UNIVERSITÀ DEGLI STUDI DI PADOVA
FACOLTÀ DI SCIENZE STATISTICHE

Corso di Laurea Specialistica
in STATISTICA E INFORMATICA

TESI DI LAUREA

STIMA DEL NUMERO DI CLUSTER

LAUREANDO:
Nicola Scarso

RELATORE:
Ch.mo Prof. Giancarlo Diana

Anno Accademico 2008/2009

Alla mia famiglia

Indice

Prefazione

La cluster analisi è un importante strumento per esplorare la struttura presente nei dati, che viene applicata in una varietà di discipline sia scientifiche che ingegneristiche. Può essere definita come un insieme di metodi e procedure finalizzati al raggruppamento degli oggetti in categorie o classi, in base a delle caratteristiche di somiglianza.

Il termine *clustering* è tipicamente usato per descrivere il partizionamento di un insieme di oggetti in gruppi o cluster ("grappoli"), in modo tale che gli oggetti siano simili all'interno dello stesso cluster e diversi in cluster differenti.

Uno dei problemi comuni a tutte le tecniche di *clustering*, è la difficoltà nel decidere il numero di cluster presenti in un certo insieme di dati, in quanto non si hanno a disposizione classi predefinite o altre informazioni, che possano aiutare a classificare le osservazioni. É proprio da questa difficoltà che è nata l'idea di questo lavoro, con l'intenzione di offrire una procedura che sia in grado di aiutare nella risoluzione di questo problema.

Nel primo capitolo, si introducono concetti generali sulla cluster analisi, e la distinzione tra metodi di raggruppamento gerarchici e non gerarchici.

Nel secondo capitolo, vengono presentati i principali metodi per individuare il numero di cluster in un data set e la distinzione tra indici interni, che sono funzioni della somma dei quadrati entro i cluster e/o tra i cluster, e indici esterni che determinano una misura di accordo tra due partizioni.

Nel terzo capitolo, vengono introdotti tre nuovi metodi, a partire dal metodo jump, descritto nel secondo capitolo: jump-differenza, jump con Γ stimata e jump-differenza con Γ stimata, dove Γ rappresenta la matrice di covarianza comune. Queste nuove proposte sono state introdotte per tentare di migliorare la performance rispetto al metodo da cui derivano.

Nel quarto capitolo, si riportano i risultati di simulazione per sette diversi sce-

nari, e viene analizzato il comportamento dei metodi migliori su due dataset reali desunti dalla letteratura.
Infine, nell'appendice vengono riportate le funzioni che sono state utilizzate.

Capitolo 1

Introduzione

Lo scopo della classificazione è l'assegnazione di oggetti a classi sulla base di misurazioni fatte sugli stessi. Ci sono due principali aspetti della classificazione:

- analisi discriminante o *supervised learning*;
- cluster analisi o *unsupervised learning*.

Mentre nell'analisi discriminante le classi sono definite, e si usa l'informazione presente in un insieme di oggetti training per classificare osservazioni future, nella cluster analisi non c'è un sistema di classificazione pre-assegnato, ossia non c'è una variabile risposta di riferimento che indica a quale classe assegnare una determinata osservazione. Questo vuol dire non avere nessuna indicazione sul numero e sulla natura dei gruppi e si cerca un metodo per formarli a partire dalle variabili disponibili. Il presente lavoro si concentra proprio su quest'ultimo importante problema.
L'obiettivo della cluster analisi è semplicemente quello di trovare una conveniente e valida organizzazione dei dati, e non di stabilire regole per classificare osservazioni future.

1.1 Cluster analisi

Kaufman e Rousseeuw (1990) definiscono la cluster analisi come "l'arte di trovare gruppi nei dati". Da tale definizione si percepisce subito che stimare il numero di gruppi in un data set è un passo cruciale per questo tipo di analisi e non sempre semplice. Per capire cosa significa questo, si osservino le figure 1.1 e 1.2. Nella figura 1.1 si riportano 60 osservazioni misurate su due variabili. Si possono osservare due gruppi ben distinti di 30 osservazioni ciascuno.

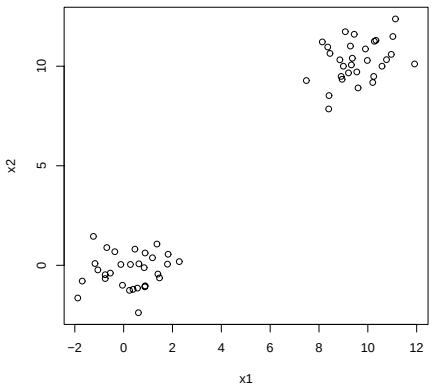

Figura 1.1: 2 cluster da distribuzione normale con vettori medi $(0,0)'$, $(10,10)'$ e con matrice I_2 di varianza/covarianza.

Nella figura 1.2 si riportano, invece, 20 osservazioni per ciascuno di tre gruppi. A differenza della figura 1.1 non è ben chiara la distinzione tra i gruppi, e persone differenti potrebbero avere opinioni diverse sul loro numero.
I gruppi sono chiamati cluster, e la loro "scoperta" è lo scopo della cluster analisi.
Generalmente, uno vuole formare cluster in modo tale che gli oggetti in uno stesso cluster siano simili ad ogni altro, mentre oggetti in cluster diversi devono essere il più possibile dissimili.
La classificazione di oggetti simili in gruppi è un importante attività umana. Ogni giorno, questo fa parte di un processo di apprendimento: i bambini imparano a distinguere tra cani e gatti, tra tavole e sedie, tra uomini e donne, migliorando continuamente la loro capacità a schemi di classificazione.
Un esempio tipico, nel campo del marketing, è quello della "segmentazione" della

