Data mining

31 luglio 2024

Indice

1 Introduzione al data mining **7**
1.1 Introduzione . . . 7
1.2 Dati . . . 8
1.2.1 Tipi di dati . . . 8
1.2.2 Preprocessing dei dati . . . 8
1.2.2.1 Aggregation . . . 8
1.2.2.2 Sampling . . . 9
1.2.2.3 Dimensionality reduction . . . 9
1.2.2.4 Feature subset selection . . . 9
1.2.2.5 Feature creation . . . 11
1.2.2.6 Discretizzazione . . . 11
1.2.2.7 Trasformazione di variabile . . . 13
1.2.3 Misure di similarità e dissimilarità . . . 14
1.2.3.1 Similarità/dissimilarità tra oggetti con un solo attributo . . . 14
1.2.3.2 Dissimilarità tra oggetti con molteplici attributi 15
1.2.3.3 Similarità tra oggetti con molteplici attributi . . 16
1.2.3.4 Esempi di misure di prossimità . . . 16
1.2.3.5 Problemi nel calcolo della prossimità . . . 17

2 Classificazione **19**
2.1 Introduzione . . . 19
2.1.1 Setting . . . 19
2.1.2 Overfitting . . . 19
2.1.3 Valutare la performance di un classificatore . . . 20
2.1.3.1 Holdout . . . 20
2.1.3.2 Random subsampling . . . 21
2.1.3.3 Cross-validation . . . 21
2.1.3.4 Bootstrap . . . 21
2.2 Decision Tree . . . 22
2.2.1 Costruzione albero . . . 22
2.2.2 Selezione dell'attributo di test . . . 23
2.2.3 Peculiarità della tecnica . . . 24
2.2.4 Overfitting e pruning negli alberi decisionali . . . 25
2.3 k-nearest-neighbor classifier . . . 26
2.3.1 Implementazione . . . 26
2.3.2 Caratteristiche . . . 26
2.4 Classificatori Bayesiani . . . 26

2.4.1 Naive Bayes classifier . . . 27
2.4.1.1 Costruzione del modello . . . 27
2.4.1.2 Utilizzo . . . 28
2.4.1.3 Caratteristiche . . . 28
2.4.2 Bayesian Belief network . . . 28
2.5 Artificial Neural Network . . . 29
2.6 Support Vector Machine . . . 29
2.7 Ensemble Methods . . . 29
2.7.1 Rationale per i metodi ensemble . . . 29
2.7.2 Metodi per costruire un classificatore ensemble . . . 30
2.7.2.1 Bagging . . . 30
2.7.2.2 Random forest . . . 31

3 Association analysis **33**
3.1 Definizione del problema . . . 34
3.1.1 Definizioni . . . 34
3.1.2 Supporto e confidenza . . . 34
3.1.3 Focus e fasi del problema di mining . . . 35
3.2 Identificazione degli itemset frequenti . . . 35
3.2.1 Principio Apriori . . . 36
3.2.2 Generazione dei candidati . . . 36
3.2.3 Individuazione degli itemset frequenti . . . 39
3.3 Algoritmo FP growth . . . 39
3.3.1 Rappresentazione dell'FP-tree . . . 39
3.3.2 Identificazione degli itemset frequenti . . . 40
3.4 Generazione delle regole . . . 44

4 Clustering **47**
4.1 Introduzione . . . 47
4.1.1 Tipi di clustering . . . 48
4.2 Clustering basato su prototipo . . . 49
4.2.1 Algoritmi partitivi . . . 50
4.2.1.1 Introduzione . . . 50
4.2.1.2 Accuratezza della partizione . . . 51
4.2.1.3 Funzioni di perdita . . . 51
4.2.1.4 Funzioni obiettivo non solo quantitative . . . 52
4.2.2 K-means . . . 52
4.2.2.1 Algoritmo base . . . 52
4.2.2.2 Scelta dei centroidi iniziali . . . 53
4.2.2.3 Considerazioni ulteriori . . . 54
4.2.2.4 Limitazioni . . . 54
4.2.2.5 Bisecting K-means . . . 54
4.2.3 Clustering con Mixture models . . . 55
4.2.3.1 Introduzione . . . 55
4.2.3.2 Stima di massima verosimiglianza . . . 56
4.2.3.3 Stimare dei parametri nei mixture model: l'algoritmo EM . . . 56
4.2.3.4 Esempi . . . 56
4.2.3.5 Vantaggi e svantaggi . . . 58
4.3 Clustering gerarchico (agglomerativo) . . . 59

4.3.1 Clustering agglomerativo: algoritmo base 59
4.3.1.1 Algoritmo aggregativo di Anderberg 59
4.3.1.2 Definire la dissimilarità tra cluster 60
4.3.2 Esempi 60
4.3.3 Caratteristiche e issues 61
4.4 Density based clustering 64
4.4.1 Density estimation 64
4.4.1.1 Approcci alla stima di densità 64
4.4.1.2 Metodi di stima della densità 65
4.4.1.3 Stima di densità e clustering 70
4.4.2 DBSCAN 70
4.4.2.1 Selezione dei parametri di DBSCAN 71
4.4.2.2 Punti di forza e debolezza 72
4.4.3 DENCLUE 72
4.4.3.1 Algoritmo 72
4.4.3.2 Cluster riconosciuti 72
4.4.3.3 Pro e contro 74
4.5 Algoritmi scalabili 75
4.5.1 Algoritmi a passo unico 75
4.5.2 BIRCH 76
4.5.2.1 Elementi strumentali dell'algoritmo 76
4.5.2.2 Funzionamento algoritmo 78
4.6 Valutazione/validazione del clustering 79
4.6.1 Misure per cluster partizionali 80
4.6.1.1 Valutazione mediante coesione e separazione . . 80
4.6.1.2 Valutazione mediante matrice di prossimità . . . 82
4.6.2 Misure per cluster gerarchici 82
4.6.3 Determinare il corretto numero di cluster 82
4.6.4 Clustering tendency 83
4.6.5 Misure supervisionate di validità 83
4.6.5.1 Misure classification oriented 84
4.6.5.2 Misure similarity oriented 84
4.6.5.3 Misure per il clustering gerarchico 85

Capitolo 1

Introduzione al data mining

1.1 Introduzione

Data Mining e Knowledge Discovery Il **data mining** è il processo di scoperta automatica di informazioni utili in grandi moli di dati; operativamente mischia i mezzi tradizionali di analisi dei dati con sofisticati algoritmi per la processazione di grandi quantità di dati.
Il data mining è la fase centrale degli step che costituiscono il **knowledge discovery in databases** (**KDD**), ovvero il processo (complessivo) di convertire dati grezzi in informazioni utili; altre fasi sono:

1. data **preprocessing**:trasformazione dei dati grezzi nel formato appropriato per la successiva analisi (integrazione di varie fonti, cleaning per la rimozione di noise ed osservazioni duplicate, selezione di records/variabili necessarie per l'analisi). Precede il data mining;

2. **postprocessing**: fasi susseguenti all'analisi che servono per integrarne i risultati in un processo decisionale (scelta dei risultati più significativi, visualizzazione, interpretazione di pattern).

Ragioni all'origine del data mining

- scalabilità: la quantità di dati sta crescendo e pertanto sono sempre più necessari nuovi algoritmi atti a maneggiare tali volumi

- nuovi dati: non cresce solo la quantità, ma anche la complessità dei dati (dimensione, spazio, tempo) e la loro eterogeneità (nuovi tipi di dati: pagine xml dati di dna ecc)

- proprietà e distribuzione geografica dei dati: se i dati sono fisicamente distribuiti è necessario lo sviluppo di algoritmi distribuiti

Task di data mining I task tipici sono:

- *predizione*: si suddivide in classificazione e regressione a seconda che la variabile studiata sia qualitativa o quantitativa, ma la sostanza è costruire un modello per predire la variabile come una funzione di altre variabili esplicative

- *analisi di associazione*: utilizzato per trovare caratteristiche (variabili) nei dati fortemente associate/correlate tra loro
- *cluster analysis*: utilizzato per individuare gruppi simili tra loro e differenti dagli altri
- *anomaly detection*: identificazione di ossevazioni con caratteristiche sensibilmente differenti dal resto dei dati, gli outlier

1.2 Dati

1.2.1 Tipi di dati

Nel datamining un dataset è visto come una collezione di **data object** (altrimenti detti record, osservazioni ecc) descritti da un determinato numero di **attributi** (altrimenti dette *variabili*, *field feature*) che catturano le caratteristiche base dell'oggetto.

Nello specifico un **attributo** è una proprietà/caratteristica di un oggetto che può variare tra oggetti e/o nel tempo.
Una **scala di misurazione** è una regola (funzione) che associa un valore simbolico o numerico all'attributo di un oggetto; formalmente il processo di *misurazione* è l'applicazione di una scala di misurazione ad un particolare attributo di uno specifico oggetto

I tipi di attributi sono i soliti: nominali e ordinali per gli attributi qualitativi a intervalli o a rapporti per le gli attributi quantititative

1.2.2 Preprocessing dei dati

Il preprocessing è la fase di preparazione dei dati strumentale a quella successiva di data mining vero e proprio. Esso consta in un ampio numero di differenti strategie/tecniche differenti; qui ci si concentra su:

1. aggregation
2. sampling
3. dimensionality reduction
4. feature subset selection
5. feature creation
6. discretizzazione
7. trasformazione di variabile

1.2.2.1 Aggregation

Consta nel combinare due o più oggetti in uno singolo riassuntivo (ad esempio da singole transazioni a transazioni giornaliere). Ad esempio per gli attributi quantitativi si procede ad aggregazione applicando una funzione (somma o una media tipicamente) all'insieme degli attributi aggregati.

Vi sono diverse *motivazioni* per aggregare:

- data set più contenuti richiedono minore memoria e tempo di processazione nelle fasi successive;
- l'aggregazione può fornire una visuale di più alto livello;

Svantaggio dell'aggregazione può essere quello di perdere informazione rilevante nella sintesi.

1.2.2.2 Sampling

Consta nella selezione di un subset di data objects da analizzare. Le motivazioni sono ridurre la base di dati per rendere più rapida la sua processazione.

1.2.2.3 Dimensionality reduction

I dataset possono essere caratterizzati da un grande numero di features dei data objects; la *dimensionality reduction* diminuisce il numero di attributi da anlizzare creando nuovi attributi che siano una sintesi/combinazione dei vecchi attributi.
I *benefici* sono:

- velocità di esecuzione
- si facilita la visualizzazione
- si lumita la *curse of dimensionality*: con questo termine si indica il fatto che molteplici algoritmi dati diventano faticano di più/sono meno affidabili all'aumento della dimensionalità dei dati da gestire: ad esempio nella classificazione non vi sono data objects a sufficienza che permettano la creazione di un modello affidabile, mentre nel clustering densità e distanza tra punti diventano meno significativi

Le tecniche più comuni per la riduzione della dimensionalità, in special modo per dati quantitativi, utilizzano tecniche di algebra lineare per proiettare i dati da uno spazio multidimensionale ad uno spazio con un numero di dimensioni minori[1]:

- la **Principal components analysis** (PCA) è una tecnica per attributi continui che identifica nuovi attributi (le componenti principali) che sono combinazioni lineari degli atttributi originali, sono ortogonali tra loro e catturano il massimo ammontare di variabilità nei dati
- la **Singular value decomposition** è un'altra tecnica di algebra lineare comunemente utilizzata per la riduzione della dimensionalità

1.2.2.4 Feature subset selection

Tecnica alternativa alla dimensionality reduction per ridurre il numero di attributi analizzati (i benefici sono analoghi a quelli per la dimensionality reduction); potrebbe sembrare che un approccio del genere porti necessariamente ad una perdita di informazioni, ma questo non è il caso se sono presenti variabili *ridondanti* o *irrilevanti*.

[1] Confronta appendice A e B del Tan per approfondimenti

Al di la del buon senso e la conoscenza dell'ambito analizzato vi sono tre approcci standard:

- approcci *embedded*: se la scelta delle feature è parte integrante dell'algoritmo di data mining impiegato (nel seguito), che sceglie autonomamente quali variabili impiegare (ad esempio gli alberi decisionali funzionano in questo modo)

- approcci *filter*: feature selezionate prima dell'applicazione dell'algoritmo di data mining e indipendentemente da questo (ad esempio usare set di attributi la cui corrleazione a coppie è la minore possibile)

- approcci *wrapper*: si utilizza l'algoritmo di data mining in maniera strumentale (una black box) per trovare il subset migliore di attributi

Il processo di feature selection Approfondiamo gli ultimi due (il primo è algoritmo specifico). Filter approach e wrapper:

- si compongono entrambe di quattro fasi:

  1. una misura di valutazione del subset analizzato
  2. una strategia di ricerca che controlli la generazione di un nuovo subset di features: diverse strategie di ricerca possono essere impiegate, dovrebbero essere computazionalmente efficienti e trovare insiemi ottimali di features (tradeoff necessari tra le due)
  3. un criterio di stop nella ricerca (di solito basato sul numero di iterazioni o su altre soglie minime/massime da superare)
  4. una procedura di validazione: alcune idee:
     - eseguire l'algoritmo di data mining sull'insieme completo di dati e su un suo subset, sperando che quest'ultimo produca risultati che sono migliori o almeno buoni quanto quelli prodotti dall'esecuzione sul dataset completo
     - utilizzare diversi algoritmi di scelta delle feature e comparare i risultati dell'esecuzione dell'algoritmo di data mining

- si differenziano solamente per la prima delle quattro componenti:

  - per il wrapper la valutazione del subset usa un algoritmo di data mining
  - per il filter la tecnica di valutazione è distinta dall'algoritmo di data mining

Feature weighting Alternativo alla scelta drastica è quello di applicare pesi alle variabili, basati sulla conoscenza dell'ambito specifico o determinati automaticamente (ad esempio i metodi di classificazione support vector machines producono pesi per ogni variabile)

1.2.2.5 Feature creation

L'idea è creare dai dati originali un nuovo insieme di attributi che catturino le informazioni importanti. Vi sono tre metodi:

- feature extraction
- mappting dei dati ad un nuovo spazio
- feature construction

Creazione di feature L'idea è combinare gli attributi in qualcosa di sintetico/informativo, facendo uso di domain expertise. Ad esempio calcolare il BMI a partire da peso e altezza.

Feature extraction Consiste nella estrazione di informazioni di più alto livello a partire da dati su aspetti specifici; ad esempio invece dell'elevato numero di pixel e del loro colore, l'informazione se l'immagine contiene un volto o un paesaggio.
Questi processi sono tipicamente dipendenti dallo specifico contesto in cui si opera.

Mapping dei dati ad un nuovo spazio Si effettua una trasformazione radicale del dataset applicando qualche trasformata (ad esempio trasformata di Fourier per le serie storiche).

1.2.2.6 Discretizzazione

Alcuni algoritmi di data mining, specialmenti certi di classificazione, richiedono che i dati siano in forma di attributi categorici, pertanto può essere necessario, alternativamente:

- trasformare un *attributo continuo* in uno categorico (**discretizzazione**) o nello specifico in uno dicotomico (**binarizzazione**);
- ridurre il numero di classi di un *attributo categorico* ordinale o nominale (con molteplici classi): questo è fatto sulla base della conoscenza del campo analizzato o verificando/confrontando la susseguente performance dell'algoritmo di data mining

Nel prosieguo ci concentriamo sul primo dei due; una distinzione di base sui metodi di discretizzazione è se l'informazione di classe è utilizzata (*supervised*) o meno (*unsupervised*).

Approcci di discretizzazione unsupervised Se non si fa uso di informazioni di classe si possono adottare i seguenti approcci:

- equal width: si divide il range dei valori assunti dagli attributi in intervalli di ampiezza uguale
- equal frequency: si divide il range dei valori assunti dagli attributi in intervalli di ugual frequenza (ma verosimilmente diversa ampiezza)
- k-means clustering
- visualizzazione e scelta dei cutoff

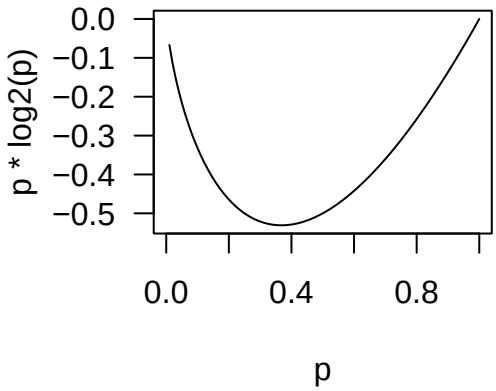
Discretizzazione supervised Ipotizzando invece di avere una variabile quantitativa da ripartire in gruppi ma di disporre anche di una variabile qualitativa che fornisce una informazione di classe (ad esempio un colore) possiamo adottare l'approccio di piazzare gli split in modo da massimizzare la *purezza* degli intervalli (ovvero il fatto che gli intervalli numerici tendano maggiormente ad includere uno e un solo colore). Gli approcci basati sull'**entropia** sono i più promettenti.

Supponiamo che:

- vi siano k differenti classi (colori)
- m_i sia il numero di valori della i-esima partizione della variabile numerica
- m_{ij} siano il numero di valori della j-esima classe (colore) nell'i-esima partizione
- $p_{ij} = m_{ij}/m_i$ come la probabilità (frazione di valori della classe j nell'intervallo i

L'*entropia* e_i dell' i-esimo *intervallo* è data dall'equazione:

$$e_i = -\sum_{j=1}^{k} p_{ij} \log_2 p_{ij} \tag{1.1}$$



Possiamo vedere il termine entro sommatoria

Ora supponiamo di avere in una determinata partizione:

- n intervalli
- complessivamente m valori
- m_i il numero di valori di un dato i-esimo intervallo
- $w_i = m_i/m$ la proporzione di valori che appartengono ad un dato i-esimo intervallo

L'*entropia totale* di una prefissata partizione è la media ponderata delle entropie individuali:

$$e = \sum_{i=1}^{n} w_i e_i \tag{1.2}$$

Intuitivamente l'entropia misura la *purezza* dell'intervallo; se l'intervallo contiene solo valori di una classe (colore) allora la sua entropia è 0 e non contribuisce nulla all'entropia generale della ripartizione; se alternativamente i colori sono egualmente ripartiti all'interno delle classi, l'entropia raggiunge un massimo.

Un approccio semplice al partizionamento di un attributo continuo:

- si inizia bisezionando l'intervallo di valori in maniera tale che vi sia la minore entropia possibile
- lo splitting è ripetuto nuovamente, scegliendo l'intervallo con l'entropia peggiore (maggiore)
- si procede sino a che un determinato numero di intervalli non è stato raggiunto o un altro criterio soddisfatto

1.2.2.7 Trasformazione di variabile

Vi sono due importanti tipi: trasformazioni funzionali e normalizzazioni

Trasformazioni funzionali Si applica una funzione matematica ad ogni valore individualmente: se x è la variabili, esempi di trasformazioni possono essere $x^k, \log x, e^x, \sqrt{x}, 1/x \sin x$ o $|x|$
Ragioni di una trasformazione in data mining (al di la della normalità in distribuzione) possono essere praticità (ad esempio adoperare il $\log_{10}$ per ottenere l'ordine di grandezza di misure di magnitudo profondamente differenti).
Ovviamente le trasformazioni debbono essere applicate con cauzione dato che cambiano la natura del dato, il che può essere problematico se ciò non è pienamente apprezzato.

Normalizzazioni Serie di funzioni applicate ad un dato per far si che abbia determinate proprietà. Ad esempio:

- la standardizzazione $(x - \mu_x)/s_x$ in statistica fa si che in seguito la distribuzione abbia media 0 e sd unitaria: se differenti variabili debbono essere combinate in qualche modo, una trasformazione del genere è spesso necessaria per evitare che variabili con valori larghi prevalgano il risultato dei calcoli
- se vi sono outlier nella formula della standardizzazione
  - si adopera la mediana invece che la media
  - si passa alla deviazione standard assoluta $\sigma_A = \sum_{i=1}^{m} |x_i - \mu|$ dove x_i è l'i-esimo valore della variabile, m il numero di oggetti e μ la mediana

1.2.3 Misure di similarità e dissimilarità

Sono misure rilevanti, in quanto presenti in numerosi algoritmi di data mining; la

- *similarità* tra due oggetti è una misura (solitamente non negativa e spesso compresa tra 0 e 1) che indica la somiglianza;
- *dissimilarità* funziona specularmente e può andare nel range $[0, 1]$ o in $[0, \infty)$

Ad un aumento dell'indice si associa un aumento della similarità (rispettivamente della dissimilarità).
Per riferirsi congiuntamente a questi concetti si adopera a volte il termine **prossimità**. Dato che la *prossimità* tra due oggetti è una funzione della prossimità tra i relativi attributi, in seguito ad una carrelata sulle trasformazioni utili tra indici di prossimità, partiamo descrivendo le misure per quantificare la prossimità tra oggetti aventi un solo attributo, per passare poi ad oggetti multi-attributo.

Trasformazioni utili Può essere necessario effettuare delle conversioni:

- per riportare una similarità (o dissimilarità) espressa mediante un numero finito di valori (ad esempio 1 per niente simili e 10 identici) nel range $[0, 1]$ si può applicare la normalizzazione

$$x' = \frac{x - \min x}{\max x - \min x} \tag{1.3}$$

- per riportare una dissimilarità in $[0, \infty)$ ad $[0, 1]$ si può adoperare

$$x = \frac{x}{1 + x} \tag{1.4}$$

- per trasformare similarità in dissimilarità, se entrambe sono in $[0, 1]$, allora sono il complemento ad 1 l'una dell'altra, per cui ad esempio possiamo adottare

$$d = 1 - s \tag{1.5}$$

  con d dissimilarità ed s similarità; alternativamente possiamo definire la similarità come il negativo della dissimilarità (o viceversa)

$$d = -s \tag{1.6}$$

  In generale, qualsiasi funzione monotona decrescente può essere utilizzata per convertire similarità in dissimilarità o viceversa.

1.2.3.1 Similarità/dissimilarità tra oggetti con un solo attributo

Le misure più frequenti nel campo monodimensionale sono riportate in tabella 1.1 dove per compattezza si indica $d(x, y) = d$, $s(x, y) = s$, intendendo con x e y il valore assunto dall'attributo nei vari data object.

