayes e regola di decisione

Sia $\mathbf{x}$ un'osservazione e $c \in \mathcal{C}$ una classe candidata. Per decidere la classe usiamo il **teorema di Bayes**:

$$P(c \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c)\, P(c)}{P(\mathbf{x})}.$$

- **Probabilità a priori** $P(c)$: quanto la classe c è probabile *prima* di vedere i dati (in pratica: frequenza della classe nel train).
- **Likelihood** $P(\mathbf{x} \mid c)$: quanto è plausibile osservare $\mathbf{x}$ *se* la classe fosse c.
- **Evidenza** $P(\mathbf{x})$: probabilità complessiva di osservare $\mathbf{x}$ (uguale per tutte le classi).

La decisione ottima *MAP* (Maximum A Posteriori) è

$$\hat{c}(\mathbf{x}) = \arg\max_{c \in \mathcal{C}} P(c \mid \mathbf{x}) = \arg\max_{c \in \mathcal{C}} P(\mathbf{x} \mid c)\, P(c),$$

poiché $P(\mathbf{x})$ non dipende da c e non influisce sull'arg max.

6.4.2 Naive Bayes

Idea. Assumiamo che, fissata la classe c, le feature siano indipendenti (*assunzione naive*). Allora la verosimiglianza fattorizza:

$$P(\mathbf{x} \mid c) = \prod_{j=1}^{d} P(x_j \mid c).$$

Regola di decisione (MAP, in scala logaritmica). Le probabilità condizionali sono molto piccole e un prodotto di tante quantità prossime a 0 può portare problemi di underflow. Per ovviare a questi problemi si considera il **log-likelihood**:

$$\hat{c}(\mathbf{x}) = \arg\max_{c \in \mathcal{C}} \Big[\log P(c) + \sum_{j=1}^{d} \log P(x_j \mid c) \Big].$$

Questo si traduce in una somma anziché in un prodotto di termini.

Stima essenziale delle probabilità.

- **Prior** $P(c)$: frequenza della classe nel training.
- **Attributi discreti**: frequenze condizionate con *Laplace smoothing* ($+\alpha$) per evitare zeri.
- **Attributi continui**: modello gaussiano per $x_j \mid c$ con media e varianza stimate sui dati della classe.

Vantaggi e svantaggi. Molto veloce e facile da implementare, l'assunzione di indipendenza condizionale potrebbe non essere sempre vera e potrebbe portare ad una perdita di accuratezza (tali dipendenze non possono essere modellate da questo modello).

6.4.3 Reti Bayesiane

Una **rete bayesiana** è un DAG le cui variabili $\{X_1, \ldots, X_d\}$ fattorizzano come

$$P(X_1, \ldots, X_d) = \prod_{i=1}^{d} P(X_i \mid \mathrm{Pa}(X_i)),$$

dove $\mathrm{Pa}(X_i)$ sono i genitori di X_i nel grafo. Il DAG codifica indipendenze condizionali; i CPT (tabelle di probabilità condizionate) specificano i parametri.

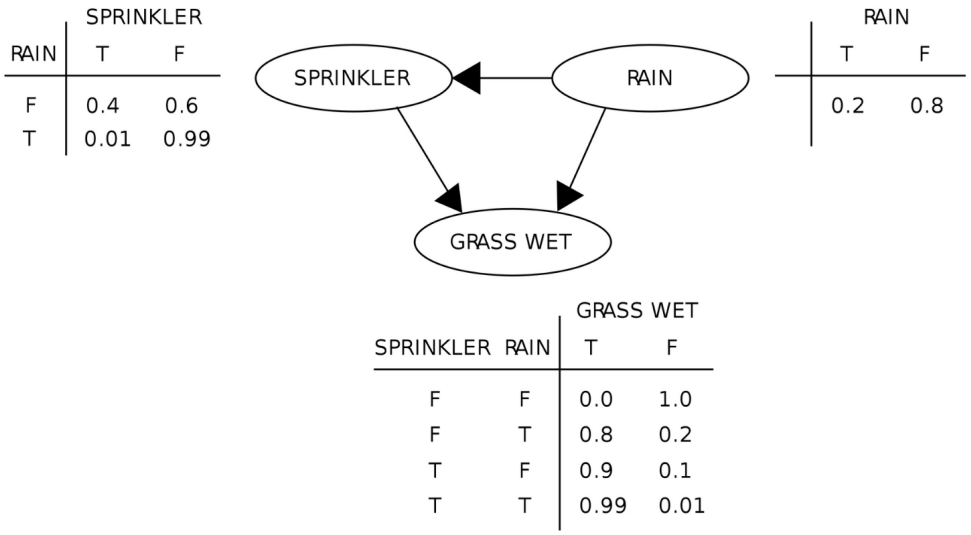

Figura 6.5: Rete bayesiana *Sprinkler–Rain–Grass*: il DAG orientato specifica le dipendenze e indetermina la fattorizzazione della congiunta $P(\mathrm{Rain})\,P(\mathrm{Sprinkler} \mid \mathrm{Rain})\,P(\mathrm{GrassWet} \mid \mathrm{Sprinkler}, \mathrm{Rain})$. Le tabelle mostrano le CPD (Conditional Probability Tables) dei nodi. A differenza di Naive Bayes, le feature possono essere dipendenti dato la/e causa/e (qui *Rain*), e tale dipendenza è resa esplicita dagli archi.