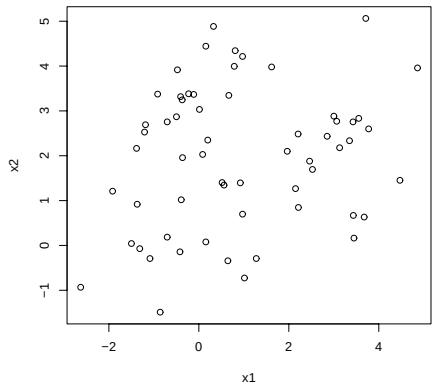

Figura 1.2: 3 cluster da distribuzione normale con vettori medi $(0,0)'$, $(3,2)'$, $(0,3)'$ e con matrice I_2 di varianza/covarianza.

clientela di un'azienda. Dalle informazioni raccolte sull'utilizzo dei prodotti, dati anagrafici, risposte a questionari si perviene alla formazione di gruppi di clienti, detti "segmenti". Per identificare ciascun segmento è importante definire quali siano gli aspetti salienti ossia, in altre parole, le caratteristiche che un cliente deve possedere per far parte di un gruppo piuttosto che di un altro. Nelle scienze sociali, si classificano le persone in base al loro comportamento e alle loro preferenze. Altri esempi possono riguardare la geografia (classificazione di regioni), l'archeologia (classificazione di reperti archeologici), la medicina (classificazione di malattie), le piante, gli animali e così via.
Marriott (1971) ha specificato le seguenti due domande, alle quali deve essere data nell'ordine una risposta, per cercare di arrivare ad una soluzione soddisfacente del problema:

a) Qual è la migliore suddivisione delle osservazioni di un campione in un dato numero, diciamo k, di gruppi?

b) Qual è il migliore valore di k?

Una possibile risposta alla domanda a) è la suddivisione che minimizza la variabilità entro i gruppi, misurata da una appropriata funzione obiettivo. Per rispondere alla domanda b) bisogna suddividere il campione in gruppi, utilizzando diversi valori di $k = 2, 3, 4, ...$, e selezionare quel valore di k per cui la

partizione finale sembra essere la migliore. Tale partizione sarà individuata tramite l'ottimizzazione di un criterio o metodo, come quelli descritti nel prossimo capitolo.
É evidente, per quanto detto, che la similarità (o dissimilarità) è fondamentale per la definizione di un cluster. Le possibili scelte per come quantificarla sono tantissime e variano sia con la natura delle variabili in gioco (continue, binarie, nominali, ordinali, ecc.), sia con gli obiettivi che ci si prefigge.
Gli algoritmi di clustering presenti in letteratura cercano di misurare la similarità tra oggetti, e poi procedono nel raggruppare gli oggetti con bassa dissimilarità e separare quelli con alta dissimilarità. Questi algoritmi vengono generalmente classificati in metodi di raggruppamento gerarchici e metodi di raggruppamento non gerarchici.

1.2 Metodi di raggruppamento gerarchici

In questo tipo di metodi i dati vengono associati ad una struttura ad albero in modo tale che le foglie dell'albero corrispondano alle osservazioni e i nodi a sottoinsiemi di osservazioni. La stessa natura di albero introduce una gerarchia nei sottoinsiemi associati ai rami.
Esistono due ampie famiglie di metodi gerarchici:

- **metodi agglomerativi**: partendo da uno stato iniziale di n gruppi, in cui ogni osservazione fa gruppo a sè, si procede effettuando successive fusioni di gruppi con bassa dissimilarità tra loro, fino a che $k = 1$, cioè tutte le osservazioni appartengano ad uno stesso gruppo;

- **metodi divisivi**: partendo da uno stato iniziale con $k = 1$ gruppi, ossia da un unico gruppo, si procede per suddivisioni successive fino ad arrivare a n gruppi.

Entrambi i procedimenti operano su una matrice di dissimilarità. La caratteristica principale di questo tipo di metodi è che una volta che due gruppi sono stati uniti non saranno più separati in seguito e, in modo analogo, una volta che due gruppi vengono separati non faranno più parte dello stesso cluster. Inoltre, applicando un algoritmo di questo tipo, si usa lo stesso albero per tutti i valori

di k, facendo riferimento ogni volta a un livello diverso dell'albero stesso. Si tratta quindi di una struttura rigida.
I metodi per la valutazione delle distanze tra i gruppi ai fini delle aggregazioni o divisioni successive sono:

- **Metodo del legame singolo** (Single Linkage): la distanza tra i gruppi è definita come la minima tra tutte le distanze che si possono calcolare tra ciascuna unità i di un gruppo e ciascuna unità j di un altro gruppo.

- **Metodo del legame completo** (Complete Linkage): la distanza tra i gruppi è definita come la massima distanza tra tutte le distanze che si possono calcolare tra ciascuna unità i di un gruppo e ciascuna unità j di un altro gruppo.

- **Metodo del legame medio** (Average Linkage): la distanza tra i gruppi è definita come la media tra tutte le distanze che si possono calcolare tra due gruppi.

- **Metodo del centroide**: la distanza tra i gruppi è definita dalla distanza tra i loro centroidi, che sono i vettori dei valori medi delle p variabili nei gruppi.

- **Metodo di Ward**: si basa sulla scomposizione della devianza totale nella somma delle devianze entro i cluster e tra i cluster. Nel passare da $k+1$ a k gruppi (aggregazione) la devianza entro i cluster aumenta, mentre la devianza tra i cluster diminuisce. Ad ogni passo, il metodo di Ward aggrega tra loro i gruppi per cui vi è il minor incremento della devianza entro i gruppi.

Esempi di metodi gerarchici sono AGNES e DIANA (Kaufman e Rousseeuw, 1990).

1.3 Metodi di raggruppamento non gerarchici

Questi metodi sono caratterizzati da un procedimento che deve individuare dei punti di aggregazione, detti **centroidi**, attorno ai quali costruire dei gruppi. Ogni osservazione del data set viene assegnata al gruppo del centroide più vicino.