Tipo	Dissimilarità	Similarità
Nominale	$d = \begin{cases} 0 & \text{se } x = y \\ 1 & \text{se } x \neq y \end{cases}$	$s = \begin{cases} 0 & \text{se } x \neq y \\ 1 & \text{se } x = y \end{cases}$
Ordinale[2]	$d = \frac{\lvert x - y \rvert}{n - 1}$	$s = 1 - d$
Quantitativa	$d = \lvert x - y \rvert$	$s = -d$ $s = \frac{1}{1+d}$ $s = 1 - \frac{d - \min d}{\max d - \min d}$

Tabella 1.1: Misure di prossimità nel campo monodimensionale

1.2.3.2 Dissimilarità tra oggetti con molteplici attributi

Supponendo di avere due punti $\mathbf{x}$ e $\mathbf{y}$ in uno spazio ad n dimensioni una misura di *distanza* generale tra i due punti è la **distanza di Minkowski**, definita come

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^{n} |x_k - y_k|^r \right)^{\frac{1}{r}} \tag{1.7}$$

con x_k e y_k rispettivamente il k-esimo attributo di $\mathbf{x}$ e $\mathbf{y}$.
Scegliendo il *parametro* r si hanno alcuni casi notevoli:

- se $r = 1$ ha la **Manhattan Distance**; a parte a pensare all'esempio delle strade di Manhattan, un esempio comune è la **Hamming distance** che consiste nel numero di bits che sono differenti tra due oggetti che hanno solamente attributi binari (ad esempio due vettori binari)
- se $r = 2$ ha la **Distanza euclidea**

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2} \tag{1.8}$$

- se $r = \infty$ si ha la **Supremum distance**, ovvero viene calcolata il massimo delle distanze tra i singoli attributi dei due oggetti.[3]

Dato poi un insieme di m punti è possibile costruire una matrice $m \times m$ di distanze, chiamata appunto **distance matrix** (simmetrica), contenente le distanze delle coppie di punti considerate.
Distanze come la euclidea hanno alcune proprietà notevoli:

- non negatività:

$$d(\mathbf{x}, \mathbf{y}) \geq 0, \ \forall \mathbf{x}, \mathbf{y} \tag{1.10}$$
$$d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y} \tag{1.11}$$

[3]Formalmente la supremum distance è definita come

$$d(\mathbf{x}, \mathbf{y}) = \lim_{r \to \infty} \left(\sum_{k=1}^{n} |x_k - y_k|^r \right)^{\frac{1}{r}} \tag{1.9}$$

- simmetria:

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}),\ \forall \mathbf{x}, \mathbf{y} \tag{1.12}$$

- disuguaglianza triangolare

$$d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}),\ \forall \mathbf{x}, \mathbf{y}, \mathbf{z} \tag{1.13}$$

Misure che soddisfino tutte e tre le proprietà sono dette **metriche**

1.2.3.3 Similarità tra oggetti con molteplici attributi

Se $s(\mathbf{x}, \mathbf{y})$ è la similarità tra due punti $\mathbf{x}$ e $\mathbf{y}$, proprietà tipiche sono:

$$0 \leq s \leq 1 \tag{1.14}$$
$$s(\mathbf{x}, \mathbf{y}) = 1 \iff \mathbf{x} = \mathbf{y} \tag{1.15}$$
$$s(\mathbf{x}, \mathbf{y}) = s(\mathbf{y}, \mathbf{x}),\ \forall \mathbf{x}, \mathbf{y} \tag{1.16}$$

Non vi è un equivalente della disuguaglianza triangolare per le misure di similarità; a volte possibile mostrare che una misura di similarità può essere convertita di una metriche[4].

1.2.3.4 Esempi di misure di prossimità

In questa sezione vediamo esempi specifici di misure di similarità e dissimilarità.

Misure di similarità per dati binari Siano $\mathbf{x}$ e $\mathbf{y}$ due oggetti consistenti di n attributi binari; la comparazione dei due oggetti conduce alle seguenti quantità:

- f_{00}: numero di attributi dove $\mathbf{x}$ è 0 e $\mathbf{y}$ è 0
- f_{01}: numero di attributi dove $\mathbf{x}$ è 0 e $\mathbf{y}$ è 1
- f_{10}: numero di attributi dove $\mathbf{x}$ è 1 e $\mathbf{y}$ è 0
- f_{11}: numero di attributi dove $\mathbf{x}$ è 1 e $\mathbf{y}$ è 1

Simple matching coefficient Il SMC conta egualmente presenze ed assenze in maniera uguale e corrisponde alla percentuale di item dove i due oggetti sono uguali. È definito come

$$SMC = \frac{f_{00} + f_{11}}{f_{00} + f_{01} + f_{10} + f_{11}} \tag{1.17}$$

[4] Ad esempio cosine e jaccard distance che si vedrà a breve

Coefficiente di Jaccard Nelle applicazioni che riguardano attributi binari asimmetrici (per i quali è più interessante l'1 corrispondente alla presenza della caratteristica che lo 0, corrispondente all'assenza) per calcolare la similarità di due oggetti (ad esempio gli item di due transazioni fatte da altrettanti clienti[5]) non si vogliono considerare gli esiti di assenza perché se le variabili sono fortemente asimmetriche (pochi 1 e per lo più 0) rischiano di confondere e rendere tutti fittiziamente simili (applicando l'SMC)

$$J = \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \tag{1.18}$$

Cosine similarity I documenti scritti sono spesso rappresentati come vettori dove ogni attributo rappresenta la frequenza con cui una determinata parola occorre nel documento (escluse parole di poco interesse, parole comuni); si tratta di dati sparsi (con molti 0) in cui a differenza del caso degli item nel carrello vi possono esser dati di conteggio (> 1).

La cosine similarity tra due vettori multivariati, consiste semplicemente nel calcolare il coseno tra i due

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|} \tag{1.19}$$

Se la cosine similarity[6] è:

- 1, l'angolo tra $\mathbf{x}$ e $\mathbf{y}$ è 0 e a parte la lunghezza i due vettori hanno la massima similarità
- 0, l'angolo tra i due è di 90° ed essi non condividono alcun termine in comune

Extended Jaccard coefficient (Tanimoto coefficient) Misura di applicazione simile alla cosine similarity è la versione estesa del coefficiente di Jaccard, che si riduce al coefficiente di Jaccard nel caso di attributi binari. È definito come:

$$EJ(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \mathbf{x} \cdot \mathbf{y}} \tag{1.20}$$

Correlazione La correlazione ρ (di Pearson intende) tra due vettori è una misura della relazione lineare dei due; vi è perfetta correlazione se $|\rho| = 1$

1.2.3.5 Problemi nel calcolo della prossimità

Standardizzazione e correlazione tra misure di distanza Nel caso la distanza sia multidimensionale e vi siano varianze diverse per attributi diversi o vi sia correlazione tra un attributo e l'altro (nel gruppo complessivo delle osservazioni) utilizzare distanze non normalizzate (come ad esempio quella euclidea) può far si che la distanza tra due oggetti dipenda da uno o pochi attributi di questi che presentano la maggior variabilità.

[5] Non consideriamo momentaneamente la possibilità che un cliente acquisti più confezioni dello stesso prodotto.

[6] La misura è necessariamente nell'intervallo $[0, 1]$ poiché il calcolo proviene da conteggi nulli o positivi

La **distanza di Mahalanobis** è utile quando gli attributi sono correlati, hanno differenti range di valori e la distribuzione dei dati è approssimativamente gaussiana (multivariata). È definita come

$$\text{mahalanobis}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{y})^T \tag{1.21}$$

dove $\boldsymbol{\Sigma}^{-1}$ è l'inversa della matrice di varianza/covarianza.
Il calcolo è computazionalmente esoso, ma può essere meritevole se i dati sono correlati; se gli attributi sono relativamente scorrelati ma hanno differenti variabilità, allora la standardizzazione delle variabile (pre uso della distanza euclidea) è sufficiente.

Combinare similarità di attributi eterogenei Molte delle definizioni precedenti di similarità si bassano sull'assunto che gli attributi siano dello stesso tipo; per calcolare la dissimilarità di oggetti che presentano varie tipologie di attributi si può calcolare la similarità tra ogni attributo separatamente facendo uso delle formule di tabella 1.1 e combinare queste in maniera tale che la risultante similarità complessiva sia in $[0, 1]$.
Possiamo applicare due varianti al procedimento base di cui sopra:

- nel caso vi siano attributi assimmetrici (e rischio di trattare tutto simile per utilizzo di SMC invece che Jaccard), il fix più semplice è di ometterli quando il loro valore è 0 per entrambi gli oggetti la cui similarità è confrontata.
- se vi sono attributi che si reputano più importanti si possono applicare pesi (tipicamente che sommano ad 1 e sono interpretabili come percentuale di peso sul computo totale)

Alla luce di queste considerazioni, complessivamente, la similarità tra due oggetti $\mathbf{x}$ e $\mathbf{y}$, aventi n attributi può essere espressa come

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^{n} w_k \delta_k s_k(\mathbf{x}, \mathbf{y})}{\sum_{k=1}^{n} \delta_k} \tag{1.22}$$

dove:

- s_k è la similarità univariata tra i due oggetti considerando il k-esimo attributo
- δ_k è una variabile indicatrice pari a 0 se l'attributo è asimmetrico ed entrambi gli oggetti hanno un valore di 0 o se uno degli oggetti a valori mancanti per l'attributo medesimo
- w_k è il peso assegnato al k-esimo attributo

Al fine di considerare i pesi, la definizione della distanza di Minkowski può essere modificata come segue:

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^{n} w_k \left| x_k - y_k \right|^r \right)^{\frac{1}{r}} \tag{1.23}$$

Capitolo 2

Classificazione

2.1 Introduzione

2.1.1 Setting

In questo capitolo sezione ci si occupa di classificazione e in particolar modo di **classificazione supervisionata**:

- l'input è una collezione di record ove ogni record è caratterizzato da una tupla $(\mathbf{x}, y)$ dove $\mathbf{x}$ è un set di attributi e y è un attributo qualitativo che corrisponde alla classe dell'oggetto;
- l'obiettivo è costruire un **modello di classificazione** che mappi ogni set di attributi $\mathbf{x}$ di un gruppo di dati (detto **training set**) ad una determinata classe di y (conosciuta) su un set di dati di training sia per descrivere il campione di training (*descriptive modeling*), che ...
- ... per poi poterlo applicare ad un nuovo set di dati (detto **test set**) al fine di predire il gruppo di appartenenza sconosciuto (*predictive modeling*)

La classificazione si dice supervisionata perché almeno in un primo momento (quello di training) è conosciuta la classe di appartenenza y degli oggetti analizzati.
Esempi di metodi impiegati per la costruzione del modello di classificazione (anche detti, appunto, classificatori) sono: decision tree, rule-based classifiers, neural networks, support vector machines e classificatori bayesiani.

2.1.2 Overfitting

Gli errori commessi da un modello di classificazione sono generalmente suddivisi in due tipi:

- training errors: percentuale di errori di classificazione commessi sul test set
- generalization errors: : percentuale attesa di errore di classificazione del modello se applicato su record nuovi (ad esempio il validation set)

Un buon modello deve avere basso il training error ma anche basso il generalization. Il problema dell'overfitting è il seguente: un modello che fitti il training set troppo bene può avere performance peggiore sul generalization error rispetto ad un modello con un training error più alto.
L'overfitting può essere dovuto a:

- alla presenza di *noise* nel training set: ad esempio errori di inserimento nella classe attributo fanno si che se l'albero è costruito per riprodurre al meglio tale dataset incontrerà verosimilmente più errori su casistica nuova;

- *mancanza di rappresentatività* (basato su troppo pochi elementi) della popolazione nel training set

- *multiple comparison* e inflazione dell'errore di prima specie: l'effettuare confronti e valutazioni ripetutamente fa si che ci si esponga in misura via via maggiore a falsi positivi (associazioni puramente casuali nei dati) che influenzino la costruzione del classificatore, a meno che le soglie di decisione non vengano opportunamente modificate per tenere conto dei confronti multipli;

- *complessità* del modello: maggiore essa è e maggiore sarà il rischio di overfitting

2.1.3 Valutare la performance di un classificatore

La valutazione della performance di un modello di classificazione si basa sulla percentuale dei record del test set correttamente previsti; i conteggi necessari sono rappresentati nella consueta tabella 2×2 dove si confrontano classe predetta ed effettiva (detta *confusion matrix*).
Tra i metodi più impiegati si hanno metodo dell'holdout, random subsampling, cross-validation e bootstrap.

2.1.3.1 Holdout

I dati a disposizioni vengono partizionati in due set disgiunti, il training set e il test set. Il modello è costruito sul training set e la sua performance è valutata sul test set.
Le limitazioni sono:

- un numero minore di casi è disponibile per il training (rispetto all'utilizzare tutti i dati a nostra disposizione)

- vi è *trade off tra stabilità del modello e affidabilità della stima di errore*: se il training set è troppo risicato la variabilità potenziale dell'albero costruito aumenta, mentre se il test set è troppo piccolo perde precisione la stima di errore (a livello di intervallo di confidenza)

- essendo il training e il test set presi dallo stesso insieme, non sono del tutto indipendenti: una classe sovrarappresentata in un set sarà sottorappresentata nell'altro e viceversa

2.1.3.2 Random subsampling

Si ripete il metodo holdout diverse volte per creare variabilità nello splitting di training/test set, dopodiché si calcola una media dell'accuratezza ottenuta nelle varie iterazioni.
Limiti:

- ancora non si fa utilizzo di tutti i dati per il training
- non si ha controllo sul numero di volte ogni record è usato, per cui alcuni record item potrebbe essere usati come training più di altri

2.1.3.3 Cross-validation

In questo approccio ogni record è utilizzato lo stesso numero di volte per il training ed esattamente una volta nel testing.
Il caso generale è il k**-fold cross validation**:

- i dati sono suddivisi in k partizioni di dimensioni uguali
- la procedura dura k iterazioni:
  1. $k-1$ gruppi sono selezionati per fare il training del modello
  2. la partizione rimanente viene utilizzata per il calcolo dell'errore
- l'errore totale è calcolato a partire da una somma o media degli errori delle varie iterazioni

Un caso speciale della k-fold cross-validation si ha quando $k = N$, approccio che viene detto **leave one out**. Quest'ultimo approccio ha il vantaggio di utilizzare più dati possibili per il training, mentre il punto di debolezza è la lunghezza della procedura (dovuta alla costruzione di N modelli). Infine la varianza di ciascun test è abastanza alta dato che si fonda solo su una osservazione.

2.1.3.4 Bootstrap

Fino ad ora i metodi estraevano senza reimmissione; il bootstrap invece esegue la seguente procedura b volte:

1. crea un training set estraendo con reimmissione da un pool di unità.
   Si noti che quando N è sufficientemente largo

$$\lim_{N\to\infty} 1 - \left(1 - \frac{1}{N}\right)^N = 1 - e^{-1} = 0.632 \tag{2.1}$$

   in altree parole 63.2% è la probabilità di una generica unità di essere presente in almeno uno dei training set del bootstrap (per N alto che coincide sia con la dimensione di ogni campione che con il numero di ripetizioni del bootstrap).

2. le unità non estratte entrano a far parte del validation set
3. il modello ottenuto sul training set è applicato al test set e viene calcolato l'errore e_i dell'i-esimo campione $1 < i < b$

In seguito al calcolo dell'errore per tutti le b iterazioni si necessiterà di aggregare queste misure di errore in una unica. La varianti più utilizzata è il **bootstrap .632** che calcola l'accuratezza complessiva combinando l'accuratezza di ogni campione bootstrap (ϵ_i) con l'accuratezza calcolata da un training set che contenga tutti i dati originali acc_s:

$$acc_{boot} = \frac{1}{b}\sum_{i=1}^{b}(0.632 \cdot \epsilon_i + 0.368 \cdot acc_s) \tag{2.2}$$

2.2 Decision Tree

Un albero di decisione è una struttura ad albero in cui:

- ogni nodo interno denota un test su un attributo;
- ogni ramo rappresenta un risultato del test;
- i nodi foglia rappresentano classi di un determinato attributo.

Per classificare un oggetto, i valori degli attributi vengono usati per calcolare i test nei nodi interni dell'albero, a partire dalla radice. I risultati dei nodi determinano il percorso seguito dalla radice a un nodo foglia, che rappresenta la previsione per la classe del nuovo oggetto.

2.2.1 Costruzione albero

L'algoritmo per la costruzione di un albero di decisione è un algoritmo greedy, ricorsivo, divide et impera, che accetta in input un insieme di dati di training di ciascuno dei quali si conosce la classe:

- inizialmente l'albero contiene un solo nodo che rappresenta tutti i dati di training
- se i dati sono tutti della stessa classe, il nodo diventa una foglia ed è etichettato con la classe
- altrimenti, si usa una misura (information gain) di entropia per selezionare l'attributo che separa meglio i dati in classi individuali. L'attributo diventa l'attributo di test o di "decisione" del nodo
- viene creato un ramo per ogni possibile risultato del test e si partizionano i dati conseguentemente
- l'algoritmo viene chiamato ricorsivamente per ogni nuovo nodo e corrispondente insieme di dati di training. L'attributo scelto per il test al nodo non viene considerato nelle chiamate ricorsive

In seguito alla costruzione dell'albero decisionale una fase di **potatura** (*tree pruning*) può essere effettuata per: ridurre le dimensioni dell'albero e renderlo meno suscettibile al fenomeno dell'*overfitting*

2.2.2 Selezione dell'attributo di test

Nella costruzione, le misure utilizzate per selezionare lo split migliore (determinato attributo con una data ripartizione) sono spesso basate sul grado di impurità dei nodi figli che ne risulterebbero.
Se:

- t indica un dato nodo;
- c è il numero delle classi dell'attributo outcome;
- $p(i|t)$ è la percentuale di record al nodo t che appartengono alla classe i-esima dell'attributo outcome

una misura comune di impurità raggiunta da un singolo nodo l'entropia raggiunta da un singolo nodo è

$$\text{Entropy}(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t) \tag{2.3}$$

Questa è la misura di impurità che adotteremo noi[1]

Giunti ad un determinato punto dell'albero, per confrontare la performance di diversi attributi/split candidati confrontare il grado di impurità del nodo genitore con il grado di impurità dei figli una volta effettuato lo split. Il guadagno Δ è un criterio che può essere adottato per giudicare la bontà dello split

$$\Delta = I(\text{nodo genitore}) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j) \tag{2.4}$$

dove

- $I(\cdot)$ è la misura di impurità adottata (nel nostro caso l'entropia)
- N è il numero di record che appartengono al nodo genitore
- k sono i nodi figli della partizione considerata
- $N(v_j)$ è il numero di record che appartengono al nodo figlio $j$esimo
- $I(v_j)$ è la misura di impurità del j-esimo nodo figlio

Alcune considerazioni:

- gli algoritmi che costruiscono gli alberi spesso scelgono un attributo/split che massimizzi il guadagno Δ; dato poi che l'impurità del genitore è costante, alla fine il tutto la massimizzazione del guadagno è equivalente alla minimizzazione delle misure di impurità pesate dei nodi figli (sotto vari attributi/split candidati)
- quando l'entropia è scelta come misura di impurità Δ è chiamato **information gain**

[1] Altre misure di impurità possibili sono:

$$\text{Gini}(t) = 1 - \sum_{i=1}^{c} [p(i|t)]^2$$

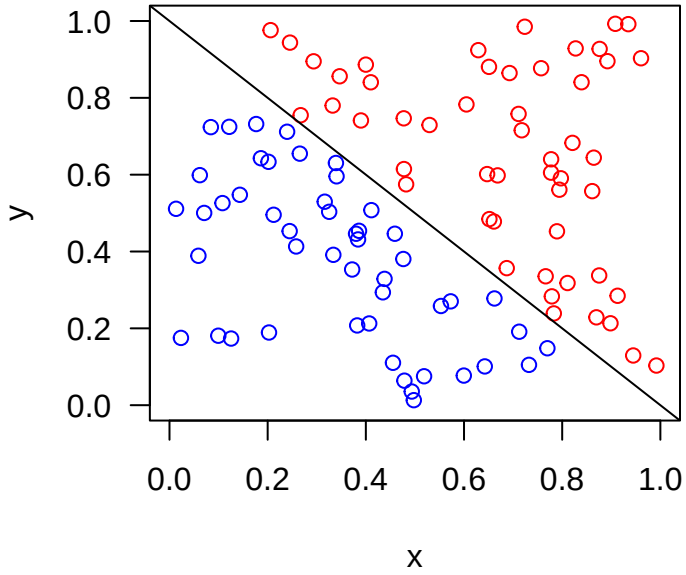
$$\text{ClassificationError}(t) = 1 - \max_i [p(i|t)]$$

2.2.3 Peculiarità della tecnica

Le caratteristiche principali del metodo:

- è un approccio **non parametrico** per la costruzione di modelli di classificazione: non richiede alcuna assunzione sulla distribuzione della classe predetta e altri attributi;
- esistono algoritmi per farlo in tempi ragionevoli;
- una volta costruito, la classificazione di una nuova unità è una cosa estremamente veloce
- alberi decisionali di dimensioni contenuti sono relativamente facili da interpretare
- l'accuratezza degli alberi è spesso comparabile a quella di altri metodi
- sono metodi robusti alla presenza di *noise* e *attributi ridondanti*[2]
- l'approccio di partizionamento ricorsivo fa si che il numero di record diventi via via minore mano a mano che ci si sposta verso il basso nell'albero (**data fragmentation**).
  Ad un certo punto il numero di record potrebbe0 essere ragionevolmente considerato troppo piccolo per procedere oltre: non ha senso creare un albero da applicare su nuove osservazioni quando, arrivati ad un determinato punto, abbiamo poche informazioni per costruirlo.
  Una possibile soluzione è di impedire ulteriori splitting quando il numero di record cade al di sotto di una determinata soglia
- si sono considerati splitting univariati basati, appunto, solo su una variabile presa alla volta; può essere che a volte ciò non sia sufficiente/efficace come avviene in figura.
  Si possono comunque implementare criteri che riguardino pèiù parametri alla volta:
  - *alberi di decisione obliqui* permettono di superare la limitazione: nell'esempio i dati potrebbero essere splittati perfettamente mediante un singolo nodo con test
    $$x + y < 1$$
    Questo approccio ha il limite di essere computazionalmente più esoso rispetto ad una classificazione univariata
  - *costructive induction* è un approccio alternativo che si fonda sul creare degli attributi compositi a partire dagli attributi e aggiungerli al pool degli attributi disponibili, per poi procedere in maniera univariata tradizionale.
    Più efficiente dell'albero di decisione obliquo, ma il rischio di questo è di creare ridondanza.
- letteratura ha mostrato che la scelta della misura di impurità ha un effetto limitato sulla performace di un albero di decisione, mentre impatto maggiore ha la strategia utilizzata per la potatura dell'albero

[2] Se due variabili sono molto correlate tra loro ne verrà scelta una e molto probabilmente l'altra verrà ignorata perché non porta un beneficio



2.2.4 Overfitting e pruning negli alberi decisionali

Nello specifico caso degli *alberi decisionali* overfitting si può avere se l'albero diviene troppo grande, complesso e basato alle foglie su pochi casi che riproduce sempre più perfettamente il training set ma su un validation set la performance va progressivamente degradandosi.
Nello specifico contesto degli alberi decisionali vi suono due strategie per evitare overfitting il pre-pruning e il post-pruning

Pre-pruning Consiste nello stoppare l'algoritmo di crescita dell'albero prima che riproduca perfettamente il training set; per fare ciò deve essere adottata una stopping rule più stringente (ad esempio stoppare quando il guadagno osservato sta al di sotto di una determinata soglia. Ciò evita che si generino alberi inutilmente complessi, posto che il problema si sposta sulla scelta della soglia ottimale (troppo alta fa si che il modello non si possa sviluppare e si stoppi quasi subito andando in *underfitting*, troppo bassa non pone rimedio al rischio dell'overfitting)

Post-pruning In questo approccio l'albero è inizialmente accresciuto sino alla sua dimensione massima, dopodiché si procede alla potatura, che avviene bottow up way. La potatura può essere effettuata rimpiazzando un subtree (quindi una intera zona) con, alternativamente:

- un nodo foglia con classe determinata in base alla maggioranza dei record

- il subbranch più utilizzato di questa parte di albero

Il postpruning viene terminato tende a dare risultati migliori del prepruning perché si basano le scelte su un albero intero mentre nel prepruning può essere che si termini precocemente la crescita dell'albero

2.3 k-nearest-neighbor classifier

Un approccio differente dal decision tree è quello adottato dal classificatore k-nearest neighbor, che classifica utilizzando direttamente i dati del training set senza passare da una fase di costruzione di un modello di classificazione.