Uso per la classificazione. Dato $\mathbf{x}$, si calcolano (o si approssimano) $P(y \mid \mathbf{x})$ tramite inferenza sul DAG (*esatta* o *approx* con sampling/variational). Le reti bayesiane generalizzano Naive Bayes (che è un caso particolare con Y genitore di tutte le feature e nessun'altra dipendenza).

6.5 Classificatori discriminativi

I classificatori discriminativi stimano direttamente una funzione di decisione $f : \mathbb{R}^d \to \mathbb{R}$ (o, opzionalmente, la probabilità condizionata $P(y \mid \mathbf{x})$) senza modellare la distribuzione congiunta $P(\mathbf{x}, y)$. Dato un nuovo esempio $\mathbf{x}$ si valuta $f(\mathbf{x})$ e si assegna l'etichetta corrispondente. Rispetto ai modelli generativi richiedono generalmente meno assunzioni sui dati. Tipici esempi sono il Perceptron e le Support Vector Machines (SVM).

6.5.1 Classificazione lineare e non lineare

Lineare: la regola di decisione è basata su una combinazione lineare degli attributi

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b,$$

Dove $\mathbf{w} \in \mathbb{R}^d$ è il vettore dei pesi e $b \in \mathbb{R}$ è il bias (termine di soglia).

Non lineare: Il problema di avere spazi non lineare è che non è possibile separare le classi con un iperpiano. Per risolvere questo problema si fa una trasformazioni di spazi vettoriali in spazi di dimensione superiore dove la separazione lineare è possibile. Questo si ottiene tramite il **kernel trick**, che permette di calcolare prodotti scalari in spazi trasformati senza dover esplicitamente mappare i dati.

Esistono anche classificazioni lineari e non lineari binarie, dove si utilizzano approcci simili.

6.5.2 Perceptron

Definizione. Il Perceptron è un classificatore lineare binario che produce la funzione

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b.$$

Con etichette $y \in \{-1, +1\}$. Si parla di un algoritmo di machine learning, quindi è necessario un modo, per il perceptron, di apprendere i parametri $\mathbf{w}$ e b dai dati di addestramento.

Regola di aggiornamento. Dato un esempio $(\mathbf{x}^{(i)}, y^{(i)})$, se la previsione $\hat{y}^{(i)}$ è errata (ossia $\hat{y}^{(i)} \neq y^{(i)}$) si aggiorna ogni peso component-wise secondo la notazione usata in figura:

$$\hat{\vartheta}_j \leftarrow \vartheta_j + \alpha \left(y^{(i)} - \hat{y}^{(i)}\right) x_j^{(i)}, \qquad j = 0, \ldots, d,$$

dove $\alpha > 0$ è il learning rate, ϑ_j indica il valore corrente del peso e $\hat{\vartheta}_j$ il valore aggiornato. Equivalentemente, in forma vettoriale si ottiene

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(y^{(i)} - \hat{y}^{(i)}\right) \mathbf{x}^{(i)}, \qquad b \leftarrow b + \alpha \left(y^{(i)} - \hat{y}^{(i)}\right).$$

Se $\hat{y}^{(i)} = y^{(i)}$ il modello non viene modificato.

Proprietà. Se i dati sono linearmente separabili, il Perceptron converge in un numero finito di aggiornamenti (teorema di Novikoff). In pratica si itera per più epoche o fino a soddisfare un criterio di stop.

Algoritmo. L'algoritmo del perceptron è definito come:

1. Inizializza i pesi $\mathbf{w}$ e il bias b a zero o a valori casuali.
2. Per ogni tupla y_j nel training set:
   (a) Calcola la previsione $\hat{y}^{(i)}$

(b) Se $\hat{y}^{(i)} \neq y^{(i)}$, aggiorna i pesi e il bias secondo la regola di aggiornamento.

(c) Incrementa i e ripeti fino a completare il training set.

3. Ripeti il passo 2 per un numero prefissato di epoche o fino a soddisfare un criterio di stop. Nel caso di learning offline, si ripete il passo 2 finché l'errore medio di classificazione sul training set non scende sotto una soglia prefissata.