A differenza dei metodi gerarchici, i metodi non gerarchici costruiscono una partizione alla volta e quindi bisogna ripetere l'algoritmo per ogni valore di k. L'individuazione della partizione ottimale comporterebbe l'esame di tutte le configurazioni possibili in k gruppi del data set. Dato l'elevato numero di raggruppamenti possibili, soprattutto all'aumentare del numero n di osservazioni, bisogna ricorrere a degli algoritmi sub-ottimi. Questo vuol dire, una volta fissati k e la posizione iniziale dei centroidi, minimizzare una prefissata funzione obiettivo. L'algoritmo infatti procede in modo iterativo raggruppando successivamente le osservazioni intorno ai centroidi, soggetti anch'essi ad aggiornamento iterativo, fino ad un punto di convergenza. La convergenza è assicurata, ma non è detto che corrisponda ad un minimo assoluto della funzione obiettivo.
I due metodi non gerarchici più conosciuti e adatti alle sole variabili numeriche sono il metodo *k-means* (MacQueen, 1967) e il metodo PAM (Kaufman e Rousseeuw, 1990).
Dato che in questo tipo di metodi bisogna definire preventivamente il numero k di cluster e tale numero è difficile da individuare tramite un'analisi grafica preliminare dei dati, si presentano nel prossimo capitolo i principali metodi per la ricerca di k.

Capitolo 2

Metodi per stimare il numero di cluster

Come è noto, per valutare i risultati di un algoritmo di clustering, abitualmente si fa ricorso a degli indici che possono portare a una decisione corretta o non corretta. Ne deriva, come sottolineato da Milligan e Cooper (1985), che ci siano due tipi di errore:

- 1° tipo: si presenta quando un indice segnala più cluster di quelli effettivamente presenti nei dati (sovrastima);
- 2° tipo: si presenta quando un indice segnala pochi cluster rispetto a quelli presenti nei dati (sottostima).

Sebbene la severità di questi tipi di errore potrebbe cambiare in riferimento al contesto del problema, quello di 2° tipo potrebbe essere considerato più grave in qualche analisi, perché ci potrebbe essere una perdita di informazione unendo tra di loro cluster distinti.
Gli indici generalmente vengono classificati in **indici interni** e **indici esterni**.

Gli **indici interni** sono tipicamente funzioni della somma dei quadrati entro i cluster e/o tra i cluster. Vengono detti interni nel senso che sono calcolati a partire dalle stesse osservazioni che sono usate per creare il clustering. Esempi di tali indici, che verranno illustrati in dettaglio nei paragrafi successivi, sono l'indice di Calinski e Harabasz, l'indice di Hartigan, l'indice di Krzanowski e Lai,

la statistica Gap, la statistica silhouette, la Split Silhouette Media, il metodo jump, la statistica Gap pesata e la statistica DD-Gap pesata.

Gli **indici esterni** invece determinano una misura di accordo tra due partizioni: la prima $U = \{u_1, ..., u_R\}$ è una struttura di dati pre-specificata, mentre la seconda $V = \{v_1, ..., v_C\}$ è il risultato di una procedura di clustering. Tale accordo può essere espresso attraverso una tabella di contingenza (tabella 2.1), dove n_{ij} denota il numero di osservazioni che fanno parte dei cluster u_i e v_j. Siano inoltre $n_{i.} = \sum_{j=1}^{C} n_{ij}$ e $n_{.}j = \sum_{i=1}^{R} n_{ij}$ le somme per riga e per colonna di tale tabella, ossia il numero di osservazioni nei gruppi u_i e v_j, e $Z = \sum_{i=1}^{R} \sum_{j=1}^{C} n_{ij}^2$.

	v_1	v_2	...	v_C	
u_1	n_{11}	n_{12}	...	n_{1C}	$n_{1.}$
u_2	n_{21}	n_{22}	...	n_{2C}	$n_{2.}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
u_R	n_{R1}	n_{R2}	...	n_{RC}	$n_{R.}$
	$n_{.1}$	$n_{.2}$	...	$n_{.C}$	n

Tabella 2.1: tabella di contingenza per 2 partizioni di n osservazioni

Alcuni indici comunemente usati per calcolare la similarità fra due partizioni, sono riportati in tabella 2.2.

Rand	$1 + \left(Z - (1/2)(\sum_{i=1}^{R} n_{i.}^2 + \sum_{j=1}^{C} n_{.j}^2) \right) / \binom{n}{2}$
Jaccard	$(Z - n) \,/\, \left(\sum_{i=1}^{R} n_{i.}^2 + \sum_{j=1}^{C} n_{.j}^2 - Z - n \right)$
Fowlkes e Mallows	$(1/2)\,(Z - n) \,/\, \left[\sum_{i=1}^{R} \binom{n_{i.}}{2} \sum_{j=1}^{C} \binom{n_{.j}}{2} \right]^{1/2}$

Tabella 2.2: indici esterni per partizioni

Hubert e Arabie (1985) hanno proposto una correzione per l'indice Rand, in modo che assuma valore 0 quando le partizioni sono determinate casualmente, e 1 quando vi è perfetta corrispondenza tra esse. L'espressione diventa quindi:

$$Rand\ \mathrm{A} = \frac{\sum_{i=1}^{R} \sum_{j=1}^{C} \binom{n_{ij}}{2} - \left[1/\binom{n}{2}\right] \sum_{i=1}^{R} \binom{n_{i.}}{2} \sum_{j=1}^{C} \binom{n_{.j}}{2}}{(1/2) \left[\sum_{i=1}^{R} \binom{n_{i.}}{2} + \sum_{j=1}^{C} \binom{n_{.j}}{2} \right] - \left[1/\binom{n}{2}\right] \sum_{i=1}^{R} \binom{n_{i.}}{2} \sum_{j=1}^{C} \binom{n_{.j}}{2}}$$

Un altro modo per esprimere la similarità tra due partizioni, è la considerazione delle seguenti funzioni indicatrici ($x_1, ..., x_n$ rappresentano le n osservazioni):

- $I_U(i,j) = 1$ se x_i e x_j, con $i \neq j$, appartengono allo stesso cluster u_r, per $1 \leq r \leq R$, e 0 altrimenti;
- $I_V(i,j) = 1$ se x_i e x_j, con $i \neq j$, appartengono allo stesso cluster v_c, per $1 \leq c \leq C$, e 0 altrimenti.