2.3.1 Implementazione

L'idea:

1. è trovare i k oggetti del training set che sono abbastanza simili (i nearest neighbors, appunto) a quello da classificare.
   La ricerca si basa su metriche di distanza viste in precedenza tra l'oggetto da classificare e tutti gli item del training set, scegliendo i k più vicini;

2. dopodiché si procede a utilizzare la classe dei k-nearest-neighbor per classificare, adottando alternativamente un *sistema tipo voto* (la classe assegnata all'oggetto da classificare è quella più frequente dei suoi neighbor, nel caso ci siano dei pareggi si può estrarre casualmente tra le classi più frequenti) o *pesando il voto* in base (inversamente proporzionali, del tipo $w_i = 1/d(\mathbf{x}'/\mathbf{x}_i)^2$) alla distanza.

Aspetto cruciale del metodo è la scelta di k:

- se troppo piccolo il classificatore potrebbe esser suscettibile ad overfitting a causa di noise nel training set

- se invece k è troppo grande si potrebbe andare incontro a misclassificazione poiché si fa uso di punto molto lontani

2.3.2 Caratteristiche

Le peculiarità :

- presenta aspetti di flessibilità superiore rispetto ad un albero decisionale che propone dei cutoff lineari

- si richiano però cattive performance se non si sceglie la misura di prossimità più adeguata (ad esempio se la scala della misurazione non è considerata)

2.4 Classificatori Bayesiani

Questo è la prima famiglia di classificatori che vediamo che iniziano ad effettuare classificazione esplicitamente sulla base di probabilità e nello specifico

applicando il teorema di Bayes

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)} \tag{2.5}$$

o, nella sua versione multivariata in cui $\mathbf{X}$ è l'insieme di attributi degli oggetti mentre Y è la variabile di classificazione:

$$P(Y|\mathbf{X}) = \frac{P(\mathbf{X}|Y) \cdot P(Y)}{P(\mathbf{X})} \tag{2.6}$$

dove:

- $P(Y)$ è la **prior** probability di una determinata classe
- $P(Y|\mathbf{X})$ è la **posterior** probability di Y (dato un determinato set di covariate $\mathbf{X}$)
- $P(\mathbf{X}|Y)$ è detta **class conditional probability**
- $P(\mathbf{X})$ è detta **evidence**

Nella fase di training:

- dobbiamo modellare le probabilità a posteriori per ogni combinazione di $\mathbf{X}$ e Y sulla base del training set.
  Lo step più impegnativo della formula è la stima della class conditional probability;
- sulla base delle probabilità a posteriori stimate un record del test set avente solo $\mathbf{X}$ potrà esser classificato trovando quale classe di Y massimizza la probabilità a posteriori.
  Di fatto, poi, dato che il denominatore è sempre costante di fatto può essere ignorato (e si classifica solo sulla base del valore maggiore del numeratore).

2.4.1 Naive Bayes classifier

Un classificatore naive Bayes stima la class conditional probability ipotizzando che gli attributi $\mathbf{X}$ siano condizionalmente indipendenti data la class label y.

2.4.1.1 Costruzione del modello

Ipotizzando che il set di attributi $\mathbf{X} = \{X_1, X_2, \ldots, X_d\}$ si componga di $d$attributi

$$P(\mathbf{X}|Y = y) = \prod_{i=1}^{d} P(X_i|Y = y) \tag{2.7}$$

Pertanto dobbiamo stimare solamente la probabilità condizionata di ogni X_i dato Y:

- per attributi categorici $P(X_i = x_i|Y = y)$ è semplicemente la frazione di item nella classe y un particolare valore x_i

- per attributi continui, o si discretizza e si procede come al precedente punto o alternativamente si assume che tali valori probengano da una determinata distribuzione di probabilità e si stimano i parametri della stessa utilizzandoi dati di training.
  In questo secondo caso spesso si sceglie la Gaussiana e ci si preoccupa di stimare media e varianza della distribuzione per ogni sottogruppo determinato dalla attributo di classe. Dopodiché di fatto si usa *come* probabilità la densità[3] della curva (del gruppo considerato) nel punto di interesse (dato dal valore assunto dalla variabile quantitativa).

Un potenziale problema è che se la class conditional probability per un attributo è zero la probabilità a posteriori della classe svanisce; stimare la prima sulla base delle frequenze può essere un metodo fragile, soprattutto se si dispone di un training set risicato o sbilanciato (senza molti valori di classe rappresentati).
Il problema può essere affrontato utilizzando l' **m-estimate approach**, che applica:

$$P(x_i|y_j) = \frac{n_c + mp}{n + m}$$

dove:

- n è il numero di casi nella classe y_j
- n_c è il numero di casi nella classe y_j che assumono il valore x_i
- p è un altro parametro da specificare che può esser visto come una prior probability di osservare l'attributo con valore x_i tra quelli della calsse y_j
- m è un parametro da fornire detto *equivalent sample size* e determina il trade off tra la prior probability p è l'observed probability n_c/n.

In generale uesto approccio fornisce un approccio più robusto per stimare probabilità quando il training set è piccolo.

2.4.1.2 Utilizzo

Una volta costruito il modello, per classificare un record il classificatore naive bayesiano calcola la probabilità a posteriori per ogni classe di Y come

$$P(Y|\mathbf{X}) = \frac{P(Y) \cdot \prod_{i=1}^{d} P(X_i|Y)}{P(\mathbf{X})} \tag{2.8}$$

2.4.1.3 Caratteristiche

Il metodo

- è robusto a noise e attributi irrilevanti
- la sua performance degrada in presenza di attributi correlati

2.4.2 Bayesian Belief network

`TODO`

[3] che comunque è proporzionale alla probabilità nell'intorno

2.5 Artificial Neural Network

TODO

2.6 Support Vector Machine

TODO

2.7 Ensemble Methods

Un metodo ensemble si fonda un un insieme di classificatori base, sviluppati sul training set, tutti impiegati per la predizione della calsse di una nuova osservazione; il metodo effettua la classificazione utilizzando le classificazioni proposte dai vari metodi base come un voto sulla possibile classe.
Questi metodi spesso migliorano l'accuratezza di classificazione rispetto all'utilizzo di un classificatore isolato.

2.7.1 Rationale per i metodi ensemble

Perché la performance aumenta utilizzando un metodo ensemble? Vediamolo attraverso un esempio. Ipotizziamo che un metodo ensemble si fondi su 25 classificatori binari, ognuno di essi caratterizzati da un rate di errore $\epsilon = 0.35$, sulla base di un sistema voti. Possiamo avere due casi estremi:

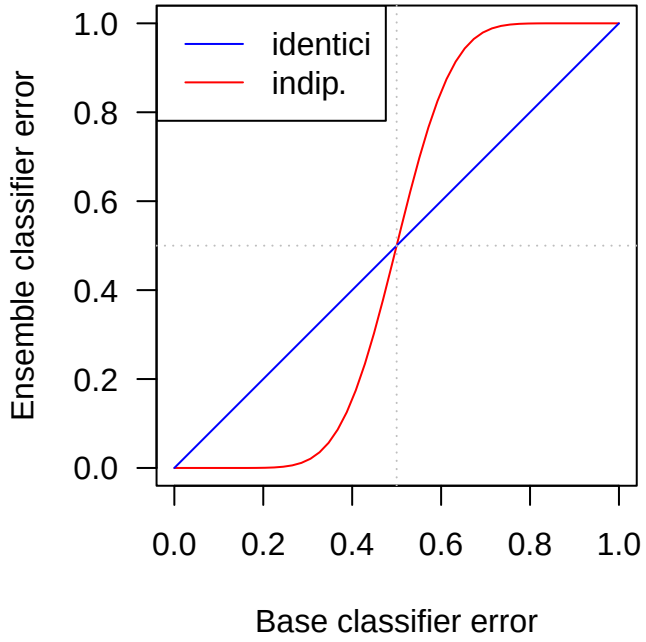
- i 25 classificatori sono identici: allora il metodo ensemble sbaglierà anch'esso nell'ϵ dei casi

- se invece i classificatori sono indipendenti l'ensemble farà una predizione erronea solo se più della metà dei base classifiers sbaglia, e nel nostro caso questo avviene nello

$$\sum_{i=13}^{2} 5\binom{25}{i}\epsilon^i(1-\epsilon)^{25-i} = 0.06$$

  dei casi, che è molto meno del singolo classifier

Generalizzando l'error rate di un classificatore ensemble basato su 25 classificatori base (identici o indipendenti) si trova nella figura: si nota che in caso di classificatori base non identici, l'ensemble performa peggio che il singolo classificatore base quando $\epsilon > 0.5$.



Sono pertanto due le condizioni necessarie per avere un beneficio nell'uso di classificatori ensemble:

- i classificatori base siano il più indipendenti possibili
- il classificatore base dovrebbero fare meglio di un classificatore che predice effettuando tiri a caso (cioè dovrebbe essere $\epsilon < 0.5$)

2.7.2 Metodi per costruire un classificatore ensemble

I metodi ensemble posono essere costruiti in vario modo, approfondiamo due approcci:

- manipolando il training set: molteplici training set (quindi selezione di oggetti) sono creati attraverso ricampionamento; un classificatore è costruito da ciascuno dei training set. *Bagging* e *boosting* sono due esempi di metodi che manipolano il training set passatogli
- manipolando le feature utilizzate: in questo approccio, un subset delle feature è scelto per formare ogni training set. *Random forest* è un esempio che adotta questo approccio e usa gli alberi decisionali come classificatore di base

2.7.2.1 Bagging

Conosciuto anche come aggregazione di bootstrap, consiste nel ripetere il seguente procedimento k volte:

- campionare (con reimmissione) da un dataset con distribuzione uniforme per creare un campione bootstrap avente la stessa dimensione del dataset originale

- si costruisce un classificatore sulla base del campione bootstrap

Una volta che si hanno i k classificatori, gli elementi del test set vengono classificati da ciascuno di essi e complessivamente la procedura restituisce come classificazione quella che ha ricevuto più voti.
Il bagging è in particolar modo utile quando si hanno classificatori di base sensibili al training sample (noise).

2.7.2.2 Random forest

Le random forest combinano la predizione fatta da molteplici decision tree, ognuno di essi generato sulla base di un set indipendente di attributi. E' un approccio che non risente della dimensionalità del dataset e della presenza di attributi molto correlati tra loro.
L'algoritmo è come segue:

- si seleziona un insieme F di attributi casualmente
- sulla base di questi viene creato un albero di decisione

Una volta che gli alberi sono costruiti le predizioni sono combinate utilizzando uno schema di voto.

Capitolo 3

Association analysis

La estrazione di regolarità (pattern) frequenti in un dataset è un problema rilevante nel data mining. Qui ci si concentra soprattutto sugli insiemi di articoli presenti in una transazione commerciale[1], tipologia di dato che può essere rappresentato da tabella 3.1, e memorizzato come una serie di attributi dicotomici[2].

Consideriamo una frase "Nel 90% delle transazioni in cui si acquistano pane e burro si acquista anche latte". Questa affermazione può essere formalizzata come una regola

$$\{\text{Pane, Burro}\} \rightarrow \{\text{Latte}\}$$

composta da un *antecedente* ("si acquista pane e burro") e un *conseguente* ("si acquista latte"), nonchè caratterizzata da una *confidenza* (espressa come percentuale).

Finalità dell'analisi, qui, è progettare/impiegare algoritmi che estraggano insiemi di regole a partire da basket data[3]

[1] La diffusione della tecnologia dei codici a barre (anni 90) ha permesso la memorizzazione di enormi quantità di basket data, cioè dei dati di acquisto dei prodotti ogni transazione.
La gestione del supermercato comporta decisioni che possono essere rese molto piú efficaci e tempestive se si analizza l'associazione degli articoli nei basket data. Molte organizzazioni possiedono enormi quantità di dati di questo tipo che non vengono memorizzati in basi di dati perché la tecnologia tradizionale dei DBMS non consente di estrarre le informazioni in essi contenute in modo efficiente

[2] Si parte con una descrizione semplificata; metodi per la gestione di altri aspetti quali quantità di un item acquistato o prezzo di acquisto sono trattati nel capitolo 7 del Tan.

[3] Esempi di applicazioni possono essere:

- trovare tutte le regole aventi "Diet Coke" come conseguente: tali regole potrebbero suggerire una strategia per incrementare le vendite della Diet Coke, ad esempio volta a rendere piú probabile il verificarsi delle condizioni che costituiscono le premesse delle regole

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Tabella 3.1: Esempio di market basket data

3.1 Definizione del problema

3.1.1 Definizioni

Per inquadrare teoricamente il problema:

- sia $I = \{i_1, i_2, \ldots, i_d\}$ l'insieme di tutti gli item (ad esempio oggetti o articoli gestititi da un supermercato)
- sia $T = t_1, t_2, \ldots, t_N$ l'insieme delle N transazione
- ogni transazione è un sottoinsieme degli item considerati: $t_i \subseteq I$
- definiamo con il termine *itemset* una generica collezione di articoli; se un itemset contiene k è chiamato k-itemset[4];
- il *supporto* di una itemset X consiste nel numero [5] di transazioni che contengono un determinato itemset:

$$\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in T\}| \tag{3.1}$$

- una *regola associativa* è una implicazione nella forma $X \to Y$ dove $X, Y \subseteq I$, sono insiemi di item disgiunti, $X \cap Y = \emptyset$; X è detto itemset dell'antecedente, Y itemset della conseguente.

3.1.2 Supporto e confidenza

La forza di una regola di associazione può essere misurata in termini del suo supporto e della sua confidenza:

- **supporto**: sintetizza la probabilità con cui l'itemset formato dall'unione di antecedente e susseguente è presente nell'insieme delle transazioni analizzato. Formalizzato come:

$$s(X \to Y) = \frac{\sigma(X \cup Y)}{N} \tag{3.2}$$

  Una regola che supporto limitato potrebbeessere occorsa semplicemente per caso; anche se così non fosse, poi, è spesso di poco interesse (troppo particolare).

- **confidenza**: determina la probabilità di presenza dell'itemset della conseguente nelle transazioni che includono l'itemset dell'antecedente. Formalizzata come:

$$c(X \to Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{3.3}$$

  La confidenza misura la forza dell'associazione tra X e Y.

  - regole aventi "sottaceti" nell'antecedente: tali regole potrebbero suggerire quali prodotti sarebbero penalizzati se si decidesse di cessare la vendita dei sottaceti.
  - regole che mettono in relazione prodotti sullo scaffale A e prodotti sullo scaffale B: tali regole potrebbero suggerire una differente disposizione dei prodotti sugli scaffali
  - trovare le "migliori" k regole aventi "sottaceti" come conseguente: il criterio di qualità è basato sulla confidenza delle regole, o del numero di transazioni in cui la regola è verificata

[4] Ad esempio {Beer, Diapers, Milk} è un esempio di un 3-itemset

[5] Ad esempio {Beer, Diapers, Milk} ha supporto pari a 2 nel dataset di tabella 3.1

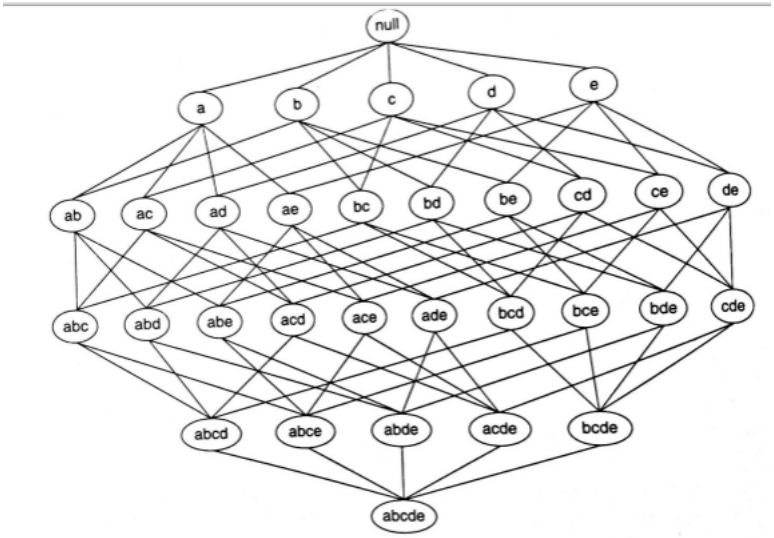


Figura 3.1: Struttura a reticolo per itemset

3.1.3 Focus e fasi del problema di mining

Dato un set di transazioni T siamo interessati a identificare tutte le **regole forti**, ovvero che abbiano al contempo un supporto non inferiore ad una determinata soglia *minsup* e una confidenza non inferiore a un'altra soglia *minconf*.
Un **approccio di forza bruta**, ovvero calcolare supporto e confidenza entro un set di transazioni T per ogni possibile regola definibile sulla base di un insieme di item I, sarebbe computazionalmente proibiltivo.
La **strategia** adottata per ottimizzare la ricerca delle regole segue due step:

1. selezionare innanzitutto gli itemset che rispettino il supporto (*frequent itemset generation*) ...
2. ... per calcolare solo dopo la confidenza e selezionare le regole (*rule generation*).

La prima delle due fasi è quella computazionalmente più pesante; tecniche per entrambe le fasi sono presentate nelle due sezioni a seguire.

3.2 Identificazione degli itemset frequenti

Una struttura a reticolo pò essere impiegata per listare tutti i possibili itemset derivabili da un insieme I (nell'esempio in tabella 3.1, $I = \{a, b, c, d, e\}$).
In generale un dataset che contega k item può generare potenzialmente sino a $2^k - 1$ itemset frequenti (escludendo l'insieme vuoto); un approccio di forza bruta per trovare gli itemset frequenti richiederebbe di comparare ogni candidato con tutte le transazioni considerate, approccio computazionalmente costoso.
Vi sono diversi modi per ridurre la comlessità computazionale di questo step:

- ridurre il numero di itemset candidati: il *principio Apriori* è un modo efficace di rimuovere itemset senza contare il loro supporto

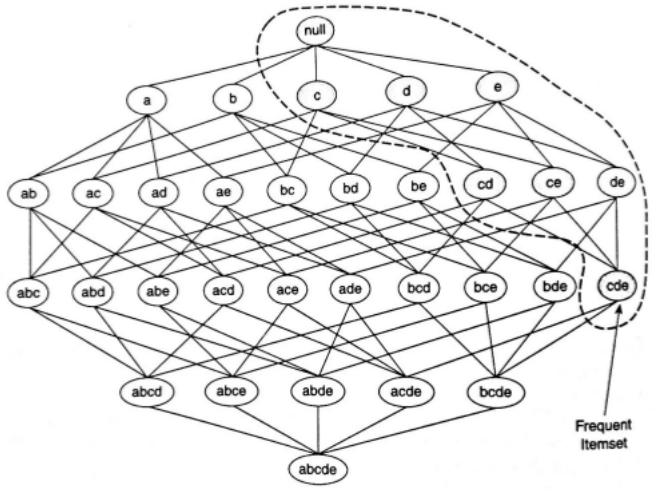


Figura 3.2: Se {c, d, e} è frequente, allora lo sono anche tutti i suoi subset.

- ridurre il numero di comparazioni: invece di comparare ogni itemset candidato con ogni transazione si utilizzano strutture dati avanzati alternativamente per memorizzare gli itemset candidati o per comprimere il dataset delle transazioni (*algoritmo FP-tree growth*)

3.2.1 Principio Apriori

Il principio stabilisce che:

- se un itemset è frequente (cioè supera una determinata soglia di supporto), banalmente saranno frequenti anche i suoi sottoinsiemi (fig 3.2) ;
- se invece un itemset è infrequente, allora tutti i suoi superset sono anch'essi infrequenti (fig 3.3).
  L'intera struttura reticolare non frequente può essere potata dagli itemset analizzati

Il buon esito del tutto è garantito dalla proprietà anti-monotona: il supporto di un itemset non è mai superiore a quello dei suoi subset.