Multiclass: One–Vs–One (OVO). One–Vs–One costruisce un classificatore binario Perceptron per ogni coppia di classi (C_p, C_q); per K classi si addestrano $K(K-1)/2$ modelli. In fase di predizione, ogni classificatore vota per una delle due classi che confronta e si assegna la classe con il maggior numero di voti (voting).

Motivazione: OVO è utile quando le classi sono relativamente poche e si desidera che ogni modello risolva una decisione binaria semplice; ogni modello vede dati di due sole classi, spesso permettendo separazioni più semplici e modelli più piccoli. Lo svantaggio principale è il numero di classificatori e la gestione del voto/pareggio.

Algoritmo (per ciascuna coppia (C_p, C_q)):

1. Costruisci il training set rimuovendo istanze non appartenenti a C_p o C_q.
2. Inizializza i pesi ϑ_j e il bias.
3. Per ogni epoca e per ogni esempio $(\mathbf{x}^{(i)}, y^{(i)})$ nel sottoinsieme: calcola $\hat{y}^{(i)} = \text{sign}(f(\mathbf{x}^{(i)}))$; se $\hat{y}^{(i)} \neq y^{(i)}$ aggiorna

$$\hat{\vartheta}_j \leftarrow \vartheta_j + \alpha\,(y^{(i)} - \hat{y}^{(i)})\,x_j^{(i)} \quad (j = 0, \ldots, d).$$

Multiclass: One–Vs–All (OVA). One–Vs–All costruisce un classificatore Perceptron per ogni classe C_k dove il problema è C_k vs "resto". Si addestrano K modelli; alla predizione si calcola il punteggio $f_k(\mathbf{x})$ per ogni modello e si sceglie la classe con il punteggio più alto.

Motivazione: OVA è più parsimonioso in termini di numero di modelli rispetto a OVO (si addestrano K modelli invece di $K(K-1)/2$) e può essere più efficiente quando K è grande. Tuttavia ogni modello OVA affronta un problema sbilanciato (una classe vs tutte le altre), il che può richiedere tecniche di bilanciamento o regolarizzazione.

Algoritmo (per ciascuna classe C_k):

1. Crea etichette binarie $y^{(i)} = +1$ se l'esempio appartiene a C_k, altrimenti $y^{(i)} = -1$.
2. Inizializza i pesi ϑ_j e il bias.
3. Per ogni epoca e per ogni esempio $(\mathbf{x}^{(i)}, y^{(i)})$: calcola $\hat{y}^{(i)} = \text{sign}(f(\mathbf{x}^{(i)}))$; se $\hat{y}^{(i)} \neq y^{(i)}$ aggiorna

$$\hat{\vartheta}_j \leftarrow \vartheta_j + \alpha\,(y^{(i)} - \hat{y}^{(i)})\,x_j^{(i)} \quad (j = 0, \ldots, d).$$

Breve nota comparativa: OVO tende a produrre modelli più specialistici e può funzionare meglio quando le classi sono ben separate a coppie; OVA è più semplice ed efficiente per molti problemi pratici ma richiede attenzione allo sbilanciamento delle classi.

6.5.3 Support Vector Machines (SVM)

Margine massimo. Le SVM cercano l'iperpiano che massimizza il margine tra le due classi. Con etichette $y_i \in \{-1, +1\}$ e dati $(\mathbf{x}_i, y_i)_{i=1}^n$, l'hard–margin SVM è:

$$\begin{aligned} &\min_{\mathbf{w},b} \ \tfrac{1}{2}\|\mathbf{w}\|^2 \\ &\text{s.t.} \ \ y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, n. \end{aligned}$$

Soft–margin (forma standard). Per dati non separabili si introducono slack variables $\xi_i \geq 0$ e un parametro di regolarizzazione $C > 0$ (segue la convenzione usata nel libro):

$$\begin{aligned} &\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{s.t.} \ \ y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \ i = 1, \ldots, n. \end{aligned}$$

Qui C controlla il trade–off tra margine e violazioni (più C grande implica penalità maggiore per gli errori).

Forma duale e kernel trick. La formulazione duale è una QP in termini dei moltiplicatori α_i; la soluzione usa prodotti scalari tra esempi che, sostituiti con un kernel $K(\mathbf{x}, \mathbf{z})$, permettono modellare non linearità. Esempi di kernel comuni:

Polinomiale: $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}\cdot\mathbf{z}+1)^d$, $\quad extRBF : K(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right)$, $\quad extSigmoide : K(\mathbf{x}, \mathbf{z}) = \tanh(k\,\mathbf{x}$

Proprietà e multiclass. SVM è intrinsecamente binaria; il multiclass si ottiene con OVA/OVO o con estensioni dirette. Le SVM tendono a essere efficaci in spazi ad alta dimensione e sono robuste (solo pochi esempi, i support vectors, definiscono la soluzione), ma l'addestramento può essere pesante su grandi dataset.