I quattro possibili risultati che si possono ottenere da queste due funzioni, sono riassunti nella tabella 2.3.

		I_V	
		1	0
I_U	1	a	b
	0	c	d

Tabella 2.3: tabella di contingenza tra due funzioni indicatrici

Dalla tabella si nota che:

- a è il numero di coppie di osservazioni che sono nello stesso cluster sia in U che in V;
- b è il numero di coppie di osservazioni che sono nello stesso cluster in U ma non in V;
- c è il numero di coppie di osservazioni che sono nello stesso cluster in V ma non in U;
- d è il numero di coppie di osservazioni che sono in differenti cluster sia in U che in V.

Il collegamento tra le tabelle 2.1 e 2.3 è quindi definito dalle seguenti equazioni:

- $a = \sum_{i=1}^{R} \sum_{j=1}^{C} \binom{n_{ij}}{2}$
- $b = \sum_{j=1}^{C} \binom{n_{.j}}{2} - \sum_{i=1}^{R} \sum_{j=1}^{C} \binom{n_{ij}}{2}$
- $c = \sum_{i=1}^{R} \binom{n_{i.}}{2} - \sum_{i=1}^{R} \sum_{j=1}^{C} \binom{n_{ij}}{2}$

- $d = \binom{n}{2} - a - b - c$

Di conseguenza, gli indici esterni definiti in tabella 2.2, possono anche assumere la forma riportata in tabella 2.4.

Rand	$\frac{a+d}{a+b+c+d}$
Jaccard	$\frac{a}{a+b+c}$
Fowlkes e Mallows	$\frac{a}{\sqrt{(a+b)(a+c)}}$

Tabella 2.4: indici esterni per partizioni

2.1 Indice di Calinski e Harabasz

Per ogni numero di cluster $k \geq 2$, l'indice di Calinski e Harabasz è definito come

$$ch(k) = \frac{tr\mathbf{B}_k/(k-1)}{tr\mathbf{W}_k/(n-k)} \tag{2.1}$$

dove n e k sono rispettivamente il numero totale di osservazioni e il numero di cluster. $tr\mathbf{B}_k$ è la traccia della matrice dei cluster tra i gruppi (**B**etween groups), mentre $tr\mathbf{W}_k$ è la traccia della matrice dei cluster entro i gruppi (**W**ithin groups).
Il numero di cluster ottimo si ottiene in corrispondenza del k per cui vi sono grandi dissimilitarità tra i cluster e grandi similarità entro i cluster: la soluzione è quindi il valore di k che massimizza $ch(k)$.

2.2 Indice di Hartigan

Nel criterio proposto da Hartigan si mette a confronto la variabilità dei dati entro k cluster, con quella entro $k+1$ cluster, ottenuta dividendo in due parti uno di quelli precedentemente formati. In realtà, un cluster verrà diviso in modo ottimale, se i cluster erano realmente distinti, e in generale non ci sarà una semplice relazione tra la partizione ottima a k gruppi e la partizione ottima a $k+1$ gruppi.
Per ogni numero di cluster $k \geq 1$, l'indice è definito come

$$h(k) = \left(\frac{tr\mathbf{W}_k}{tr\mathbf{W}_{k+1}} - 1\right)(n-k-1) \tag{2.2}$$

Il rapporto è una misura della riduzione della variabilità entro i cluster tra la partizione a k cluster e la partizione a $k+1$ cluster. Se le osservazioni provengono da una distribuzione normale, $h(k)$ si distribuisce come una F. Tuttavia, l'autore sottolinea che non è corretto usare la distribuzione F, e suggerisce che un cluster può essere aggiunto finchè $h(k) > 10$, e quindi di stimare il numero di cluster ottimo come il più piccolo $k \geq 1$ tale che $h(k) \leq 10$.

2.3 Indice di Krzanowski e Lai

Nel derivare una regola per determinare il numero di gruppi in un data set, Krzanowski e Lai hanno seguito l'approccio generale di Marriott (1971).
Essi mostrano che

$$k^{2/p} tr\mathbf{W}_k$$

(dove p è il numero di variabili) dovrebbe rimanere approssimativamente costante rispetto a k e si sceglie il valore di k che minimizza tale quantità. Tuttavia, dai risulati riportati nel loro articolo, emerge che in alcune situazioni potrebbe qualche volta succedere che il minimo valore non è $k = 1$ per dati omogenei o che $k^{2/p} tr\mathbf{W}_k$ decresce con k per dati fortemente raggruppati. Di conseguenza, invece di usare direttamente la funzione $k^{2/p} tr\mathbf{W}_k$, Krzanowski e Lai suggeriscono di usare differenze successive della stessa funzione, ossia

$$DIFF(k) = (k-1)^{2/p} tr\mathbf{W}_{k-1} - (k)^{2/p} tr\mathbf{W}_k$$

Se i dati vengono da una popolazione uniformemente distribuita, idealmente i valori di $DIFF(k)$, per $k \geq 2$, dovrebbero essere casualmente distribuiti attorno allo zero, sebbene in pratica tendano ad avere una piccola tendenza positiva. Supponendo che i dati siano raggruppati in k^* gruppi, quello che ci si aspetta è che $tr\mathbf{W}_k$ decresca notevolmente al crescere di k, finchè questa è inferiore al numero corretto di gruppi k^*, ma decresca più lentamente per $k > k^*$. Da queste considerazioni, segue che:

- per $k < k^*$, sia $DIFF(k)$ che $DIFF(k+1)$ dovrebbero assumere valori grandi e positivi;
- per $k > k^*$, sia $DIFF(k)$ che $DIFF(k+1)$ dovrebbero assumere valori piccoli. A volte, uno o entrambi potranno essere anche negativi.