3.2.2 Generazione dei candidati

Un esempio di generazione degli itemset frequenti con Apriori che faccia uso dei dati di tabella 3.1 è riportato graficamente in figura 3.4:

- assumiamo che la soglia di supporto minimo sia 60%, corrispondente ad un conteggio di 3 (su 5 transazione della tabella);
- si inizia con gli itemset formati da 1 solo item: dopo aver contato il supporto {Cola} ed {Eggs} sono scartati perché appaiono in meno di 3 transazioni
- al secondo step vengono generati gli 2-itemset candidati basati sugli 1-itemset frequenti; si effettua il conteggio, si screma e si itera ulteriormente

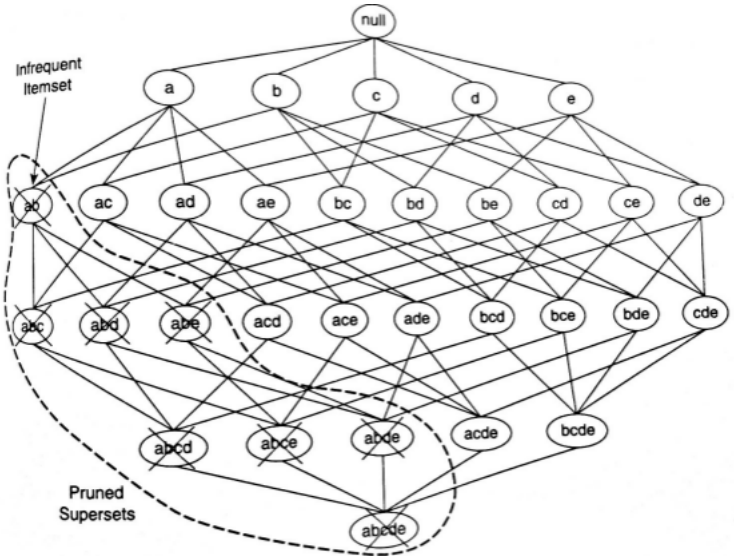


Figura 3.3: Se {a, b} è infrequente, allora lo sono anche tutti i suoi supset.

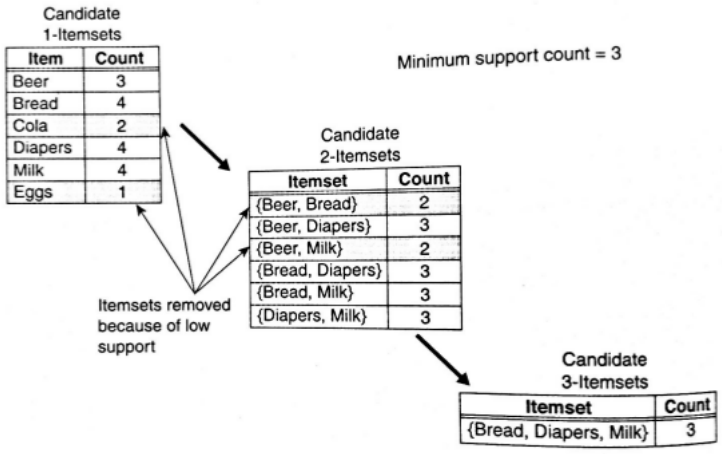


Figura 3.4: Generazione degli itemset frequenti con Apriori.

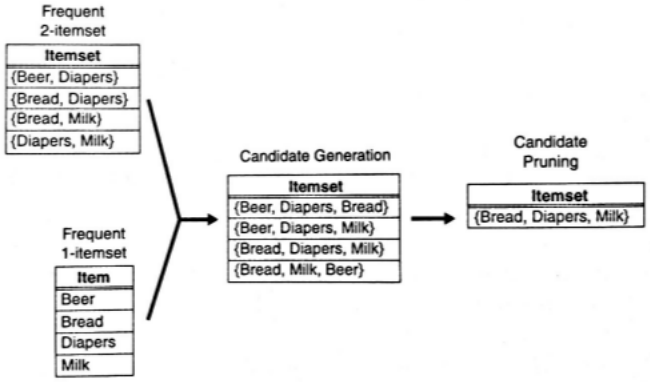


Figura 3.5: Generazione di candidati 3-itemset mediante $F_{k-1} \times F_1$

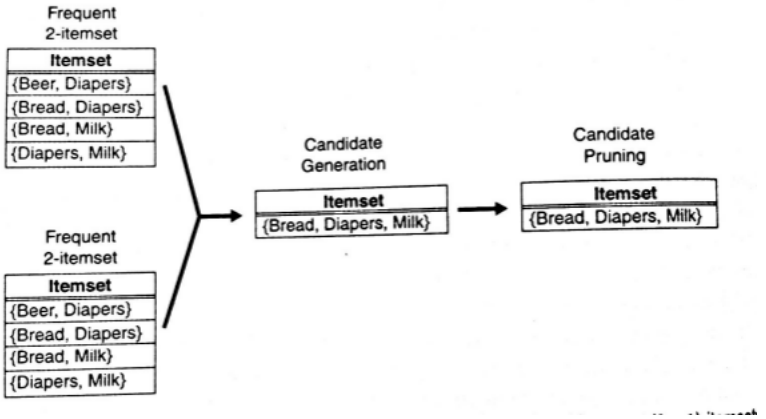


Figura 3.6: Generazione di candidati 3-itemset mediante $F_{k-1} \times F_{k-1}$

Per generalizzare denotiamo con C_k gli itemset *candidati* composti di k elementi, e con F_k gli itemset *frequenti* composti di k elementi (ovvero che superano la soglia di supporto prefissata). Apriori ricerca iterativamente per livelli (algoritmo level-wise):

- al primo passo viene generato F_1 tra gli C_1 usando *minsup*;
- al k-esimo passo vengono generati i candidati C_k utilizzando F_{k-1};

La generazione degli itemset candidati avviene in due passi: il passo di *join* e il passo *prune* (dove l'algoritmo a priori effettua una prima scrematura precedente alla fase di conteggio e determinazione degli itemset frequenti).

Generazione dei candidati: passo di joim Vi sono diversi metodi atti a generare C_k sulla base di F_{k-1}:

- il metodo $F_{k-1} \times F_1$ crea il prodotto cartesiano tra i $(k-1)$-itemset frequenti e gli 1-itemset frequenti (figura 3.5); questo approccio però non esclude la generazione di candidati duplicati

Tid	Items
1	a b c
2	a b d
3	b c d
4	b c
5	a c e f
6	c d
7	b c e
8	c d e f
9	c d f
10	a b c e

Tabella 3.2: Un esempio di dataset

- il metodo $F_{k-1} \times F_{k-1}$: l'algoritmo richiede l'ordinamento lessicografico degli item entro itemset, ed effettua il merge di una coppia di $(k-1)$-itemset frequenti solo se i primo $k-2$ item sono identici (figura 3.6). L'utilizzo dell'ordinamento lessicografico impedisce la generazione di duplicati.

Generazione dei candidati: passo di prune Il pruning dei candidati riduce C_k usando il principio Apriori: per ogni $X \in C_k$, se X ha un subset di cardinalità $k-1$ che non è frequente, nemmeno X è frequente e si elimina da C_k.
Si noti che la verifica della frequenza del subset è eseguibile efficientemente senza accedere al data base, infatti $k-1$ itemset frequenti sono in F_{k-1} che è già calcolato

3.2.3 Individuazione degli itemset frequenti

Gli itemset rimasti dopo la fase di prune nella generazione dei candidati costituiscono C_k. Per concludere la ricerca, poi, la determinazione gli itemset frequenti avviene mediante verifica della cardinalità attraverso accessi al data base; gli itemset con cardinalità inferiore alla soglia *minsup* sono eliminati, quelli che rimangono costituiscono F_k.

3.3 Algoritmo FP growth

Lalgoritmo prende una strada differente rispetto all'apriori nel trovare gli itemset frequenti, facendo uso di una struttura dati chiamata FP-tree nel quale memorizzare inizialmente il set delle transazioni.

3.3.1 Rappresentazione dell'FP-tree

Ipotizziamo di avere un set di transazioni con item ordinati al proprio interno come in tabella 3.2.
Un FP tree è una rappresentazione compressa dei dati sulle transazioni; queste vengono lette una alla volta e mappate in un path di un albero con nodi che corrispondono ad item e associato un conteggio rappresentante quante volte si è "passati" da quel nodo.

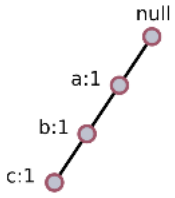


Figura 3.7: L'albero in al termine della prima transazione

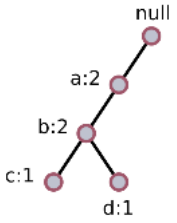


Figura 3.8: L'albero al termine della seconda transazione

Figura 3.7 e 3.8 mostrano la costruzione dell'albero in seguito alla lettura delle prime due transazioni, 3.10 l'albero completo e 3.9 l'albero completo una volta che sono stati creati puntatori tra nodi con lo stesso item (per ragioni di efficienza durante la generazione degli itemset frequenti.

3.3.2 Identificazione degli itemset frequenti

FP-growth è un algoritmo che genera gli itemset frequenti a partire da un FP-tree, esplorando l'albero dal basso all'alto e nello specifico per ogni item $f, e, d, \ldots, a$ (ovvero in ordine lessicografico inverso rispetto all'ordinamento entro transazione) si estraggono i percorsi che terminano con l'item. Questi sono detti percorsi prefisso; una rappresentazione dei percorsi prefisso si trova in 3.11.

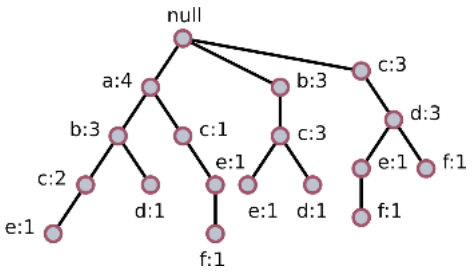


Figura 3.9: L'albero completo

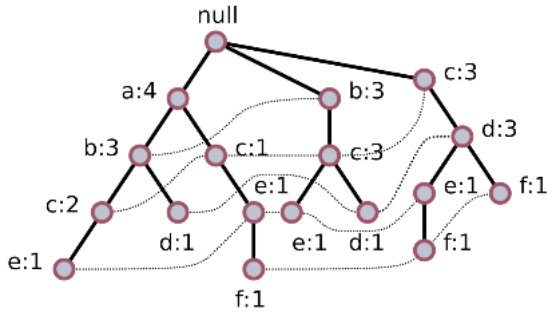


Figura 3.10: L'albero completo con puntatori di efficienza

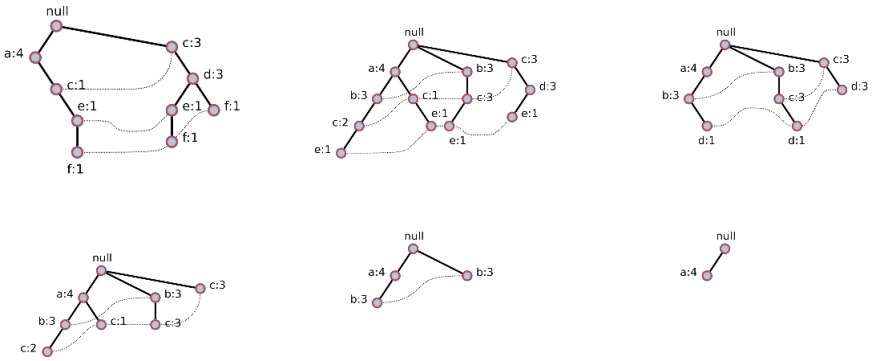


Figura 3.11: Percorsi prefisso

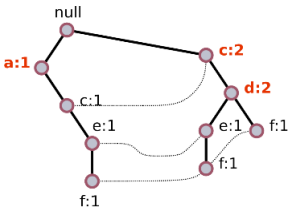


Figura 3.12: Fase 1 fp tree condizionale

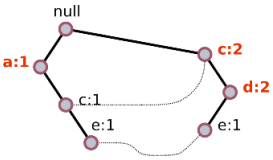


Figura 3.13: Fase 2 fp tree condizionale

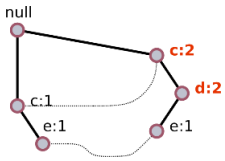


Figura 3.14: Fase 3 fp tree condizionale

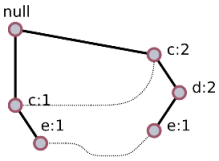


Figura 3.15: FP-tree condizionale di f

Si considerano i vari percorsi prefisso a turno; nel seguito ci si focalizza sul percorso terminante in f. Ipotizziamo che il supporto minimo necessario per essere considerati frequenti sia 2.
Si procede come segue:

- si sommano i supporti assoluti di f utilizzando i puntatori:
- in questo caso l'itemset $\{f\}$ è dichiarato frequente perché supporto per f è 3 che è superiore alla soglia minima (2)
- dato che $\{f\}$, l'algoritmo deve risolvere il sottoproblema di trovare gli itemset frequenti che terminano in ef, df, cf, af.
  Per farlo viene generato un cd **FP-tree condizionale**; a partire dal percorso prefisso di f:

  1. è necessario correggere i supporti lungo i percorsi eliminando i contributi delle transazioni che non contengono f (figura 3.12)
  2. eliminare i nodi f (figura 3.13)
  3. eliminare i nodi con supporto insufficiente; ad esempio a ha supporto pari a 1, quindi compare in una sola transazione insieme a f. Pertanto qualsiasi transazione $t_i \supseteq af$ è non frequente e l'item a può essere eliminato dall'elaborazione 3.14

Una volta ottenuto l'FP-tree condizionale per f (figura 3.15) possiamo risolvere i sottoproblemi per ef, df, cf, af.
Considerando ef:

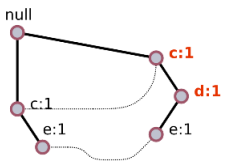


Figura 3.16

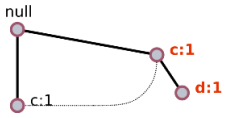


Figura 3.17

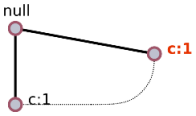


Figura 3.18

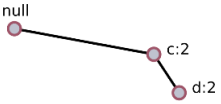


Figura 3.19

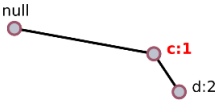


Figura 3.20

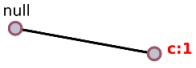


Figura 3.21

Figura 3.22

- troviamo i percorsi prefisso di e nel FP-tree condizionale per f
- sommiamo i supporti assoluti di e utilizzando i puntatori; in questo caso il supporto di e è 2 quindi $\{e, f\}$ è anch'esso frequente
- si procede ricorsivamente risolvendo i sottoproblemi per def e cef: si costruisce un FP-tree condizionale per ef:
  1. si parte correggendo i supporti assoluti (figura 3.16)
  2. si elimina e (figura 3.17)
  3. d non ha supporto assoluto sufficiente ed è eliminato; ogni transazione $t_i \supseteq def$ è non frequente. Il restante albero ha un solo item con somma dei supporti 2, quindi cef è frequente (figura 3.18).

Terminata la ricerca degli itemset frequenti che terminano in ef , ricerca gli itemset frequenti che terminano in df:

- si rova i percorsi prefisso di d nel FP-tree condizionale per f (figura 3.19)
- supporto di d è 2, quindi df è frequente
- si passa a risolvere i sottoproblemi per cdf, bdf, adf; si genera un FP-tree condizionale per df e si applicano
  - si corregge `c:2` in `c:1` (figura 3.20)
  - si elimina d (figura 3.21)
  - si elimina c per supporto insufficiente e termina l'elaborazione per df (figura 3.22)

I restanti sottoproblemi di ricerca degli itemset frequenti che terminano in cf, bf, af non devono essere trattati perché non esistono corrispondenti percorsi prefisso.

3.4 Generazione delle regole

Infine, una volta individuati gli itemset frequenti, la generazione delle regole per ciascuno di essi avviene come segue:

- una regola associativa piò essere ricavata partizionandol'itemset Y in due sottoinsiemi non vuoti, X e $Y \setminus X$ tale che la regola

$$X \to Y \setminus X \tag{3.4}$$

  Ogni k-itemset frequente può produrre sino a $2^k - 2$ regole di associazione (ignorando le regole che hanno antecedenti o conseguenti vuote).

- se la soglia di confidenza minima è rispettata la regola è presa per buona, altrimenti viene scartata.

Si noti che calcolare la confidenza di una data regola non richiede scan addizionali del dataset delle transazioni: ad esempio sia $X = \{1, 2, 3\}$; considerando la regola $\{1, 2\} \to \{3\}$ da esso generata, la confidenza di questa regola è $\sigma(\{1, 2, 3\})/\sigma(\{1, 2\})$. Dato che $\{1, 2, 3\}$ è frequente, anche $\{1, 2\}$ lo sarà il conteggio di supporto per entrambi gli itemset è già disponibile dalla fase di generazione degli itemset frequenti.

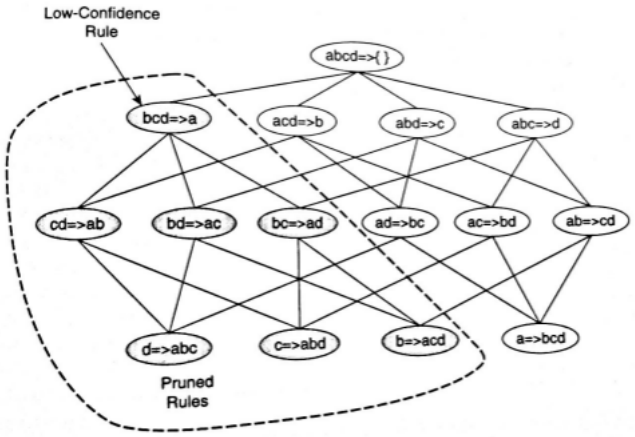


Figura 3.23: Pruning di regole di associazione utilizzando misure di confidenza

L'ottimizzazione svolta nella generazione di regole dell'Apriori L'algoritmo apriori effettua una ultima ottimizzazione nel pruning delle regole, che si basa sul seguente risultato: se una regola $X \to Y \setminus X$ non soddisfa la soglia di confidenza non lo farà neanche qualsiasi regola $\widetilde{X} \to Y \setminus \widetilde{X}$ con $\widetilde{X} \subset X$.[6] Nello specifico:

- apriori parte estraendo le regole che hanno alta confidenza tra quelle che hanno una variabile nella conseguente
- queste sono utilizzate per generare nuove regole candidate; ad esempio se $\{acd\} \to \{b\}$ e $\{abd\} \to \{c\}$ hanno elevata confidenza, dal loro merge viene generata la regola $\{ad\} \to \{bc\}$ da testare

In figura 3.23 è mostrata una struttura reticolare per le regole di associazione generate dall'itemset (frequente) $\{a, b, c, d\}$; qualora qualsiasi nodo abbia bassa confidenza (nell'esempio $\{bcd\} \to \{a\}$, l'intero subgrafo che deriva dal nodo (formato dalle regole che hanno a nella conseguente) può essere potato e le regole ignorate dalla conta.

[6] Per dimostrarlo, se le confidenza delle regole sono $\sigma(Y)/\sigma(X)$ e $\sigma(Y)/\sigma(\widetilde{X})$, dato che $\widetilde{X} \subset X \implies \sigma(\widetilde{X}) \geq \sigma(X)$. Pertanto la regola con $\widetilde{X}$ non può avere una confidenza maggiore di quella con X.

Capitolo 4

Clustering

4.1 Introduzione

Nella letteratura la definizione di cluster è per lo più informale: secondo alcuni la definizione stessa di cluster sarebbe problematica. Alcune definizioni:

- Everitt: data una collezione di N oggetti ciascuno descritto da un insieme di p variabili, derivare una divisione utile in un certo numero di classi;
- Tan: clustering è la divisione dei dati in gruppi (cluster) significativi o utili.

A differenza dei problemi di classificazione dove si dispone sia di $\mathbf{x}$ (il set di covariate) che y (la classe nel dataset di training), nel clustering quest'ultima viene determinata solamente sulla base delle $\mathbf{x}$; per questo motivo il clustering è una tecnica di *classificazione non supervisionata*.

Proprietà generali desiderabili Per un algoritmo di clustering:

- scalabilità: efficienza quando le osservazioni e/o gli attributi sono molti
- detection di cluster di forma arbitraria (non solo globulare)
- necessità di pochi parametri di input
- insensibilità noise e/o ordine con cui si forniscono i dati
- riconoscimento e robustezza ad outlier
- supporto di dati con domini diversi (intero, float, stringa, ecc)

Finalità Il clustering viene prevalentemente intrapreso per scopi:

- **conoscitivi**: capire la struttura dei dati (ad esempio classificare i diversi tipi di soggetti aventi una data malattia)
- **strumentali**: cluster analysis usata come strumento di preprocessing

4.1.1 Tipi di clustering

Gli algoritmi di clustering possono essere classificati in base a differenti aspetti dei cluster che creano (*nestedness*, *separazione* e *completezza*) o sulla definizione di cluster che adottano.

Cluster gerarchici e partizionali La classificazione attiene alla nestedness dei cluster creati;

- *gerarchico*: i cluster formano tra di essi una gerarchia rappresentabile mediante un albero, nel senso che cluster (nodi) di dimensioni più grandi possono essere suddivisi in cluster di dimensioni più piccole (subcluster). Spesso, ma non necessariamente, i cluster/nodi foglia sono singleton, ovvero insiemi di un solo elemento;

- *partizionali* o *flat*: l'insieme di cluster è non nested, e il clustering è una semplice divisione dell'insieme di dati in subset che non overlappano.

Cluster esclusivi, overlapping e fuzzy La classificazione attiene alla separazione dei cluster creati;

- *esclusivo*: ogni oggetto è elemento di esattamente un cluster;

- *overlapping*: un oggetto può essere elemento di piú di un cluster;

- *fuzzy*: per ogni cluster viene costruita una funzione di appartenenza avente per dominio tutti gli oggetti e codominio $[0, 1]$.

Cluster completi e parziali La classificazione attiene alla completezza dei cluster creati;

- *completo*: ogni oggetto è elemento di qualche cluster;

- *parziale*: si ammette che un oggetto non appartenga ad alcun cluster. Tale oggetto può ritenersi rumore, o un outlier.

Definizione di cluster Gli algoritmi di clustering si possono classificare infine a seconda della definizione di cluster (figura 4.1) che viene adottata per creare i gruppi:

- **buona separazione**: un cluster è un insieme di oggetti che sono più vicini/simili agli altri oggetti entro il cluster che a qualsiasi altro oggetto al di fuori del cluster;

- cluster basato **su prototipo/centro**: un cluster è un insieme di oggetti più vicini/simili al prototipo che definisce il proprio cluster che a quello di qualsiasi altro cluster.
  Nel caso di variabili quantitative il prototipo è spesso un *centroide* (determinato dalla media di ciascun attributo degli oggetti che appartengono al cluster); nel caso di altri tipi di dati il prototipo è un *medoide* (il punto considerato più rappresentativo del cluster);

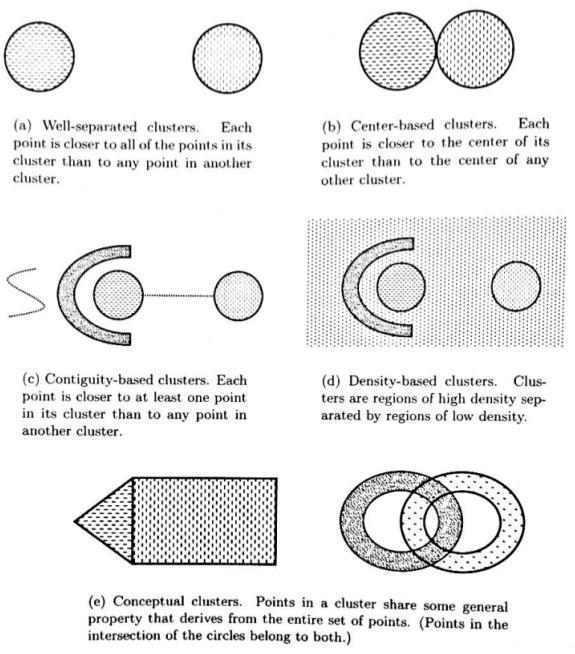


Figura 4.1: Differenti definizioni di cluster su un set di punti a due dimensioni

- cluster basato **su grafi**: se i dati sono rappresentabili come un grado dove differenti nodi sono gli oggetti e i collegamenti rappresentano connessioni tra oggetti, un cluster puiò essere definito dalle *componenti connesse*, ovvero un gruppo di oggetti che sono connessi ciascuno all'altro e non hanno connessioni con altri oggetti al di fuori del gruppo.
  Una variante importante sono i **cluster basati su contiguità**, ove due oggetti sono connessi sono se sono entro una certa distanza, e ciò implica che ogni oggetto in un cluster del genere è più vicino ad un oggetto del cluster che a qualsiasi punto di un altro cluster;

- cluster basati **su densità**: un cluster è una regione fitta di oggetti, circondata da una regione a bassa densità; una definizione del genere è spesso adottata quando i cluster sono irregolari (ad esempio non globulari), quando vi è noise o outliers;

- cluster basati **su proprietà condivisa**: più generalmente definiamo cluster come un insieme di oggetti che condividono una data proprietà. La definizione racchiude le altre come caso particolare ma permette anche nuovi tipi di cluster di essere definiti, una volta che è definita la proprietà da soddisfare. L'attività di individuare i cluster è poi detta *conceptual clustering* (ed è abbastanza vicina al pattern recognition).

4.2 Clustering basato su prototipo

Le tecniche di clustering basate su prototipo creano un partizionamento ad un livello degli oggetti: un cluster è un insieme di oggetti nel quale qualsiasi

oggetto è più vicino al prototipo che definisce il cluster che al prototipo di un altro cluster.
Due tra le tecniche più comuni sono:

- *K-means*: definisce un prototipo in termini di centroid, che solitamente è la media multivariata di un gruppo di punti (quindi può corrispondere ad alcuna unità del campione);

- *K-medoid*: definisce un prototipo in termini di un medoide, considerato il punto più rappresentativo di un gruppo (pertanto coincide con una osservazione effettiva). Di fatto questa metodologia può essere applicata ad un range ampio di situazioni, dato che richiede solo una misura di prossimità tra coppie di oggetti.

Approcci che espandono le possibilità applicative di questo questo filone:

- *fuzzy c-means*: agli oggetti è permesso di appartenere a più di un clustering e ogni appartenenza è caratterizzata da un peso. Questo rilassa il requirements che ogni oggetto appartenga a solo un cluster quando è ugualmente vicino a due prototipi;

- *clustering con mixture models*: un cluster è modellato come una distribuzione statistica: gli oggetti sono generati da un processo casuale che è caratterizzato da un tot di parametri delle distribuzioni;

- *self organizing maps*: nelle quali i cluster sono vincolati ad avere relazioni prefissate.

Nel prosieguo approfondiamo k-means e clustering basato su mixture models.

4.2.1 Algoritmi partitivi

4.2.1.1 Introduzione

K-means è un algoritmo partizionale/partitivo (esclusivo, completo). Questi tipi di algoritmi:

- generano una partizione del dataset $D = \{x_1, x_2, \ldots, x_N\}$ contenete esattamente K classi, con K parametro a scelta. L'*encoder* è una funzione a valori naturali:

  $$C : i \in \{1, \ldots, N\} \rightarrow C(i) \in \{1, \ldots, K\}$$

  che mappa (l'indice di) ogni oggetto in un naturale identificativo.

- ogni elemento della partizione deve essere non vuoto;

- ogni oggetto appartiene a esattamente una partizione;

- K è un parametro da passare all'algoritmo: se il numero di cluster non è noto, nell'applicazione sono necessarie esecuzioni per ogni valore di K in un intervallo.

4.2.1.2 Accuratezza della partizione

Le diverse possibili partizioni di cardinalità K possono differire molto per qualità; per andare alla ricerca della partizione ottimale, quello che si fa è codificare la qualità della partizione in una funzione per andare poi ad ottimizzare quest'ultima (nell'algoritmo o nella comparazione di esiti differenti).
La **loss function** è una funzione reale:

$$L : C \to L(C) \in \mathbb{R}$$

che mappa un encoder ad un numero che rappresenta l'inaccuratezza della partizione prodotta dall'encoder.
Il problema del clustering partizionale concettualmente consiste nel, determinato l'intero K e la funzione di perdita $L(\cdot)$, di trovare l'encoder C che minimizza la perdita L.

4.2.1.3 Funzioni di perdita

Le funzioni di perdita possono codificare/misurare omogeneità (somiglianza entro cluster) o separazione (diversità tra cluster). Ve ne sono diverse tra le quali vediamo:

- per *l'omogeneità*: la somma delle dissimilarità intra-cluster (*clique*) è definita da:
$$clique_{sum}(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d_{ii'}$$

- per la *separazione*: la somma delle dissimilarità inter-cluster (*sum of cuts*) definita come:
$$cut_{sum}(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')\neq k} d_{ii'}$$

Un risultato importante in merito a queste due è che la somma della clique e della dissimilarità inter-cluster è costante.

$$\begin{aligned} clique_{sum}(C) + cut_{sum}(C) &= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d_{ii'} + \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')\neq k} d_{ii'} \\ &= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\left(\sum_{C(i')=k} d_{ii'} + \sum_{C(i')\neq k} d_{ii'}\right) \\ &= \text{costante} \end{aligned}$$

Da ciò discende che:

- minimizzare la clique è equivalente a massimizzare la somma delle dissimilarità inter-cluster;

- sia la omogeneità che la separazione sono massimizzate simultaneamente.

Minimizzazione per dati vettoriali In uno spazio vettoriale con prodotto scalare, consideriamo la clique con distanze quadratiche. Se $\overline{x}_k$ denota la media del k-esimo cluster, in base all'encoder C:

$$\begin{aligned} W_{sq}(C) &= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d_{ii'}^2 \\ &= \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|x_i - \overline{x}_k\|^2 \end{aligned}$$

Questa è proprio la formula adottata da K-means (nel caso di dati quantitativi), il quale minimizza la somma delle deviazioni al quadrato dei cluster, pesata con la cardinalità del cluster

$$W_{kmeans}(C) = W_{sq}(C) = \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|x_i - \overline{x}_k\|^2$$

4.2.1.4 Funzioni obiettivo non solo quantitative

In ambito di **variabili quantitative** la metrica da considerare è l'SSE (sum of the squared error) definito come:

$$SSE = \sum_{i=1}^{K}\sum_{\mathbf{x}\in C_i} dist(\mathbf{c}_i, \mathbf{x})^2 \tag{4.1}$$

dove *dist* è la distanza euclidea (o la distanza Manhattan). In questo setting, può essere dimostrato che il centroide $\mathbf{c}_i$ che minimizza il SSE del cluster è la media, definita come:

$$\mathbf{c}_i = \frac{1}{m_i}\sum_{\mathbf{x}\in C_i} \mathbf{x}$$

Ad esempio il centroide di un cluster che contiene tre punti bidimensionali $(1,1)$, $(2,3)$ e $(6,2)$ è $((1+2+6)/3, (1+3+2)/3) = (3,2)$. Nel contrario della distanza manhattan il centroide è invece definito sula mediana.
Per **dati dicotomici** nella distanza cambia solo la misura di distanza; definiamo la

$$\text{TotalCohesion} = \sum_{i=1}^{K}\sum_{\mathbf{x}\in C_i} cosine(\mathbf{c}_i, \mathbf{x})^2 \tag{4.2}$$

Anche in questo caso il centroide è basato sulla media.

4.2.2 K-means

La notazione che ci sarà utile è riportata in tabella 4.1

4.2.2.1 Algoritmo base

l'algoritmo base segue i passi:

1. si scelgono K (numero di cluster desiderati) centroidi iniziali;

Simbolo	Descrizione
x	un oggetto
C_i	l'i-esimo cluster
$\mathbf{c}_i$	il centroide del cluster C_i
$\mathbf{c}$	il centroide di tutti i punti
m_i	il numero di oggetti nel cluster i-esimo
m	il numero di oggetti nel data set
K	il numero di cluster

Tabella 4.1: Notazione

2. ogni punto è assegnato al centroide ad esso più vicino: la misura di prossimità adottata è spesso la distanza euclidea (o la manhattan) per dati quantitativi, mentre la cosine similarity (o la distanza di Jaccard) è più appropriata per dati esclusivamente binari in generale;

3. il centroide di ogni cluster è aggiornato sulla base dei punti assegnatigli;

4. si ripetono il passo di assegnamento e aggiornamento sino a che alcun punto cambia cluster (o equivalentemente i centroidi non subiscono variazioni). Dato che la maggior parte della convergenza avviene nei primi step, spesso ci si limita a terminare l'algoritmo ad una condizione più *weak*, ad esempio quando solo l'1% pdei punti cambiano cluster.

Gli step 3 e 4 di K-means cercano di minimizzare SSE, anche se l'ottimo trovato è locale (dato che l'ottimizzazione di SSE è per determinate scelte di centroide, non tutti i possibili centroidi).

4.2.2.2 Scelta dei centroidi iniziali

La scelta di centroidi iniziali appropriati è un passo fondamentale della procedura:

- a volte i centroidi iniziali sono randomizzati, ma la performance dei cluster risultanti (mediante SSE) è spessp bassa

- una seconda strategia e randomizzare più volte i centroidi iniziali, eseguire la procedura e prendere il clustering più performante; questa strategia non sempre funziona perfettamente (ciò dipende dai dati), ma comunque è un passo avanti rispetto alla precedente

- un approccio efficace consiste nell'iniziare con un clustering gerarchico su un subset di dati; K cluster sono estratti e i centroidi di questi cluster vengono utilizzati come centroidi iniziali; l'approccio funziona spesso bene, ma è pratico solo se il campione è numericamente compatto e K è relativamente piccolo se confrontato al campione usato

- scegliere a random il primo centroide oppure considerare il centroide generale; poi per ogni centroide successivo selezionare il punto che è più lontanoda qualsivoglia centroide già battezzato. Il problema di questo approccio è chè può selezionare outlier e che è necessario calcolare numerose distanze; per porvi rimedio si può applicare la procedura ad un campione.

4.2.2.3 Considerazioni ulteriori

Outliers Quando viene impiegato l'SSE gli outlier possono influenzare i cluster; i centroidi risultanti potrebbero non essere rappresentativi quanto in assenza di outlier e anche l'SSE sarebbe più alto. Due strategie (se si decide di toglierli):

- li si individua prima di svolgere il clustering mediante le tecniche di outlier detection e li si toglie
- li si identifica in fase di post processing: ad esempio possiamo ordinare ciascun punto per l'SSE che ha contribuito ed eliminare quelli con contributo alto/anomalo

Ridurre l'SSE con il postprocessing Un modo ovvie per ridurre l'SSE è aumentare il numero di cluster creati K; ciò ridurrebbe l'SSE, ma spesso non si vuole un numero eccessivo di cluster.
La strategia spesso adottata è di focalizzarsi sui singoli cluster (dato che l'SSE è la somma dei contributi di ciascun cluster) effettuando splitting e/o merging. Due strategie che diminuiscono SSE aumentando il numero di cluster sono:

- splittare il cluster con il più largo SSE
- introdurre un nuovo centroide, spesso è scelto il punto più distante da qualsiasi centroide

Due strategie che diminuiscono il numero di cluster cercando di contenere l'aumento dell'SSE sono:

- smantellare un cluster (idealmente quello che fa incrementare SSE di meno), rimuovendo il suo centroide e riassegnando i punti ad altri cluster
- fare il merge di due cluster: vengono scelti i cluster con i centroidi più vicini, o quelli che fanno aumentare meno l'SSE

4.2.2.4 Limitazioni

K-means ha un certo numero di limitazioni:

- nell'individuare correttamente i cluster non aventi forma sferica, o che differiscono per dimensioni o densità. La difficoltà qui è che la funzione obiettiva (SSE) non matcha bene con i cluster da individuare; per quanto possibile l'esplorazione grafica aiuta qui;
- ha difficoltà nel fare clustering con outlier.

4.2.2.5 Bisecting K-means

Si tratta di una estensione dell'algoritmo base fondata su una idea semplice: per ottenere K cluster

- splittare il set di punti in due cluster;
- selezionare uno di questi e splittarlo ulteriormente;

- e così via sino a che si sono prodotti K cluster.

Sulla scelta di quale cluster da splittare possiamo scegliere alternativamente il cluster numericamente più grosso, quello con maggiore SSE o usare un criterio che contemperi i due aspetti.
Alcune peculiarità:

- spesso il bisecting k-means è usato come strategia di preprocessing per trovare i centroidi in un k-means classico
- registrando la sequenza di clustering prodotti possiamo anche usare il bisecting k-means per produrre un clustering gerarchico

4.2.3 Clustering con Mixture models

4.2.3.1 Introduzione

In questo approccio si assume che i dati siano stati generati da un processo casuale composto, creato sulla base di distribuzioni e parametri potenzialmente differenti, che si cerca di descrivere al meglio: ogni distribuzione componente corrisponde ad un cluster e i parametri della distribuzioni ne forniscono una descrizione (tipicamente in termini di centro e variabilità).
Più precisamente si assume che la generazione dei dati sia avvenuta prima selezionando una delle distribuzioni coinvolte e poi utilizzando questa per generare l'osservazione. Per un po di notazione:

- abbiamo che vi siano K distribuzioni ed m oggetti $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$
- la j-esima distribuzione ha parametri θ_j
- Θ indica l'insieme di tutti i parametri $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$
- $P(\mathbf{x}_i|\theta_j)$è la probabilità dell'i-esimo oggetto nel caso provenga dalla j-esima distribuzione
- la probabilità che la j-esima distribuzione sia scelta per generare l'oggetto è data da w_j, $1 < j < K$, e ovviamente, $\sum_j w_j = 1$.

La probabilità complessiva di un oggetto $\mathbf{x}_i$ è la somma delle probabilità dell'oggetto data ciascuna distribuzione, pesate per la probabilità dell'utilizzo di tale distribuzione:

$$P(\mathbf{x}_i|\Theta) = \sum_{j=1}^{K} w_j P(\mathbf{x}_i|\theta_j) \tag{4.3}$$

Se gli oggetti sono generati in maniera indipendente, la probabilità dell'intero campione è il prodotto della probabilità di ciascun singolo elemento $\mathbf{x}_i$

$$P(\mathcal{X}|\Theta) = \prod_{i=1}^{m} P(\mathbf{x}_i|\Theta) = \prod_{i=1}^{m} \sum_{j=1}^{K} w_j P(\mathbf{x}_i|\theta_j) \tag{4.4}$$

Siamo interessati a stimare i parametri di queste distribuzioni dai dati e descrivere dunque i cluster.

4.2.3.2 Stima di massima verosimiglianza

Partiamo da un problema semplice di stima dei parametri; ad esempio consideriamo un set di m punti generati da una gaussiana.
Un approccio standard per stimarne i parametri è quello della stima di massima verosimiglianza: assumendo che i punti siano generati indipendentemente la probabilità del campione è il prodotto delle loro probabilità (o meglio densità)

$$P(\mathcal{X}|\Theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \tag{4.5}$$

La funzione di sopra, ovvero la probabilità dei dati vista come una funzione dei parametri è, chiamata funzione di verosimiglianza.
Per stimare μ e σ, gli stimatori di massima verosimiglianza scelgono i valori dei parametri per i quali i dati sono i più probabili, ovvero che massimizzano 4.5 (per ragioni pratiche, poi si usa la log verosimiglianza nella massimizzazione).

4.2.3.3 Stimare dei parametri nei mixture model: l'algoritmo EM

Il problema è che, non conoscendo quale punto è generato da quale distribuzione del mixture model, non possiamo applicare il metodo della massima verosimiglianza direttamente.
L'algoritmo EM ci viene in soccorso: si tratta di una tecnica per risolvere problemi di massima verosimiglianza complessi composto dalle fasi:

1. si sceglie un valore iniziale arbitrario dei parametri;
2. **expectation** step: si calcolano le probabilità che ogni punto appartenga a ciascuna distribuzione, ovvero

$$P(\text{distribution } j|\mathbf{x}_i, \Theta)$$

3. **maximization** step: si usano queste stime per fornire nuove stime dei parametri;
4. si ripetono gli step 2 e 3 sino a che le stime dei parametri variano di poco.

4.2.3.4 Esempi

Esempio 1 La densità della normale univariata è;

$$P(x_i|\Theta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

I parametri sono $\theta = (\mu, \sigma)$ con μ la media e σ la deviazione standard.
Ora si assuma che vi siano due distribuzioni gaussiane caratterizzate da una stessa deviazione standard di 2 e dai media -4 e +4 rispettivamente; si assuma ance che ognuna delle due sia selezionata con egual probabilità, ovvero $w_1 = w_2 = 0.5$. Allora

$$P(x|\Theta) = \frac{1}{2\sqrt{2\pi}} e^{-\frac{(x+4)^2}{8}} + \frac{1}{2\sqrt{2\pi}} e^{-\frac{(x-4)^2}{8}}$$

TODO: I pesi dove sono finiti qui?

Iniziamo l'algoritmo EM facendo previsioni iniziali delle due medie μ_1 e μ_2,

diciamo $\mu_1 = -2$ e $\mu_2 = 3$; dunque i parametri iniziali $\theta = (\mu, \sigma)$ sono $\theta_1 = (-2, 2)$ e $\theta_2 = (3, 2)$, mentre l'insieme completo dei parametri del modello è $\Theta = \{\theta_1, \theta_2\}$:

- per l'expectation step vogliamo calcolare la probabilità che i punti provengano da una particolare distribuzione, ovvero vogliamo $P(\text{distribution } 1|x_i, \Theta)$ e $P(\text{distribution } 2|x_i, \Theta)$. Questi valori possono essere espressi da una diretta applicazione della regola di Bayes:

$$P(\text{distribution } j|x_i, \Theta) = \frac{0.5P(x_i|\theta_j)}{0.5P(x_i|\theta_1) + 0.5P(x_i|\theta_2)}$$

  Ad esempio ipotizzando che un punto sia 0, utilizzando la formula della gaussiana, calcoliamo che $P(0|\theta_1) = 0.12$ mentre $P(0|\theta_2) = 0.06$ (anche qui in realtà si tratta di densità). Usando questi valori nell'equazione di sopra troviamo che $P(\text{distribution } 1|x_i, \Theta) = 0.12/(0.12 + 0.06) = 0.66$ mentre $P(\text{distribution } 2|x_i, \Theta) = 0.06/(0.12 + 0.06) = 0.33$. Ciò significa che il punto 0 è più probabile che appartenga alla prima distribuzione

- dopo aver calcolato le probabilità di appartenenza di ciascun punto del campione calcoliamo le nuove stime di μ_1 e μ_2 come segue

$$\mu_1 = \sum_{i=1}^{20000} x_i \frac{P(\text{distribution } 1|x_i, \Theta)}{\sum_{i=1}^{20000} P(\text{distribution } 1|x_i, \Theta)}$$
$$\mu_2 = \sum_{i=1}^{20000} x_i \frac{P(\text{distribution } 2|x_i, \Theta)}{\sum_{i=1}^{20000} P(\text{distribution } 2|x_i, \Theta)}$$

Si ripetono gli step sino a che le stime di μ_1, μ_2 non cambiano o cambiano molto poco.