$DIFF(k^*)$ dovrebbe quindi assumere valore grande e positivo, mentre $DIFF(k^*+1)$ valore relativamente piccolo (anche negativo).
Sulla base di quanto detto, il criterio, per ogni numero di cluster $k \geq 2$, è definito come

$$kl(k) = \left| \frac{DIFF(k)}{DIFF(k+1)} \right| \tag{2.3}$$

Il numero di cluster ottimo è il valore di k che massimizza $kl(k)$.

2.4 La statistica Gap

Il metodo descritto di seguito confronta la somma dei quadrati entro i cluster con quella attesa sotto una distribuzione di riferimento tramite la funzione:

$$Gap(k) = E^*\{log(tr\mathbf{W}_{kb})\} - log(tr\mathbf{W}_k) \tag{2.4}$$

dove E^* denota il valore atteso rispetto alla distribuzione di riferimento. Due sono le possibili scelte di tale distribuzione:

a) generare n valori per ognuna delle p variabili da una distribuzione uniforme, dove il supporto viene identificato per ciascuna variabile dal range dei valori della stessa (nelle simulazioni è identificata come GapUnif);

b) generare n valori per ognuna delle p variabili da una distribuzione uniforme con supporto allineato secondo le componenti principali. Nel dettaglio, se X è una matrice $n \times p$, si assume che le colonne abbiano media 0 e si calcola la decomposizione singolare $X = UDV^T$, dove U e V sono matrici ortogonali, e D matrice diagonale i cui elementi coincidono con gli autovalori di X. Si calcola $X' = XV$ e si genera la matrice Z' sulle colonne di X' come nel metodo a). Alla fine si applica la trasformazione inversa $Z = Z'V^T$ per avere il data set di riferimento (nelle simulazioni è identificata come GapPC).

Mentre il primo approccio ha il vantaggio della semplicità, nel secondo caso si prende in considerazione la forma della distribuzione dei dati. Comunque in entrambi gli approcci le variabili sono campionate in modo indipendente. Per il calcolo della statistica Gap si procede come segue:

1. Per $k = 1, ..., K$, (K è il valore massimo per k), si applica un algoritmo di clustering e si determina la $tr\mathbf{W}_k$.

2. Si generano B data set di riferimento (nelle simulazioni è stato usato $B = 30$) usando una delle distribuzioni descritte precedentemente. Per ogni valore di $k = 1, ..., K$, si determina la $tr\mathbf{W}^*_{kb}$, dove $b = 1, ..., B$. Si stima $E^*\{log(tr\mathbf{W}_{kb})\}$ con $\frac{1}{B}\sum_b log(tr\mathbf{W}^*_{kb})$, e si calcola la stima della statistica Gap (2.4).

3. Si calcola la deviazione standard sd_k del $log(tr\mathbf{W}^*_{kb})$, per $1 \leq b \leq B$, e si definisce $s_k = sd_k\sqrt{1+\frac{1}{B}}$.

Per $k \geq 1$, si stima il numero di cluster ottimo come il minimo k tale che

$$Gap(k) - Gap(k+1) + s_{k+1} \geq 0 \tag{2.5}$$

2.5 Statistica silhouette media

La statistica silhouette proposta da Kaufman e Rousseeuw (1990), può essere usata per selezionare il numero di cluster e per valutare quanto bene le osservazioni sono assegnate ai diversi gruppi.
Sia a_i la dissimilarità media tra l'osservazione i e tutte le osservazioni nel cluster, diciamo A, al quale i appartiene. Questo può essere fatto quando A contiene altre osservazioni a parte i, assumendo quindi che A non sia un insieme di una sola unità. Poi, per ogni altro cluster C diverso da A, si denoti con $d(i, C)$ la dissimilarità media di i da tutti gli oggetti di C e sia b_i la minima di queste $d(i, C)$.
La silhouette per ogni osservazione i è quindi definita come

$$s(i) = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{2.6}$$

Quando il cluster A contiene una sola osservazione, non è chiaro come $a(i)$ dovrebbe essere definito, e quindi per semplicità viene posto $s(i) = 0$. Gli autori ammettono che la scelta è certamente arbitraria, ma il valore zero sembra essere il più neutrale. Invece dalla (2.6), si vede facilmente che $-1 \leq s(i) \leq 1$ per ogni osservazione i.
La statistica silhouette media è la media di tutte le $s(i)$, ossia

$$sil_k = \frac{1}{n}\sum_i s(i) \tag{2.7}$$

Per $k \geq 2$, il numero di cluster ottimo è il valore di k che massimizza (2.7).

2.6 Stastica Split Silhouette Media

La statistica proposta al paragrafo 2.5 è in grado di identificare solo la struttura globale presente nei dati. Partendo da questa considerazione e dalla formula (2.6) è stata sviluppata la Split Silhouette Media (SSM).

Dato un risultato di clustering con k cluster, si divide ognuno di questi in due o più cluster. Il numero di questa divisione può essere determinato, ad esempio, massimizzando la funzione (2.7). Di conseguenza le osservazioni di ogni cluster diviso, hanno una nuova silhouette. La media di queste, detta "split silhouette", SS_i, $i = 1, ..., k$, è una misura di eterogeneità dei cluster.
Si definisce SSM come la media delle "split silhouette" sui k cluster, ossia

$$SSM(k) = \frac{1}{k} \sum_{i=1}^{k} SS_i \tag{2.8}$$

$SSM(1)$ non è altro che la statistica silhouette media per l'intero data set definita in 2.5.
Per $k \geq 1$, il numero di cluster ottimo è il valore di k che minimizza (2.8).