Esempio 2 Ipotizzando che i dati siano generati da una mistura di curve gaussiane, ad esempio

$$0.3N(-1, 0.5) + 0.7N(1, 0.5)$$

Formalizzando le variabili aleatorie sono

$$\Delta \sim Bernoulli(\pi)$$
$$Y_1 \sim N(\mu_1, \sigma_1)$$
$$Y_2 \sim N(\mu_2, \sigma_2)$$

La variabile aleatoria complessiva è

$$Y = (1 - \Delta)Y_1 + \Delta Y_2$$

La generazione dei dati:

- si generano y_1, y_2 secondo le distribuzioni Y_1, Y_2
- si genera $\delta = 0$ o $\delta = 1$ secondo la distribuzione Δ; δ "sceglie" da quale normale il campione venga generato

La pdf di Y è

$$f_Y(y) = (1-\pi)\phi_{\theta_1}(y) + \pi\phi_{\theta_2}(y)$$

dove $\phi_{\theta_1}(y)$ e $\phi_{\theta_2}(y)$ sono le pdf normali di parametri $\theta_1 = (\mu_1, \sigma_1)$ e $\theta_2 = (\mu_2, \sigma_2)$.
Si stimano dunque 5 parametri (scalari): $\pi, \mu_1, \sigma_1, \mu_2, \sigma_2$.
EM è una tecnica di ottimizzazione che alterna passi di Expectation (E) e Maximization (M), nello specifico:

- si definiscono parametri ausiliari $\hat{\gamma}_1, \hat{\gamma}_2, \ldots, \hat{\gamma}_n \in [0,1]$ con $\hat{\gamma}_i$ che rappresenta la responsabilità della seconda componente $\hat{\phi}_{\theta_2}$ nella generazione di y_i (e $1-\gamma_i$ rappresenta di conseguenza quella della prima componente)
- Expectation calcola, tenendo fissi parametri stimati $\hat{\pi}$ $\hat{\theta}_1 = (\hat{\mu}_1, \hat{\sigma}_1)$, $\hat{\theta}_2 = (\hat{\mu}_2, \hat{\sigma}_2)$ ogni responsabilità $\hat{\gamma}_i$ come valore atteso di una variabile bernoulliana $\Delta_i \in \{0,1\}$, $i = 1, 2, \ldots, n$
- Maximization calcola i i parametri media e varianza delle componenti $\hat{\theta}_1 = (\hat{\mu}_1, \hat{\sigma}_1)$, $\hat{\theta}_2 = (\hat{\mu}_2, \hat{\sigma}_2)$ e il parametro di composizione $\hat{\pi}$ tenendo fisse le responsabilità e pesando il contributo di ogni osservazione y_i con la sua responsabilità γ_i

Algoritmo EM per due componenti gaussiane segue le seguenti fasi:

1. Scegli parametri iniziali $\hat{\pi}$, $\hat{\pi}$ $\hat{\theta}_1 = (\hat{\mu}_1, \hat{\sigma}_1)$, $\hat{\theta}_2 = (\hat{\mu}_2, \hat{\sigma}_2)$
2. dati i parametri correnti calcola le responsabilità attese

$$\hat{\gamma}_= \frac{\hat{\pi}\hat{\phi}_{\theta_2}}{(1-\hat{\pi})\phi_{\theta_1}(y_i) + \hat{\pi}\phi_{\theta_2}(y_i)}, \quad i = 1, 1, \ldots, n$$

3. date le responsabilità correnti $\hat{\gamma}_1, \ldots, \hat{\gamma}_n$ calcola il parametro di composizione $\hat{\pi} = \frac{1}{n}\sum_{i=1}^n \hat{\gamma}_i$ e calcola medie e varianze

$$\hat{\mu}_1 = \frac{\sum_{i=1}^n (1-\hat{\gamma}_i) y_i}{\sum_{i=1}^n (1-\hat{\gamma}_i)} \tag{4.6}$$

$$\hat{\mu}_2 = \frac{\sum_{i=1}^n \hat{\gamma}_i y_i}{\sum_{i=1}^n \hat{\gamma}_i} \tag{4.7}$$

$$\hat{\sigma}_1^2 = \frac{\sum_{i=1}^n (1-\hat{\gamma}_i)(y_i - \mu_1)^2}{\sum_{i=1}^n (1-\hat{\gamma}_i)} \tag{4.8}$$

$$\hat{\sigma}_2^2 = \frac{\sum_{i=1}^n (1-\hat{\gamma}_i)(y_i - \mu_2)^2}{\sum_{i=1}^n (1-\hat{\gamma}_i)} \tag{4.9}$$

4. ripeti 2 e 3 fino a convergenza

4.2.3.5 Vantaggi e svantaggi

I contro sono:

- l'algoritmo può essere lento
- non è pratico per modelli con un gran numero di componenti

- non funzione bene con cluster che contengono solamente pochi punti se sono co-lineari
- vi è da scegliere la forma del modello da stimare

I punti di forza:

- mixture model sono più generali che k means o fuzzy c-means poiché possono usare distribuzioni di vario tipo
- possono trovare cluster di diversa dimensione e forma ellittica
- un approccio basato su modello permette un modo disciplinato di eliminare della complessità associata ai dati
- è facile caratterizzare i cluster prodotti, dato che possono essere descritti da un piccolo numero di parametri

4.3 Clustering gerarchico (agglomerativo)

Gli algoritmi di clustering gerarchico sono una seconda importante categoria di metodi. Ve ne sono due approcci:

- *agglomerativi* (agglomerativo): inizia con tanti cluster contenenti ciascuno un punto, e procedi facendo il merge dei cluster più vicini; ciò necessita di una misura di prossimità tra cluster.
  Questo è l'approccio più frequente e quello su cui ci si concentra nel seguito.
- *divisivi* (scissorio): inizia con cluster unico contenente tutti i punti e procedi nello splitting sino a che non si giunge a dei singleton (cluster di singoli punti; qui dobbiamo decidere quale cluster splittare ad ogni step e come fare lo splitting

Un clustering gerarchico è spesso presentato graficamente mediante un grafico ad albero detto **dendogramma**, che mostra le relazioni cluster-subcluster e l'ordine in cui i cluster sono mergiati.

4.3.1 Clustering agglomerativo: algoritmo base

4.3.1.1 Algoritmo aggregativo di Anderberg

Anderberg ha proposto un algoritmo aggregativo generale.

1. Inizializza:
   (a) N cluster $C_1, C_2, \ldots, C_n$ contenenti un unico oggetto ciascuno
   (b) una matrice $N \times N$ nella quale l'elemento (i, j) è $d_i j$, la distanza
   (c) il numero di cluster a N
2. Trova la coppia di cluster piú simili nella matrice, secondo una misura di dissimilarità che estenda la dissimilarità tra oggetti. Siano p, q gli indici dei cluster della coppia

3. crea l'unione di C_p e C_q e assegna l'indice q ad essa
4. aggiorna la matrice calcolando la dissimilarità tra C_q e gli altri cluster ed eliminando la riga e la colonna con l'indice p
5. decrementa il numero di cluster
6. se il numero di cluster è 1, termina; altrimenti vai al passo 2

Algoritmi aggregativi largamente usati possono essere ottenuti scegliendo la dissimilarità tra cluster.

4.3.1.2 Definire la dissimilarità tra cluster

Step chiave è la definizione della prossimità tra due cluster.
Tre criteri derivano dalla vista di un cluster come un **grafo**:

- MIN (single link): la prossimità di due cluster coincide con la prossimità dei due elementi che vi appartengono che sono più *vicini* tra loro

$$d_{AB}^{SL} = \min_{x \in A, y \in B} d_{xy}$$

- MAX (complete link): la prossimità di due cluster coincide con la prossimità dei due elementi che vi appartengono che sono più *lontani* tra loro

$$d_{AB}^{CL} = \max_{x \in A, y \in B} d_{xy}$$

- group average: la prossimità di due cluster è la media delle prossimità calcolate sul prodotto cartesiano dei punti provenienti dai due insiemi. è una via di mezzo tra i due metodi precedenti

$$d_{AB}^{GA} = \frac{1}{|A|\,|B|} \sum_{x \in A, y \in B} d_{xy}$$

Se adottiamo una prototype-based view, nella quale ogni cluster è rappresentato da un centroide:

- misuriamo la distanza tra centroidi
- metodo di Ward: si misura la prossimità tra due cluster in termini dell'incremento di SSE che risulterebbe dal farne il merge

4.3.2 Esempi

Per provare le varie robe consideriamo i seguenti dati

```
db <- data.frame(
    id = letters[1:6],
    x = c(0.40, 0.22, 0.35, 0.26, 0.08, 0.45),
    y = c(0.53, 0.38, 0.32, 0.19, 0.41, 0.30))
```

La matrice di distanze (euclidee di default) è calcolata come

```
(dists <- dist(db))

## Warning in dist(db):  NA introdotti per coercizione

##           1         2         3         4         5
## 2 0.2869669
## 3 0.2643861 0.1753568
## 4 0.4503332 0.2378024 0.1936492
## 5 0.4185690 0.1753568 0.3485685 0.3481379
## 6 0.2882707 0.2982449 0.1249000 0.2688866 0.4727579
```

Il risultato di single link è in 4.2: l'altezza al quale due cluster sono mergiati nel dendogramma riflette la distanza dei due cluster. Ad esempio la distanza tra i cluster $\{3,6\}$ e $\{2,5\}$ è data da

$$\begin{aligned} dist(\{3,6\},\{2,5\}) =& min(dist(3,2), dist(6,2), dist(3,5), dist(6,5) \\ =& min(0.15, 0.25, 0.28, 0.39) \\ =& 0.15 \end{aligned}$$

```
single <- hclust(dists, method = 'single')
plot(single,
     labels = db$id,
     main = 'Single link (MIN)',
     xlab = '', ylim=c(0, 0.25))
```

Il risultato di complete link è in 4.3: ad esempio la distanza tra i cluster $\{3,6\}$ e $\{4\}$ è data da

$$\begin{aligned} dist(\{3,6\},\{4\}) =& max(dist(3,4), dist(6,4)) \\ =& max(0.15, 0.22) \\ =& 0.22 \end{aligned}$$

```
complete <- hclust(dists, method = 'complete')
plot(complete,
     labels = db$id,
     main = 'Complete link (MAX)',
     xlab = '', ylim=c(0, 0.25))
```

4.3.3 Caratteristiche e issues

Applicazioni del clustering gerarchico si hanno quando l'applicazione stessa richiede la creazione di una classificazione gerarchica.
I problemi principali:

- *mancanza di una funzione obiettivo*: non si massimizza una funzione obiettivo generale ma si effettuano una serie di scelte locali ad ogni step su quali cluster dovrebbero essere mergiati (quindi non si possono confrontare clustering differenti).

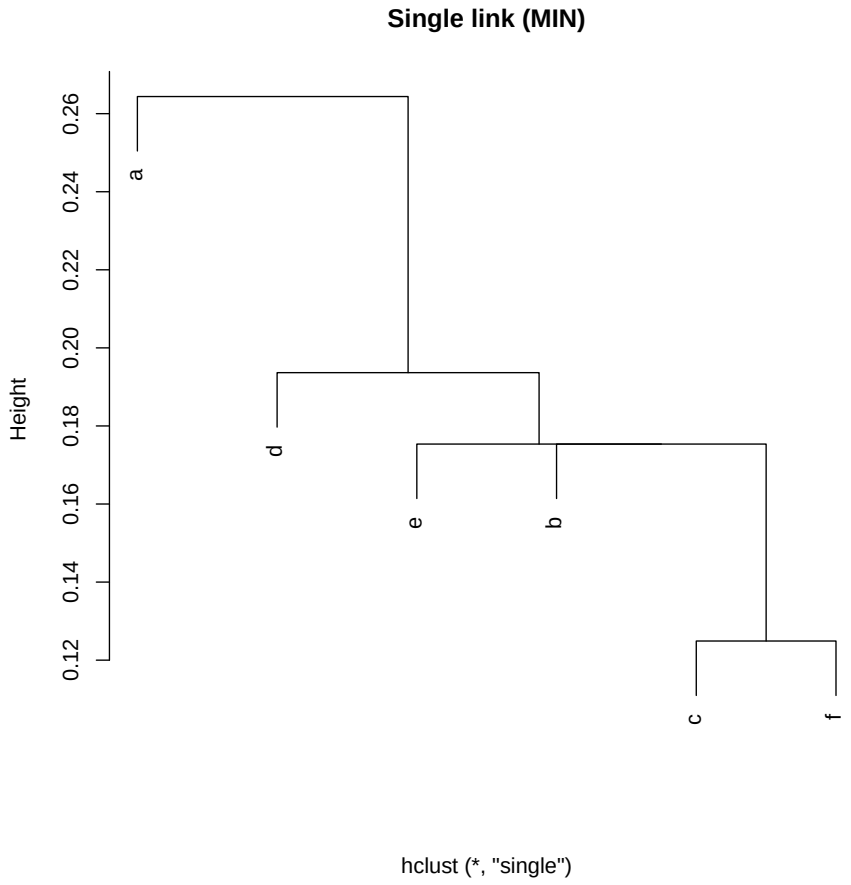


Figura 4.2: Single link (MIN)

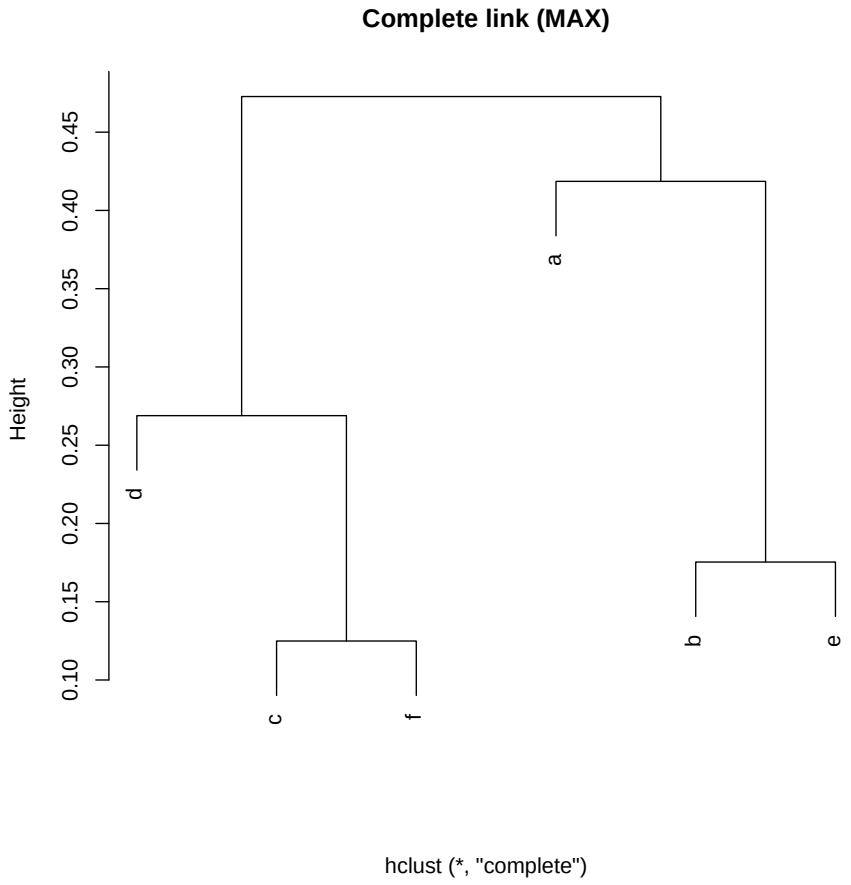


Figura 4.3: Complete link (MAX)

Questo però permette di evitare la difficoltà di risolvere un difficile problema di ottimizzazione combinatoria (si può dimostrare che minimizzare SSE in questo setting è impossibile). Inoltre non vi sono problemi difficoltà nello scegliere i punti iniziali;

- *le decisioni di merge sono definitive*: una volta che una decisione di merge è stata fatta (in maniera da ottimizzare localmente) non può essere annullata e ciò impedisce ad un criterio di ottimizzazione local di diventare uno globale.
  Ciò può essere un drawback soprattutto dati noisy, multidimensionali (ad esempio documenti).
  A parziale messa a posto si può iniziare con un clustering partitivo come un k-means per creare tanti piccoli cluster e in seguito effettuare un clustering gerarchico utilizzando questi ultimi come punto di partenza;
- sono dispendiosi di risorse (computazionali e di storage): complessità temporale quadratica e complessità spaziale
- bias in favore di alcune forme
- haining (single linkage)

4.4 Density based clustering

Il density-based clustering identifica regioni ad alta densità che sono separate da altre regioni a densità minore.

4.4.1 Density estimation

La funzione di densità di probabilità (pdf) f di una variabile casuale X descrive la distribuzione di X e permette di calcolare le probabilità di eventi:

$$P(x < X < x') = \int_x^{x'} f(x)\,dx$$

Gli statistici sono stati interessati altresì al problema inverso: dati dei dati campionati da una pdf f sconosciuta, può questa essere ricostruita?

4.4.1.1 Approcci alla stima di densità

Metodi statistici per trovare stime della funzione a partire da dataset multivariate vanno in due approcci: parametrici e non parametrici.
Negli approcci parametrici:

- si seleziona una determinata famiglia di densità, dipendente da un set di parametri
  $$f(\cdot;\theta), \theta = (\theta_1, \ldots, \theta_m)$$
  ad esempio $N(\mu, \sigma^2)$
- si ottiene la stima migliore possibile dei parametri a partire dai dati
  $$\hat{\theta}(x_1, \ldots, x_N) = (\hat{\theta}_1(x_1, \ldots, x_N), \ldots, \hat{\theta}_m(x_1, \ldots, x_N))$$

- si sostituiscono i parametri nell'espressione della densità della famiglia:

$$\hat{f}(\cdot) = \hat{f}(\cdot; \hat{\theta}(x_1, \ldots, x_N))$$

Negli approcci non parametrici:

- la funzione di densità è stimata dai dati in maniera più diretta, con minori assunzioni (sulla famiglia dalla quale i dati provengono);
- l'espressione di stima dipende anche qui da parametri

$$\hat{f}(\cdot) = \varphi(\cdot; x_1, \ldots, x_N; h), \quad h = (h_1, \ldots, h_m)$$

- la ricerca nell'ambito ha derivato valori ottimali per h con riguardo a criteri di convergenza

4.4.1.2 Metodi di stima della densità

Istogrammi di frequenza L'esempio più vecchio di stimatore non parametrico è l'istogramma *di frequenza*; definendo una origine $t_0 \in R$ e un'ampiezza non nulla $h \in \mathbb{R}_+$, l'istogramma partiziona i reali in intervalli semi aperti, o bins, tali che i tick sono multipli di tale ampiezza più l'origine:

$$B_k = [t_0 + kh, t_0 + (k+1)h), \quad k \in \mathbb{Z}$$

Sia ν_k il numero di oggetti nel bin B_k:

$$\nu_k = |B_k \cap D|$$

Il valore dell'istogramma di frequenza al punto x è il ν_k del bin che contiene x:

$$FH(x) = \nu_k, \quad \text{se } x \in B_k, \text{ per } k \in \mathbb{Z}$$

Istogrammi di densità univariata Passando all'istogramma *di densità*, assumendo una sequenza crescente di t_i che definisca i bins di ampiezza a scelta:

$$B_k = [t_k, t_{k+1})$$

il valore dell'istogramma nel k-esimo bin è uguale al numero di oggetti ν_k diviso un fattore di normalizzazione. Se $x \in B_k$ per $k \in \mathbb{Z}$:

$$\hat{H}(x) = \frac{\nu_k}{N(t_{k+1} - t_k)} = \frac{1}{N} \sum_{k=-\infty}^{+\infty} \sum_{i=1}^{N} \frac{I_{B_k}(x) I_{B_k}(x_i)}{t_{k+1} - t_k}$$

where $I(A)$ denota la funzione indicatrice dell'insieme A.
Tipicamente la sequenza di tick $\{t_k\}$ è regolare e tutti i bins hanno ampiezza uguale:

$$\forall k \in \mathbb{Z}, t_{k+1} - t_k = h$$

Similmente all'istogramma di frequenza, quello di densità è definito completamente dai dati e da due parametri, l'ampiezza h e l'origine t_0; se $x \in B_k$ per $k \in \mathbb{Z}$:

$$\hat{f}(x) = \frac{\nu_k}{Nh} = \frac{1}{Nh} \sum_{k=-\infty}^{+\infty} \sum_{i=1}^{N} I_{B_k}(x) I_{B_k}(x_i) \tag{4.10}$$

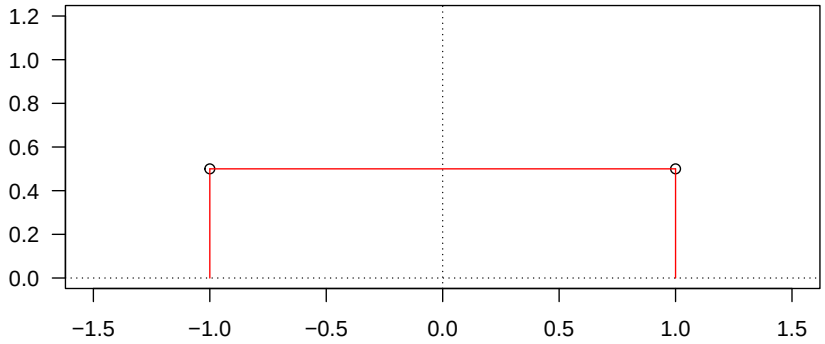


Figura 4.4: Box function

Stimatori di densità naive Similmente all'istogramma di densità, lo stimatore naive conta il numero di oggetti in un intervallo ma libera la stima dalla dipendenza sull'origine t_0.
Gli intervallo non sono definiti da una sequenza di tick: ogni campione è il centro di un intervallo; la conta degli oggetti nell'intervallo è la densità stimata a x:

$$\hat{f}(x) = \frac{1}{2Nh} \left| [x-h, x+h) \cap D \right| \tag{4.11}$$

L'espressione di cui sopra può essere vista come una stima della densità f se scritta come

$$f(x) = \lim_{h \to 0} \frac{1}{2h} P(x-h < X < x+h) \tag{4.12}$$

La forma algebrica dello stimatore passa per il definire una funzione "box" (a scatola) con dominio $\{-1, 1\}$, dove è uguale a $1/2$:

$$w(x) = \begin{cases} 1/2 & -1 \leq x \leq 1 \\ 0 & \text{altrimenti} \end{cases} \tag{4.13}$$

Lo stimatore naive somma N copie della funzione, ciascuna copia shiftata ad un unico punto proveniente dal campione e scalata per l'ampiezza della finestra:

$$\hat{f}(x) = \frac{1}{Nh} \sum_{i=1}^{N} w\left(\frac{x - x_i}{h} \right) \tag{4.14}$$

L'aspetto visivo è spesso poco smooth e soddisfacente (figura 4.5); vi sono discontinuità ad $x_i + h$.