2.7 Metodo jump

Il criterio descritto di seguito si basa sul concetto di "distorsione", che è una misura di dispersione entro i cluster.
Formalmente, sia X un data set in $p-$dimensioni formato da un miscuglio di k^* componenti (il vero numero di cluster), ognuna con covarianza Γ; siano $X_1, ..., X_k$ i sottoinsiemi disgiunti di osservazioni di X che costituiscono i possibili cluster e $c_1, ..., c_k$, un insieme di candidati centri di questi. La minima distorsione raggiungibile, associata a k centri, è così definita:

$$d_k = \frac{1}{p} \left[\frac{1}{n} \sum_{i=1}^{k} (X_i - c_i)^T \Gamma^{-1} (X_i - c_i) \right] \tag{2.9}$$

che è semplicemente la distanza media di Mahalanobis per dimensione, tra X e l'insieme dei centri $c_1, ..., c_k$.
Si può notare che, nel caso in cui Γ sia la matrice identità, la distorsione non è altro che l'errore quadratico medio. Questo equivale, nella pratica, a stimare d_k con la minima distorsione che si ottiene applicando l'algoritmo $k-$means ai dati osservati.
Il metodo jump è quindi caratterizzato dai seguenti passi:

1. Applicare l'algoritmo $k-$means per differenti valori di k e calcolare la corrispondente distorsione d_k.

2. Calcolare i salti (jumps) $J_k = d_k^{-Y} - d_{k-1}^{-Y}$, dove il valore tipico della potenza è $Y = p/2$.

3. Si definisce $d_0^{-Y} = 0$, in modo tale da poter calcolare anche J_1.

Per $k \geq 1$, il numero di cluster ottimo è il valore di k che massimizza J_k.
La scelta di $Y = p/2$ è stata suggerita dagli autori, in quanto dovrebbe fornire dei buoni risultati sulla base della definizione (2.9). Tuttavia, se tra le variabili è presente una forte correlazione, gli autori indicano che valori di Y più piccoli di $p/2$ potrebbero produrre risultati migliori. Questo però è un aspetto ancora da indagare, ma che esula dagli scopi di questo lavoro. Di conseguenza nelle simulazioni sarà utilizzato il valore $Y = p/2$, come consigliato.

2.8 Statistica Gap pesata

Gap pesata è un miglioramento della statistica Gap descritta al paragrafo 2.4, ed è definita nel seguente modo:

$$\overline{Gap}(k) = E^*\{log(\overline{W}_{kb})\} - log(\overline{W}_k) \tag{2.10}$$

dove $\overline{W}_k = \sum_{m=1}^{k} \frac{1}{2n_m(n_m-1)} D_m$, in quanto ci sono $n_m(n_m - 1)$ coppie di osservazioni differenti in un cluster, se n_m è il numero di osservazioni nell'm-esimo cluster, per $m = 1, ..., k$. D_m rappresenta invece la somma delle distanze a coppie tra le osservazioni nell'm-esimo cluster, mentre E^* denota il valore atteso rispetto a una distribuzione di riferimento.

La differenza principale, rispetto alla statistica Gap, è che invece di considerare la somma delle distanze all'interno dei cluster, considera la media di tali distanze. Ecco perchè si parla di Gap pesata, ossia la statistica Gap pesata col numero di osservazioni nei cluster.
Gli autori ritengono che considerare $\overline{W}_k$ per quantificare l'omogeneità entro i cluster, piuttosto che la $tr\mathbf{W}_k$, sia presumibilmente più robusto, perchè, a condizione che vi siano abbastanza osservazioni per rappresentare l'$m-$esimo cluster, la stima $\frac{1}{2n_m(n_m-1)} D_m$ è più stabile rispetto a D_m, in quanto quest'ultima dipende più fortemente dal numero di osservazioni allocate al cluster.

A livello computazionale, la statistica Gap pesata è calcolata nello stesso modo della statistica Gap, eccetto per la differenza citata. Per quanto riguarda la distribuzione di riferimento sotto l'assunzione $k = 1$, si rimanda a quanto descritto nel paragrafo 2.4.
Di conseguenza, si procede nel seguente modo:

1. Per $k = 1, ..., K$, si applica un algoritmo di clustering e si determina la misura $\overline{W}_k$.

2. Si generano B data set di riferimento (nelle simulazioni è stato usato $B = 30$), usando una delle distribuzioni descritte nel paragrafo 2.4. Per ogni valore di $k = 1, ..., K$, si determina la misura $\overline{W}^*_{kb}$, dove $b = 1, ..., B$. Si stima $E^*\{log(\overline{W}_{kb})\}$ con $\frac{1}{B}\sum_b log(\overline{W}^*_{kb})$, e si calcola la stima della statistica Gap pesata (2.10).

3. Si calcola la deviazione standard sd_k del $log(\overline{W}^*_{kb})$, per $1 \leq b \leq B$, e si definisce $s_k = sd_k\sqrt{1 + \frac{1}{B}}$.

Per $k \geq 1$, si stima il numero di cluster ottimo, come il minimo k, tale che

$$\overline{Gap}(k) - \overline{Gap}(k+1) + s_{k+1} \geq 0 \tag{2.11}$$

2.8.1 Stastica DD-Gap pesata

Il criterio che segue si basa su differenze successive, come l'indice di Krzanowski e Lai descritto nel paragrafo 2.3, e confrota i valori $\overline{Gap}(k)$, $\overline{Gap}(k+1)$ e $\overline{Gap}(k-1)$.
Sia

$$D\overline{Gap}(k) = \overline{Gap}(k) - \overline{Gap}(k-1)$$

la differenza nel valore della funzione $\overline{Gap}(k)$, quando il numero di cluster passa da $k-1$ a k, con $k \geq 2$.
Se i dati sono ben raggruppati in k^* ($k^* \geq 2$) gruppi, ci si aspetta che $D\overline{Gap}(k) > 0$ per $k \leq k^*$ e vicino a zero quando $k > k^*$, mentre $D\overline{Gap}(k+1) > 0$ per $k < k^*$ e $D\overline{Gap}(k+1) \approx 0$ quando $k \geq k^*$.
Di conseguenza, si definisce la statistica DD-Gap pesata come

$$DD\overline{Gap}(k) = D\overline{Gap}(k) - D\overline{Gap}(k+1) \tag{2.12}$$

Per $k \geq 2$, il numero di cluster ottimo è il valore di k che massimizza la (2.12). La figura 2.1, illustra un esempio di impiego delle statistiche $D\overline{Gap}(k)$ e $DD\overline{Gap}(k)$.

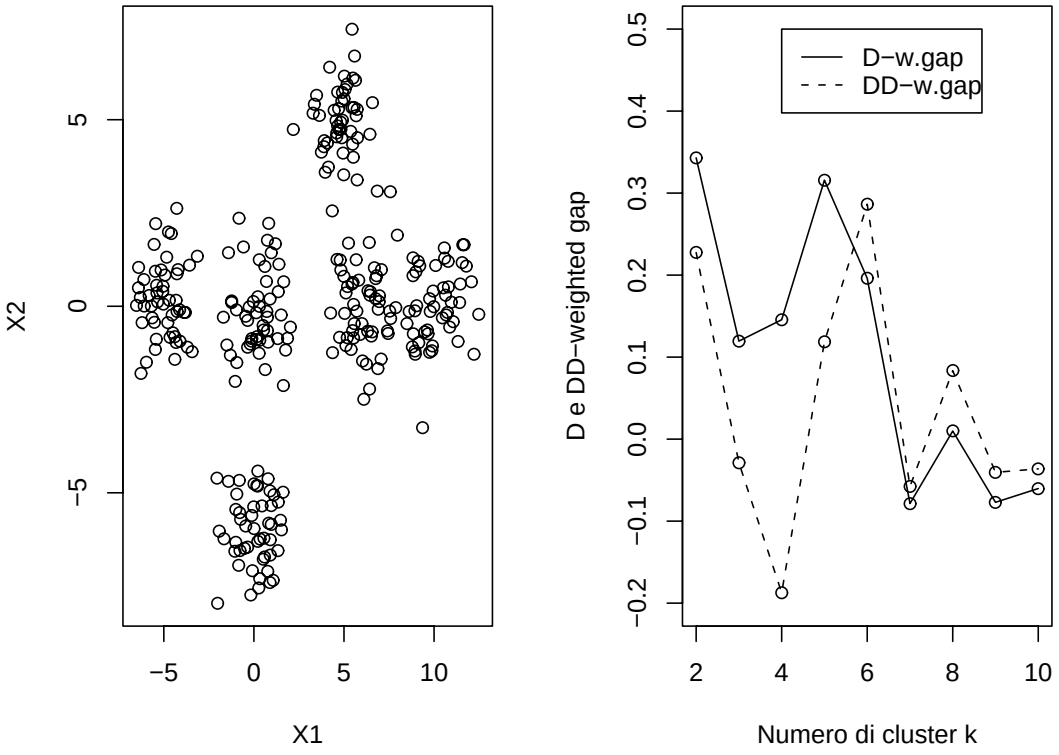

Figura 2.1: statistiche D e DD-weighted Gap. Simulati 6 cluster, da 50 osservazioni ciascuno, dalla distribuzione normale con vettori medi $(10, 0)'$, $(6, 0)'$, $(0, 0)'$, $(-5, 0)'$, $(5, 5)'$, $(0, -6)'$, e con matrice I_2 di varianza/covarianza.

2.9 Stabilità dei cluster

In questo paragrafo vengono presentati alcuni metodi che usano uno schema di ricampionamento, con l'obiettivo di testare la stabilità del data set considerato, cioè la capacità dello stesso di mantenere stabile la propria struttura.

2.9.1 La prediction strength

Il metodo descritto di seguito, valuta come alcuni gruppi possono essere predetti dai dati e quanto bene.
Attraverso una cross-validation a 2 campioni, si determinano per ogni data set, un solo campione di *training*, X_{tr}, e un solo campione di *test*, X_{te}, e si procede nel seguente modo:

1. Attraverso un algoritmo di clustering, si ottengono k cluster dal campione *training*. Questa operazione è denotata con $C(X_{tr}, k)$.

2. Attraverso un algoritmo di clustering, si ottengono k cluster dal campione *test*. Questa operazione è denotata con $C(X_{te}, k)$.

3. Si misura quanto bene i centri dei cluster del campione *training* prevedono le osservazioni del campione *test*. Questo significa, per ogni coppia di osservazioni *test* che sono assegnate allo stesso cluster *test*, valutare se vengono assegnate allo stesso cluster anche facendo riferimento ai centri *training*.

La prediction strength viene quindi definita da:

$$ps(k) = \min_{1 \leq j \leq k} \frac{1}{n_{kj}(n_{kj} - 1)} \sum_{i \neq i' \in A_{kj}} I(D[C(X_{tr}, k), X_{te}]_{ii'} = 1) \tag{2.13}$$

dove $A_{k1}, ..., A_{kk}$ indicano gli insiemi delle osservazioni nei cluster *test* $1, ..., k$, mentre $n_{k1}, ..., n_{kk}$ sono il numero di osservazioni degli stessi. La funzione indicatrice $I(D[C(X_{tr}, k), X_{te}]_{ii'} = 1)$, riportata in (2.13), vale 1 se gli elementi i e i' $(i \neq i')$ assegnati ad uno stesso cluster definito su X_{te}, appartengono allo stesso cluster anche attraverso il clustering $C(X_{tr}, k)$, e 0 altrimenti. Per ottenere dei risultati più robusti, la definizione di $ps(k)$ in (2.13), è stata modificata

nel seguente modo:

$$ps(k) = cv - ave \min_{1 \leq j \leq k} \frac{1}{n_{kj}(n_{kj}-1)} \sum_{i \neq i' \in A_{kj}} I(D[C(X_{tr}, k), X_{te}]_{ii'} = 1) \quad (2.14)$$

dove $cv - ave$ rappresenta la media su diverse divisioni casuali dei dati in X_{te} e X_{tr}.
Per $k \geq 1$, viene stimato il numero ottimo di cluster, come il più grande k tale che $ps(k) \geq 0.8$ o 0.9.