Kernel density estimator È una generalizzazione dello stimatore naive: la funzione box è rimpiazzata da una funzione kernel K che soddisfi

$$\int_{-\infty}^{+\infty} K(x) dx = 1$$

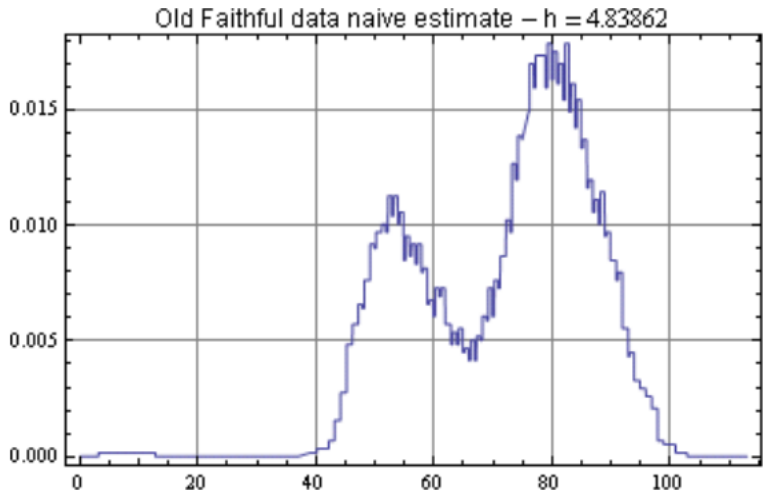


Figura 4.5: Naive estimate

Solitamente è unimodale, simmetrica, non negativa, dal dominio limitato. Il kernel density estimator è dato da

$$\hat{f}(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right) \tag{4.15}$$

h è l'ampiezza della finestra e controlla l'ammontare di smoothing.
Vari esempi funzioni kernel possono essere adottati:

- gaussiana
$$K(x) = \frac{1}{2\pi} e^{-\frac{1}{2}x^2}$$
- triangolare
$$K(x) = (1 - |x|) I_{[-1,1]}(x)$$
- Epanechnikov:
$$K(x) = \frac{3}{4}(1 - x^2) I_{[-1,1]}(x)$$
- biweight
$$K(x) = \frac{15}{16}(1 - x^2)^2 I_{[-1,1]}(x)$$

Un esempio con kernel normale

```
plot(density(faithful$waiting, bw = 4.83, kernel = 'gaussian'), col = 'blue')
```

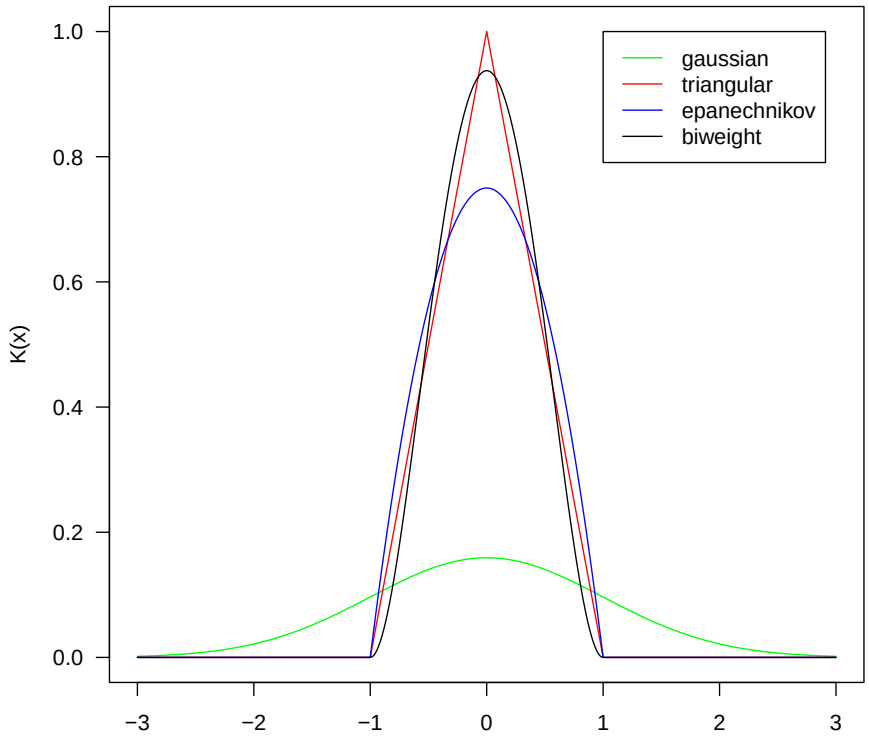


Figura 4.6: Varie funzioni kernel

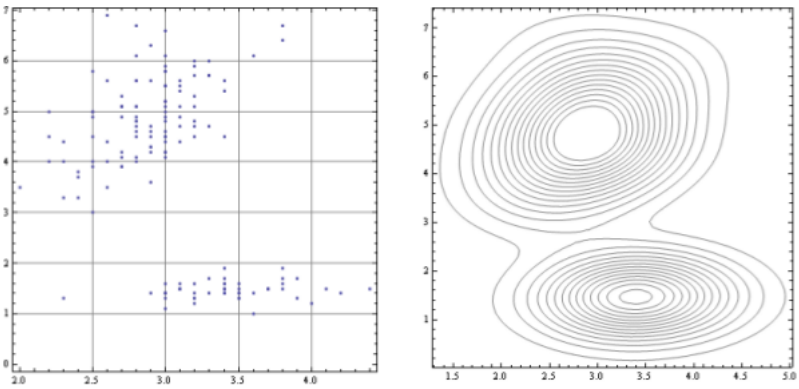
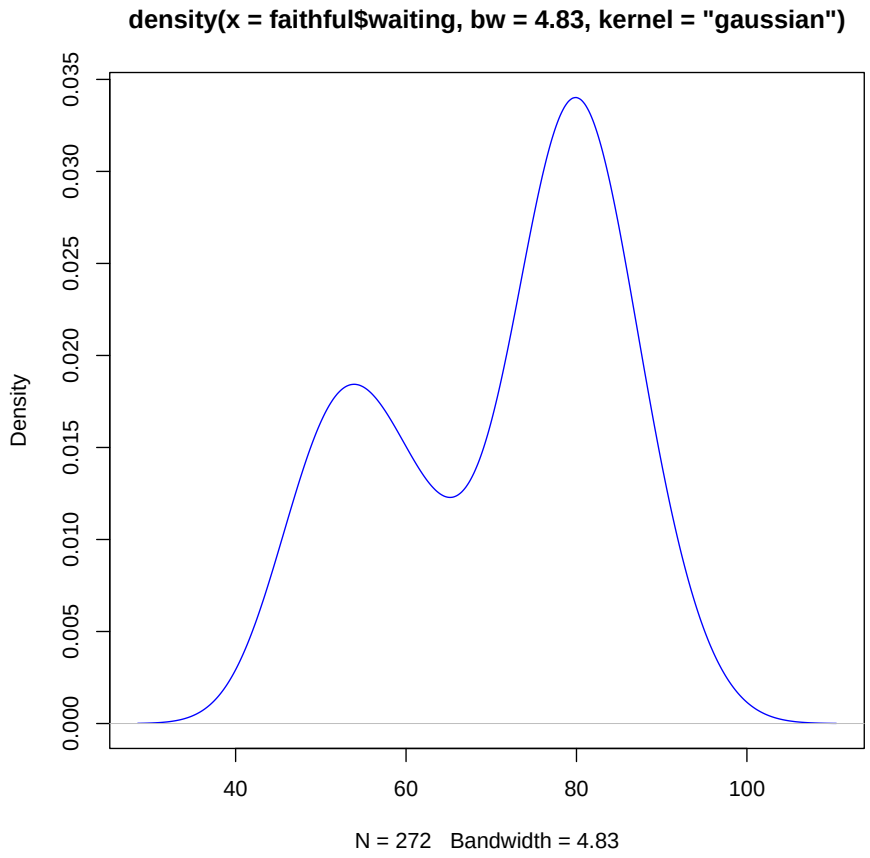


Figura 4.7: Esempio bivariato di KDE



La KDE può essere applicata anche a dati bivariati (le immagini possono essere dei contour plot, come in 4.7 o dei grafici in 3d); in generale la funzione

di densità in d dimensioni:

$$f^D(x) = \frac{1}{Nh^d} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right)$$

Ad esempio con un kernel gaussiano

$$f^D_{Gauss}(x) = \frac{1}{Nh^d(2\pi)^{d/2}} \sum_{i=1}^{N} e^{-\frac{dist(x,x_i)^2}{2h^2}}$$

4.4.1.3 Stima di densità e clustering

Sembrerebbe che stimatori smooth di densità forniscano abbastanza informazione per identificare visivamente i cluster. Occorre però avere una risposta a questo:

- Come definire però i cluster boundaries?
- I cluster sono definiti da una o più mode?
- Qualsiasi moda fa si che le si crei un cluster attorno?

I vari algoritmi density based rispondono in maniera diversa alle domande.

4.4.2 DBSCAN

DBSCAN è una procdura di clustering: basata su densità che separa regioni di noise dai cluster.
Si fonda su centre-based density: la densità è stimata per un particolare punto contando il numero di punti entro un determinato raggio ε (contando il punto stesso)[1]:

- dato un database D di oggetti, considerando un raggio $\varepsilon \in \mathbb{R}_+$ e una soglia minima di densità $MinPts \in \mathbb{N}$;
- il ε-vicinato $N_\varepsilon(p)$ di un oggetto p è l'insieme di oggetti che distano al massimo ε da p:
$$\{q \in D : d(p,q) \leq \varepsilon\}$$
- un oggetto q è *directly density reachable* da un oggetto p se appartiene al suo ε-vicinato e quest'ultimo supera la soglia $MinPts$:
$$q \in N_\varepsilon(p)$$
$$|N_\varepsilon(p)| \geq MinPts$$
- un oggetto q è *density reachable* (non directly) da un oggetto p se esiste una sequenza di oggetti $q_1, \ldots, q_n$ con $q = q_1, p = q_n$ tale che q_{i+1} è directly density reachable da q_i per $i = 1, \ldots, n-1$
- un oggetto q è *density connected* ad un oggetto p se esiste un oggetto o tale che p e q sono density reachable da o

[1] Quindi se un punto ha nel suo raggio 8 punti, la sua densità è di 9.

Selezionati ε e $MinPts$, un cluster viene definito da DBSCAN come un insieme non vuoto di oggetti tale che:

- $\forall p, q \in D$, se $p \in C$ e q è density reachable da p (con riguardo a ε e $MinPts$, allora $q \in C$ (*massimalità*);
- $\forall p, q \in C$, p è density connected a q (*connettività*)

In parole più semplici, dato ε e $MinPts$, l'algoritmo classifica ciascun punti come:

- *core points*: se entro un certo il raggio ε da questi vi sono più di $MinPts$ punti;
- *border points*: se non è un core point egli stesso, ma è nel raggio di un core point;
- *noise points*: un punto che non è ne core ne border point.

Dopodichè

1. etichetta tutti i punti come core, border o noise
2. elimina tutti i noise
3. apporre un link tra tutti i core point che sono entro ε l'uno dall'altro
4. ogni gruppo di core points connessi forma un cluster
5. assegna ogni border point ad uno dei cluster dei core points cui è associato

4.4.2.1 Selezione dei parametri di DBSCAN

Il metodo è semplice ma dipende dalla scelta del raggio ε e dalla soglia *MinPts*:

- in merito a ε, se troppo largo o troppo stretto tutti i punti avranno la stessa densità (pari rispettivamente al numero di punti disponibili o 1);
- riguardo a *MinPts* esso determina la sensibilità dell'algoritmo: se troppo piccolo anche un numero basso di punti abbastanza vicini che sono noise o outlier sono classificati come cluster, se troppo alto cluster piccoli (di dimensione inferiore a *MinPts*) sono considerati noise.

L'approccio base per la scelta è guardare alla distanza di ogni punto con il suo k-esimo punto più vicino:

- si calcola la distanza (detta *k-dist*) di ogni punto con il k-esimo più vicino (l'algoritmo DBSCAN originale considera $k = 4$, che è ragionevole per la maggior parte dei dataset bidimensionali);
- si ordinano i punti per k-dist crescente e si plottano
- ci si aspetta di vedere un rapido aumento in k-dist a partire da un certo po';
- si usa il valore di *k-dist* prima del drastico aumento come ε e il parametro k come MinPts: se si fa cio i punti per i quali k-dist è minore di ε saranno core point, gli altri saranno noise o border.

Il valore di ε determinato in questo modo dipende da k, sì, ma non cambia così drasticamente.

4.4.2.2 Punti di forza e debolezza

- Dato che DBSCAN usa una definizione di cluster basata sulla densità è resistente alla noise e può gestire cluster di shape e size arbitrari;
- DBSCAN può trovare diversi cluster che non potrebbero essere trovati con k-means (quelli di forma non globulare);
- DBSCAN può avere difficoltà nel caso le densità dei diversi cluster vari ampiamente (considerato che la soglia *MipPts* è unica); alcuni cluster "naturali" possono non essere individuati se si tiene una soglia alta per scremare un'area ad alta densità da un'altra di densità intermedia che però è noise.
- ha difficoltà condati high dimensional perché la densità è più difficile da definire per tali dati
- può essere esoso quando il calcolo dei nearest neighbor richiede di calcolare tutte le prossimità (come solitamente è per dati multidimensionali)
- complessità $O(NlogN)$

4.4.3 DENCLUE

DENCLUE (DENsity-based CLUstEring) è un algoritmo di clustering density-based, che usa funzioni di densità kernel per stimare la densità.

4.4.3.1 Algoritmo

L'algoritmo segue le fasi:

1. stima la pdf mediante uno stimatore kernel;
2. identifica i punti che sono massimi locali della stima di densità: questi sono *density attractors*;
3. mediante una procedura di "scalata" della collina associa ogni oggetto ad un density attractor (muovendo nella direzione del massimo incremento in densità); si dice che un oggetto x è density attracted ad un density attractor x^* se e solo se una procedura di climbing della collina converge a x^* partendo da x;
4. definisce un cluster come l'insieme di punti associati ad un density attractor avente un valore superiore ad una soglia $\geq \xi$ specificata dall'utente
5. unisce i cluster connessi da un path formato da punti che hanno tutti una densità $\geq \xi$

4.4.3.2 Cluster riconosciuti

DENCLUE è in grado di riconosce due tipi di cluster,

- center-defined cluster: simili ai cluster riconosciuti da algoritmi con bias verso gli ellissoidi (BIRCH, K-means)
- multicenter-defined cluster: simili a cluster con forma arbitraria (ad esempio riconosciuti da Single Link o DBSCAN)

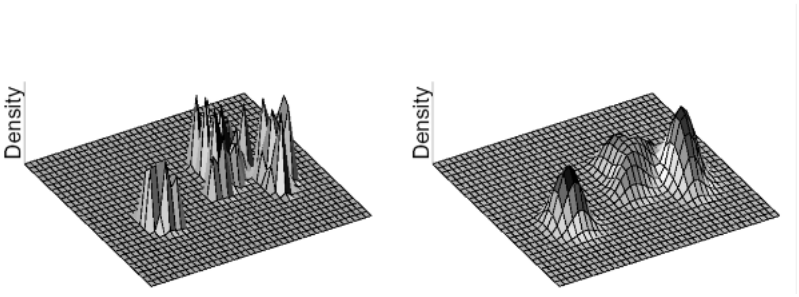


Figura 4.8: Esempio di center-defined cluster

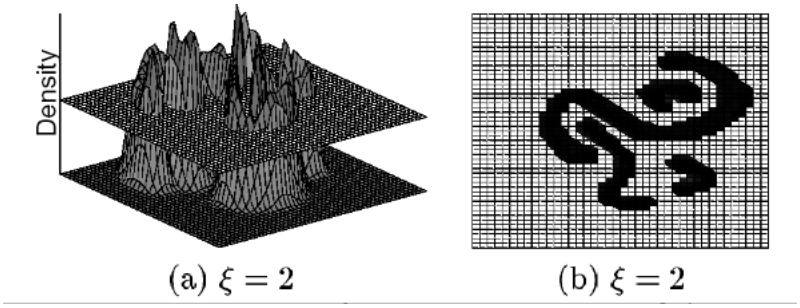


Figura 4.9: Esempio di multicenter-defined cluster

Center-defined cluster Un center-defined cluster (figura 4.8) per un attrattore x^* con riferimento ad una finestra d'ampiezza h e un livello di significatività ξ è un subset C di D tale che ogni $x \in C$ è density attracted verso x^* e

$$f^D(x^*) \geq \xi$$

Punti che sono attratti ad un density attractor x^* con

$$f^D(x^*) < \xi$$

sono considerati outlier o noise.

Multicenter-defined cluster Un multicenter-defined cluster (figura 4.9) per un insieme di attrattori di densità X con riferimento ad una finestra di ampiezza h e un livello di significatività ξ è un subset C di D tale che:

- $\forall x \in C, \exists x^* \in X : f^D(x^*) \geq \xi$, x è density attracted a x^*
- $\forall x_1^*, x_2^* \in X, \exists P(x_1^*, x_2^*) \forall p \in P : f^D(p) \geq \xi$ dove $P(u, v)$ è un path tra u e v

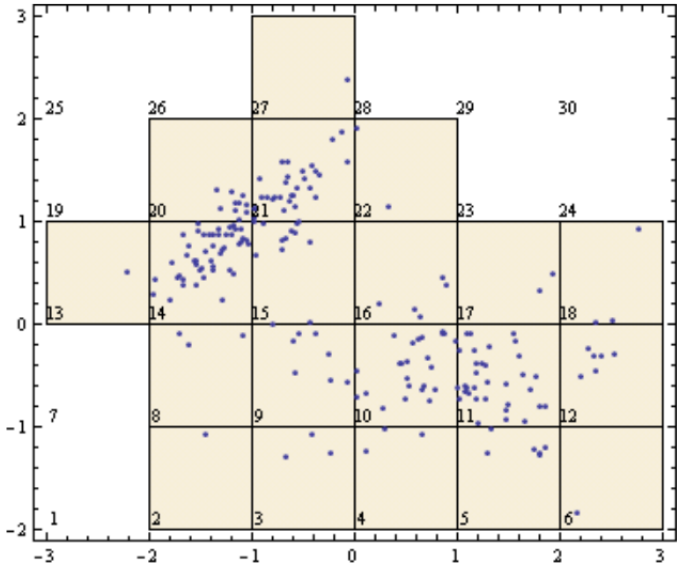


Figura 4.10: Ipercubi in un dataset fittizio

Ottimizzazione Un primo modo di ottimizzare il calcolo di una densità è limitare l'applicazione del nocciolo solamente agli oggetti in prossimità di x:

$$f^D(x) = \frac{1}{Nh^d} \sum_{i:x_i \in near(x)} K\left(\frac{x - x_i}{h}\right) \tag{4.16}$$

L'ipercubo dei dati viene fatto a cubettini (figura 4.10) con lato lungo $2h$, numerati progressivamente e indicizzati in un B-tree, poi;

- gli ipercubi limitrofi popolati da un largo numero di oggetti sono linkati
- la procedura per scalare la collina sfrutta gli indici dell'ipercubo per associare ogni oggetto ad un density attractor
- la densità di un oggetto x è calcolata utilizzando solamente oggetti nei cubi limitrofi a quello di x

4.4.3.3 Pro e contro

DENCLUE fornisce un modo flessibile ed più accurato di calcolare la densita rispetto ad altri metodi (DBSCAN è un caso particolare di DENCLUE).
I vantaggi sono:

- saldamente fondato su metodi statistici conosciuti
- immune a larghi ammontari di noise
- generalizza single link, DBSCAN, K-means: generalizza i metodi parititivi, gerarchici e basati su località
- performance migliore di DBSCAN su dataset che hanno più di 10 dimensioni, contenenti grande massa di noise;

- l'algoritmo processa solamente regioni popolate, pertanto mostra incrementi in performance
- modificabile perché produca una gerarchia di cluster modificando a step h

I punti di debolezza sono:

- tende ad essere computazionalmente più tosto delle alternative;
- molti parametri;
- guidelines to select ranges

4.5 Algoritmi scalabili

Anche il miglior algoritmo è inutile se impiega troppe risorse: BIRCH (Balanced Iterative Reducing and CLustering using Hierarchies) è un algoritmo di clustering efficiente per dati in uno spazio euclideo.

4.5.1 Algoritmi a passo unico

BIRCH rientra tra gli algoritmi a passo unico. Un tipico template di questi algoritmi segue queste fasi:

1. il primo oggetto è letto da input e si inizializza il cluster 1 in base alle sue proprietà
2. per ogni successivo oggetto letto da input:
    - se il massimo valore (considerati tutti i cluster) di una funzione di qualità (dipendente dalle proprietà dell'oggetto e dalla sinossi del cluster) soddisfa una determinata soglia, le proprietà dell'oggetto sono impiegate per aggiornare la sinossi rappresentativa del cluster che massimizza la funzione, cui l'oggetto viene assegnato;
    - altrimenti una sinossi rappresentativa di un nuovo cluster è inizializzata in base alle proprietà dell'oggetto.

Gli algoritmi a passo unico presentano le seguenti **caratteristiche**:

- complessità temporale (esecuzione) lineare nel numero di oggetti
- complessità spaziale (memoria) costante nel numero di oggetti
- sensibilità all'ordine in cui vengono forniti gli oggetti
- le sinossi (ad esempio media e varianza) che occupano spazio costante e aggiornabili in tempo costante a volte possono essere grossolane per rappresentare caratteristiche complesse di un cluster

4.5.2 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) è un sofisticato algoritmo a passo unico per il problema del clustering partitivo di dati vettoriali.
Il setting è quello di un clustering partitivo con memoria limitata in cui:

- come input si ha un data set di N vettori, un numero intero positivo K di cluster desiderati, misura basata sulla distanza
- come obiettivo vi è trovare una partizione del data set di cardinalità K che minimizzi la misura con i seguenti vincoli aggiuntivi:
  - la quantità di memoria disponibile è limitata e insufficiente per la memorizzazione dell'intero data set
  - il costo di I/O deve essere minimizzato

4.5.2.1 Elementi strumentali dell'algoritmo

Clustering feature BIRCH è basato sul concetto di **clustering feature**: essa è una sinossi di un insieme di vettori, aggiornabile in tempo e spazio costanti. Dato un insieme di vettori d-dimensionali:

$$X = \{\overrightarrow{x}_i\}, \quad i = 1, \ldots, N \quad \overrightarrow{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$$

la sua clustering feature CF_X è una tripletta:

$$CF_X = (N, \overrightarrow{LS}, SS) \tag{4.17}$$

dove:

- N è la cardinalità dell'insieme
- $\overrightarrow{LS}$ è il vettore somma dell'insieme:

$$\overrightarrow{LS} = \sum_{i=1}^{N} \overrightarrow{x}_i$$

- SS è la somma delle norme al quadrato dei suoi vettori

$$SS = \sum_{i=1}^{N} \|\overrightarrow{x}_i\|^2$$

Le clustering feature sono *additive*: se $CF_1 = (N_1, \overrightarrow{LS}_1, SS_1), CF_2 = (N_2, \overrightarrow{LS}_2, SS_2)$ sono le clustering feature di due insiemi di vettori A_1 e A_2 allora la clustering feature di $A_1 \cup A_2$ è:

$$CF_1 + CF_2 = (N_1 + N_2, \overrightarrow{LS}_1 + \overrightarrow{LS}_2, SS_1 + SS_2)$$

Ciò permette di aggiornare le CF di un cluster in modo efficiente. Ad esempio nel passo di aggiornamento della sinossi dell'algoritmo a passo unico, se C è un cluster tale che $CF_C = (N, \overrightarrow{LS}, SS)$ e $\overrightarrow{x}$ è il vettore letto, allora

$$CF_{C \cup \{\overrightarrow{x}\}} = CF_C + CF_{\overrightarrow{x}} = (N + 1, \overrightarrow{LS} + \overrightarrow{x}, SS + \|\overrightarrow{x}\|^2)$$

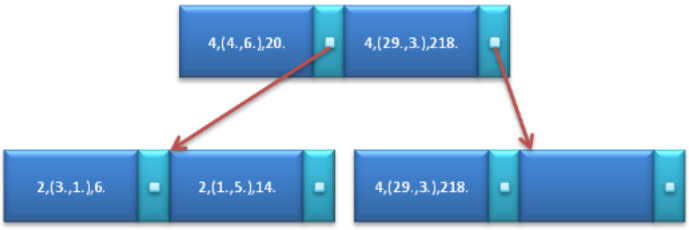


Figura 4.11: CF-tree.

CF tree Un CF Tree è una struttura dati ad albero per la memorizzazione di Clustering features da un dataset:

- i nodi sono liste di CF
- i nodi hanno dimensione fissa in byte, pertanto contengono un numero fisso di CF
- è bilanciato in altezza: dimensioni e altezza dell'albero dipendono da due parametri:
  1. fattore di branching B: il massimo numero di nodi figli
  2. soglia T : un limite alla variabilità interna all'insieme rappresentato da una CF
- ogni nodo **non foglia** contiene al piú B elementi `e[i]` ciascuno composto da (figura 4.11) una clustering feature CF e un puntatore a un nodo ptr: CF è la somma di tutte le clustering feature del nodo `(*e[i].ptr)` indirizzato da ptr.
- in merito ai nodi foglia:
  - rappresentano l'unione degli insiemi di vettori rappresentati dalle sue CF
  - contengono al massimo L clustering feature
  - sono parte di una lista doppio collegamento (per facilitare la lettura di nodi in successione)
  - ogni CF in un nodo foglia deve rappresentare un insieme di vettori con variabilità limitata: una predeterminata misura di variabilità deve essere inferiore alla soglia T.
    T influenza l'altezza dell'albero e il numero dei suoi nodi: maggiore T, minore il numero di nodi, minore l'altezza

Creazione di un CF tree Il CF tree viene costruito (figura 4.12) incrementalmente da zero come un albero di ricerca a B vie, mediante l'inserimento di un vettore alla volta in esattamente una foglia.
L'inserimento inizia alla radice e percorre l'albero seguendo i puntatori ai nodi fino a una foglia.
Durante la percorrenza, gli elementi dei nodi sono impiegati per ridurre lo spazio di ricerca:

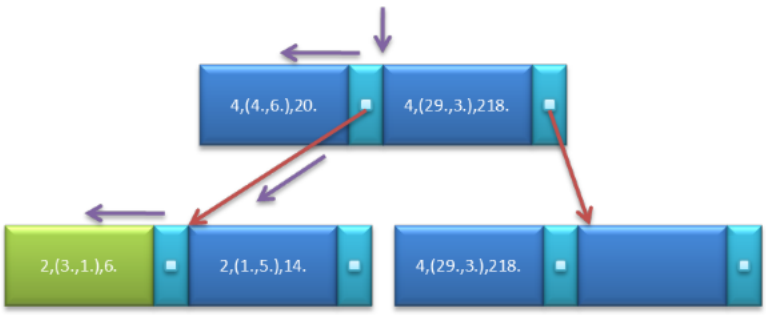


Figura 4.12: CF-tree.

- il nodo successivo nella percorrenza è quello indirizzato dall'elemento contenente la CF piú prossima al vettore
- la prossimità è basata su di una funzione distanza tra insiemi di vettori, calcolabile utilizzando le CF

Algoritmo di split del CF tree Quando una foglia si riempie fino alla propria massima capacità L, un inserimento nella foglia che richiede l'allocazione di una nuova CF scatena processo di splitting eventualmente ricorsivo.
Una nuova foglia è creata; gli elementi della foglia piena e la nuova CF sono ridistribuiti nelle due foglie:

- gli elementi con le piú lontane CF sono scelti come riferimento
- uno degli elementi di riferimento è scritto nella vecchia foglia, l'altro nella nuova
- infine, ogni elemento restante è scritto nella foglia con il riferimento piú prossimo
- la nuova foglia è indirizzata da un nuovo elemento nel nodo genitore della vecchia foglia

Misure di variabilità per un cluster Raggio:

$$R = \sqrt{\frac{\sum_{i=1}^{N} \|x_i - x_0\|^2}{N}}$$

Diametro:

$$D = \sqrt{\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \|x_i - x_j\|^2}{N(N-1)}}$$

4.5.2.2 Funzionamento algoritmo

Si sviluppa nelle seguenti fasi:

1. costruzione del CF Tree in memoria interna

2. *condensazione* (opzionale) dei dati in un intervallo ottimale per il clustering della fase 3, mediante costruzione di un CF Tree piú piccolo
3. clustering agglomerativo dei subcluster rappresentati dalle CF delle foglie
4. fase di *refinement* (opzionale) costituita da
   - ridistribuzione dei vettori rispetto ai centroidi dei cluster ottenuti nella fase 3
   - costruzione di un encoder
   - eliminazione degli outlier, cioè dei vettori troppo lontani dal centroide del proprio cluster

I motivi/benefici della costruzione di un CF tree (fase 1):

- la fase 1 crea una rappresentazione compressa in memoria interna dei dati con proprietà utili
- le operazioni di I/O non sono piú necessarie: il problema del clustering dei dati originali si riduce al problema del clustering dei subcluster delle foglie, cioè a un problema di clustering di CF
- le fasi rimanenti sono:
  - piú accurate perché molti outlier sono stati eliminati e i dati rimanenti riflettono le caratteristiche dei cluster, con una accuratezza dipendente dalla memoria disponibile
  - meno sensibili all'ordinamento perché l'ordinamento dei subcluster nelle foglie ha migliori caratteristiche di località dei dati originari

4.6 Valutazione/validazione del clustering

Nella classificazione supervisionata la valutazione del modello di classificazione risultante è parte integrante del processo di sviluppo e vi sono misure e procedure di valutazione accettate (ad esempio accuratezza, cross validazione).
Nel campo del learning non supervisionato la valutazione non è così sviluppata/utilizzata, sebbene resti importante, dato che qualsiasi algoritmo di clustering in un modo o nell'altro trova dei cluster in un data set, anche se quest'ultimo non ha una struttura a cluster naturale.

Vi sono diversi *aspetti da considerare* per la validazione:

1. determinare la clustering tendency, ovvero se una struttura non casuale effettivamente sia presente
2. determinare il corretto numero di cluster
3. valutare quanto bene i risulti di un clustering fittino i dati senza far riferimento ad informazione esterna
4. confrontare i risultati di un clustering a risultati esterni conosciuti (ad esempio una classificazione esterna)
5. confrontare due clustering per determinare quale sia il migliore

Le misure applicate per giudicare vari aspetti del clustering sono tradizionalmente classificate in:

- *non supervisionate*: misurano la bontà della struttura di clustering senza riferimento ad informazione esterna (un esempio ne è il SSE). Queste misure sono ulteriormente suddivise in due tipi:
  - misure di *cluster cohesion*: misurano quanto sono relazionati gli oggetti appartenenti ad un cluster
  - misure di *cluster separation*: misurano quanto ben distinto/separato sia un determinato cluster dai rimanenti
- *supervisionate*: misurano quanto il la struttura di clustering matchi una struttura/informazione esterna (un esempio di questa è l'entropia)
- *relative*: confrontano clustering o clusters diversi. Di fatto si tratta di confrontare due misure di performance tipo quelle di cui prima; ad esempio rapporto tra SSE o tra entropie per determinare il clustering che performa meglio.

4.6.1 Misure per cluster partizionali

4.6.1.1 Valutazione mediante coesione e separazione

Molte misure di validità interna per clustering sono basate sui concetti di coesione o separazione. In questa sezione usiamo misure di validità per tecniche di clustering basate su prototipo o su grafo

Coesione e separazione nell'approccio graph based Coesione di un cluster può essere definita come la somma dei pesi dei link nel grafo che connette i punti entro un cluster; formalmente

$$cohesion(C_i) = \sum_{\mathbf{x} \in C_i, \mathbf{y} \in C_i} proximity(\mathbf{x}, \mathbf{y}) \tag{4.18}$$

dove la funzione di prossimità può essereuna similarità, una dissimilarità o una funzione di queste quantità.
La separazione può essere misurata dalla somma dei pesi dei link dai punti in un cluster a punti in altri cluster

$$separation(C_i, C_j) = \sum_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} proximity(\mathbf{x}, \mathbf{y}) \tag{4.19}$$

Coesione e separazione nell'approccio con prototipo In questo approccio la coesione di un cluster può essere definita come la somma delle prossimità ripetto al prototipo (centroide o medoide) del cluster:

$$cohesion(C_i) = \sum_{\mathbf{x} \in C_i} proximity(\mathbf{x}, \mathbf{c}_i) \tag{4.20}$$

si noti che quella di sopra è il SSE del cluster se impostiamo la prossimità alla distanza euclidea quadrata.

Nome	Misura del cluster	Peso del cluster	Tipo	Approccio
$\mathcal{I}_1$	$\sum_{\mathbf{x}\in C_i,\mathbf{y}\in C_i} prox(\mathbf{x},\mathbf{y})$	$\frac{1}{m_i}$	coesione	graph
$\mathcal{I}_2$	$\sum_{\mathbf{x}\in C_i} prox(\mathbf{x},\mathbf{c}_i)$	1	coesione	prototipo
$\mathcal{E}_1$	$prox(\mathbf{c}_i,\mathbf{c})$	m_i	separazione	prototipo
$\mathcal{G}_1$	$\sum_{j=1,j\neq i}^{k}\sum_{\mathbf{x}\in C_i,\mathbf{y}\in C_j} prox(\mathbf{x},\mathbf{y})$	$\frac{1}{\sum_{\mathbf{x}\in C_i,\mathbf{y}\in C_i} prox(\mathbf{x},\mathbf{y})}$	sep. e coes.	graph

Tabella 4.2: Misure di coesione e separazione complessiva secondo differenti approcci

Similmente la separazione può essere misurata dalla prossimità dei due prototipi che rappresentano altrettanti cluster. Vi sono due misure di separazione poiché la separazione di un prototipo di cluster dal prototipo overall è a volte correlato alla separazione del prototipo di cluster dagli altri prototipi:

$$separation(C_i, C_j) = proximity(\mathbf{c}_i, \mathbf{c}_j) \tag{4.21}$$

$$separation(C_i) = proximity(\mathbf{c}_i, \mathbf{c}) \tag{4.22}$$

Misure di coesione/separazione complessive Possiamo combinare le misure di coesione separazione rispettivamente in un unica misura di validità come somma pesata, ovvero in generale della forma

$$overallValidity = \sum_{i=1}^{K} w_i validity(C_i) \tag{4.23}$$

come peso w_i possiamo utilizzare varie cose (spesso comunque legate alla dimensione del cluster), che danno origine insieme alla misura di validità a diverse misure possibili (tabella 4.2).

Migliorare il clustering Possiamo sfruttare le informazioni su un singolo cluster per migliorare la qualità del clustering; ad esempio con i dati di cohesion alla mano possiamo decidere di splittare un cluster se non molto coeso, o con dati di separazione decidere di mergiare dei cluster vicini.

Valutazioni entro singolo cluster Possiamo anche valutare signoli oggetti in merito al contributo che portano a coesione e separazione del cluster; oggetti che contribuiscono alla coesione e separazione sono più interni al cluster, mentre quelli all'opposto sono probabilmente in prossimità dei bordi.
Il coefficiente di silhouette si riferisce al singolo oggetto e combina siacoesione che separazione. Il processo per calcolarlo segue questi step:

1. per l'i-esimo oggetto calcolare la distanza media da tutti gli altri oggetti nel cluster. Chiamiamo il valore a_i
2. per l'i-esimo oggetto e qualsiasi cluster che non lo contenga, calcolare la distanza media dell'oggetto da tutti quelli dei cluster considerati. individuare il minimo di tale distanza e chiamarlo b_i

3. il coefficiente di silhouette per l'i-esimo oggett è

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{4.24}$$

Il coefficiente può variare tra -1 e 1 un valore negativo è non desiderabile (a_i è maggiore di b_i); l'ideale è che il coefficiente sia positivo e che a_i sia il più posssibile vicina a 0 (in corrispondenza del quale il coefficiente assume il suo massimo).
Si può poi calcolare il coefficiene di silhouette medio di un cluster e quello complessivo: quest'ultimo costituisce una misura complessiva di bontà del clustering.

4.6.1.2 Valutazione mediante matrice di prossimità

Ipotizzando di avere le label di cluster da una attività di clustering, poniamo le nostre unità in ordine dato dall'appartenenza ai vari cluster, dopodiché calcoliamo la matrice di similarità tra tutti gli elementi del dataset (posti nell'ordine di cui prima).
Idealmente la similarity dovrebbe essere massima tra gli oggetti entro cluster e minore con quelli fuori; possiamo allora:

- calcolare la correlazione con la matrice di prossimità perfetta (solamente la parte superiore o inferiore è uguale): ha una struttura a blocchi e presenta valore 1 nelle celle che corrispondono ad elementi appartenenti alla stessa classe e 0 nelle altre. Più la correlazione è alta più la matrice di similarità empirica assomiglia a quella ideale teorica

- plottare la matrice di correlazione con colore proporzionale alla correlazione; dovrebbe emergere un pattern in cui i blocchi sulla diagonale sono più scuri (maggiore correlazione entro cluster)

4.6.2 Misure per cluster gerarchici

La **distanza cofenetica** tra due oggetti è definita come la prossimità alla quale un algoritmo di clustering gerarchico agglomerativo pone per la prima volta i due oggetti nello stesso cluster. La *matrice di distanza cofenetica* è calcolata mediante il calcolo di tutte le distanze cofenetiche delle coppie di elementi
Il **coefficiente di correlazione cofenetica** è la correlazione tra gli elementi della matrice di distanza cofenetica e la matrice di dissimilarità originale, ed è una misura di quanto bene un clustering gerarchico di un certo tipo fitti bene i dati.
Il confronto dei coefficienti cofenetici tra diversi metodi di clustering gerarchico permette la scelta dell'algoritmo che in un dato dataset performa meglio.

4.6.3 Determinare il corretto numero di cluster

Possiamo usare diversi approcci:

- possiamo plottare l'SSE in funzione del numero di cluster imposti in una tecnica a-la k-means

- possiamo plottare il coefficiente di silhouette medio, sempre in funzione del numero di cluster

In entrambi i casi dovremmo notare una diminuzione di SSE/coefficiente di silhouette e poi un plateau; il punto in cui si raggiunge il plateau è una indicazione per il numero di cluster da adottare

4.6.4 Clustering tendency

Può essere pensata come l'effettiva presenza di cluster nel dataset; per valutarla potremmo alternativamente:

- utilizzare diversi algoritmi di clustering (che di fatto possono mettere in luce cluster di tipo differente) e affermare che vi sia clustering tendency solamente se almeno uno dei clustering effettuati risulta di buona qualità
- valutare senza effettuare clustering: si utilizzano test per la randomicità spaziale. Un esempio è la statistica di Hopkins:
  - generiamo p punti distribuiti casualmente (uniformemente, credo) nello spazio dei dati
  - campioniamo/scegliamo p punti effettivi del nostro campione
  - per ciascuno insieme di punti troviamo la distanza dal nearest neighbor nel dataset originale; la distanza dei punti generati è indicata con u_i, quella dei punti che appartenevano effettivamente al campione w_i
  - calcoliamo la seguente statistica

$$H = \frac{\sum_{i=1}^{p} w_i}{\sum_{i=1}^{p} w_i + \sum_{i=1}^{p} u_i} \tag{4.25}$$

    La misura è compresa tra 0 e 1; se i punti generati casualmente hanno all incirca la stessa distanza che quelli estratti allora la misura sarà circa 0.5. 0 significa che i dati sono fortemente clustered ($\sum w_i$ di molto inferiore a $\sum u_i$) mentre se =1 i dati sono regolarmente distribuiti nello spazio

- si ripete l'estrazione casuale e il calcolo per creare variabilità nella stima e poter generare un intervallo di confidenza

4.6.5 Misure supervisionate di validità

Qualora sia disponibile una classificazione esterna possiamo usarla per valutare il nostro clustering se alternativamente costituisce un gold standard di qualche sorta.
Vi sono due tipi di approcci:

- *classification oriented*: si possono usare tecniche porovenienti dalla classificazione, che valutano il livello con cui un cluster contiene oggetti dello stesso tipo
- *similarity oriented*: usare misure di simiglianza per dati qualitativi

Questo per quanto riguarda il clustering non gerarchico; vi sono poi misure specifiche per il clustering gerarchico

4.6.5.1 Misure classification oriented

- **entropia**: misura il grado con cui ogni cluster sia costituito da oggetti di una signgola classe.
  Per il cluster i-esimo calcoliamo p_{ij}, la probabilità che un membro del cluster i appartenga alla classe j, ovvero $p_{ij} = m_{ij}/m_i$ dove m_i è il numero di oggetti nel cluster i e m_{ij} è il numero di oggetti della classe j nel cluster i.
  Facendo uso di questa distribnuzione, l'entropia di ciascun cluster i è calcolata usando la formula standard $e_i = -\sum_{j=1}^{L} p_{ij} \log_2 p_{ij}$, dove L è il numero di classi.
  L'entropia totale è la somma delle entropie pesate per la dimensione del cluster, ovvero $e = \sum_{i=1}^{K} \frac{m_i}{m} e_i$ dove K è il numero di cluster e m è il numero complerssivo di punti.

- **purezza**: un'altra misura di quanto un cluster contenga oggetti di una singola class. La purezza del singolo cluster i è $p_i = \max_j p_i j$, la purezza complessiva di un clustering è $purity = \sum_{i=1}^{K} \frac{m_i}{m} p_i$

- **precisione**: la proporzione di un cluster che consta in oggetti di una specificata classe. La precisione del cluster i con riferimento alla classe j è $precision(i, j) = p_{ij}$

- **recall**: il livello con cui un cluster contiene tutti gli oggetti di una data classe. Il recall del cluster i rispetto alla classe j è

  $$recall(i, j) = m_{ij}/m_j$$

  dove m_j è il numero di oggetti nella classe j.

- **misura F**: una combinazione di precisione e recall che misura il livello con cui un cluster contiene solamente oggetti di una classe particolare e tutti gli oggetti di tale classe. La F-measure del cluster i rispetto alla classe j è

  $$F(i, j) = \frac{2 \cdot precision(i, j) \cdot recall(i, j)}{precision(i, j) + recall(i, j)}$$

4.6.5.2 Misure similarity oriented

Possiamo confrontare due matrici:

- la **matrice di similarità del cluster** ideale che ha 1 nelle cella ij-esima se gli oggetti$i$ e j appartengono allo stesso cluster e 0 altrimenti

- la **matrice di similarità di classe** ideale che ha 1 nelle cella ij-esima se gli oggetti$i$ e j appartengono allo stessa classe e 0 altrimenti

Possiamo prendere il coefficiente di correlazione di queste due matrici, che è detto Λ statistic.

Più generalmente possiamo fare uso di misure di similarità binaria, calcolando:

- f_{00}: numero di coppie di oggetti che hanno differente classe e differente cluster

- f_{01}: numero di coppie di oggetti che hanno differente classe e stesso cluster
- f_{10}: numero di coppie di oggetti che hanno stessa classe e differente cluster
- f_{11}: numero di coppie di oggetti che hanno stessa classe e stesso cluster

I due indici più usati sono la statistica di Rand (è poi il matching coefficient)

$$Randstatistic = \frac{f_{00} + f_{11}}{f_{00} + f_{01} + f_{10} + f_{11}}$$

e il coefficiente di Jaccard

$$Randstatistic = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}$$

4.6.5.3 Misure per il clustering gerarchico

La valutazione di un clustering gerarchico è più difficile per diverse motivazioni, in primis spesso la non disponibilità di una struttura gerarchica (esterna da poter confrontare). Qui diamo un approccio di valutazione in termini di un set di label categoriche flat che molto più probabilmente sono disponibili.
L'idea è valutare se il clustering contiene, per ogni classe almeno un cluster che sia relativamente puro e includa la maggior parte di oggetti di tale classe.
Per fare ciò:

- calcoliamo, per ogni classe, la misura F per ogni cluster nella gerarchia dei cluster
- per ogni classe prendiamo il massimo F ottenuto nei cluster
- calcoliamo un F overall per il clustering gerarchico mediante una media pesata di tutte le misure F per classe, dove i pesi sono basati sulla dimensione di classe:

  $$F = \sum_j \frac{m_j}{m} \max_i F(i, j)$$

  dove il massimo è preso su tutti i cluster i a tutti i livelli, m_j è il numero di oggetti nella classe j e m è il numero totale di oggetti.