2.9.2 Statistica Clest

L'idea di questo metodo è di stimare il numero di cluster k^* dividendo ripetutamente il data set originale in due insiemi non sovrapposti, rispettivamente chiamati X_{tr} e X_{te}. Per ogni iterazione e per ogni numero di cluster k, viene applicato l'algoritmo di clustering sia a X_{tr} che a X_{te}. Usando l'insieme X_{tr} viene poi costruito un predittore, il quale sarà applicato all'insieme X_{te}. Le etichette predette dell'insieme X_{te}, saranno confrontate con quelle prodotte dalla procedura di clustering attraverso uno degli indici di similarità (esterni) descritti all'inizio del capitolo. Alla fine il numero di cluster è stimato confrontando la statistica di similarità osservata per ogni k con il suo valore atteso sotto un'adatta distribuzione nulla per $k = 1$.
La procedura Clest è così definita:

1. Si ripetano i seguenti passi B volte

    a) Dividere il data set originale in due insiemi non sovrapposti X_{tr} e X_{te}.

    b) Applicare una procedura di clustering ad entrambi gli insiemi ottenuti in a).

    c) Costruire un predittore, usando l'insieme X_{tr} e le sue etichette di clustering.

    d) Applicare il predittore ottenuto in c) all'insieme X_{te}.

    e) Calcolare un indice di similarità $s_{k,b}$, per confrontare i due insiemi di etichette per l'insieme X_{te}, vale a dire quelli ottenuti dalla procedura di clustering e dalla predizione descritta al punto d).

2. Si definisce t_k come la mediana degli indici ottenuti al punto 1.

3. Si generano B_0 data set sotto un'adatta ipotesi nulla. Per ciascuno dei data set si ripete la procedura descritta nei passi 1 e 2, ottenendo quindi B_0 statistiche di similarità, ossia $t_{k,1}, ..., t_{k,B_0}$

4. Si definisce t_k^0 come la media delle B_0 statistiche calcolate nel punto precedente. Inoltre sia p_k il p-value per t_k, calcolato come la proporzione di $t_{k,b} \geq t_k$. Si calcoli infine $D_k = t_k - t_k^0$, la differenza tra la statistica di similarità osservata e la sua stima sotto l'ipotesi nulla.

Per ogni numero di cluster k, $2 \leq k \leq K$, si eseguono i passi $1 - 4$, essendo K il massimo valore che può essere assunto da k.
Il numero ottimo di cluster, viene identificato nel k che massimizza la differenza D_k, nell'insieme dei valori di k per cui

$$\{2 \leq k \leq K : p_k \leq p_{max}, D_k \geq d_{min}\}$$

Se l'insieme è vuoto, la stima del numero di cluster è $k = 1$.
I parametri della statistica Clest utilizzati nelle simulazioni, sono contenuti nella tabella 2.5.

Parametri	Valore
Massimo numero di cluster	$K = 5$
Numero di training/test iterazioni	$B = 20$
Numero di dataset da generare	$B_0 = 20$
Dimensione dell'insieme training	$2n/3$
Predittore	DLDA[1]
Distribuzione nulla	Ipotesi uniforme[2]
Indice di similarità	Fowlkes e Mallows[3]
Massimo p-value e Minima differenza	$p_{max} = 0.05$ e $d_{min} = 0.05$

Tabella 2.5: parametri della statistica Clest

[1]DLDA: Linear discriminant analysis with diagonal covariance matrix. In R si trova nella libreria sma. La funzione è `stat.diag.da`

[2]É la stessa descritta per la statistica Gap

[3]cfr. tabella 2.2

2.10 Riepilogo

Nella tabella 2.6 sono riportati gli indici che sono stati discussi in questo capitolo. Si può notare l'utile separazione fra indici che sono in grado di individuare l'assenza di cluster in un dataset ($k = 1$) e indici che non sono in grado di farlo.

$k \geq 1$
Indice di Hartigan
Statistica Gap
Prediction strength
Statistica Clest
Split Silhouette Media
Metodo jump
Statistica Gap pesata
$k \geq 2$
Indice di Calinski e Harabasz
Indice di Krzanowski e Lai
Statistica Silhouette
Statistica DD-Gap pesata

Tabella 2.6: Indici

Capitolo 3

Nuovi metodi

I criteri che vengono definiti in questo capitolo, sono un estensione del metodo jump descritto al paragrafo 2.7. Tali criteri sono stati definiti con l'intento di migliorare il funzionamento del metodo stesso, in quanto in alcune situazioni, come nel caso di presenza di correlazione tra le variabili, non da sempre risultati soddisfacenti.

3.1 Metodo jump-differenza

L'idea del criterio che segue, si basa su differenze successive, come la statistica DD-Gap pesata e l'indice di Krzanowski e Lai descritti nel capitolo precedente. In tal modo vengono posti a confronto i valori d_{k-1}^{-Y}, d_k^{-Y} e d_{k+1}^{-Y}.
Partendo dalla procedura descritta per il metodo jump, il metodo jump-differenza viene determinato nel seguente modo:

1. Applicare l'algoritmo $k-$means per differenti valori di k e calcolare la corrispondente distorsione d_k.

2. Calcolare i salti (jumps) $J_k = d_k^{-Y} - d_{k-1}^{-Y}$, con $Y = p/2$.

3. Calcolare la differenza

$$DJ_k = J_k - J_{k+1} \tag{3.1}$$

4. Si definisce $d_0^{-Y} = 0$, in modo tale da poter calcolare anche DJ_1.

Per $k \geq 1$, il numero di cluster ottimo è il valore di k che massimizza DJ_k.
Le figure 3.1 e 3.2 illustrano, per due situazioni differenti, il comportamento del metodo jump-differenza definito nella formula 3.1, e quello del metodo jump.

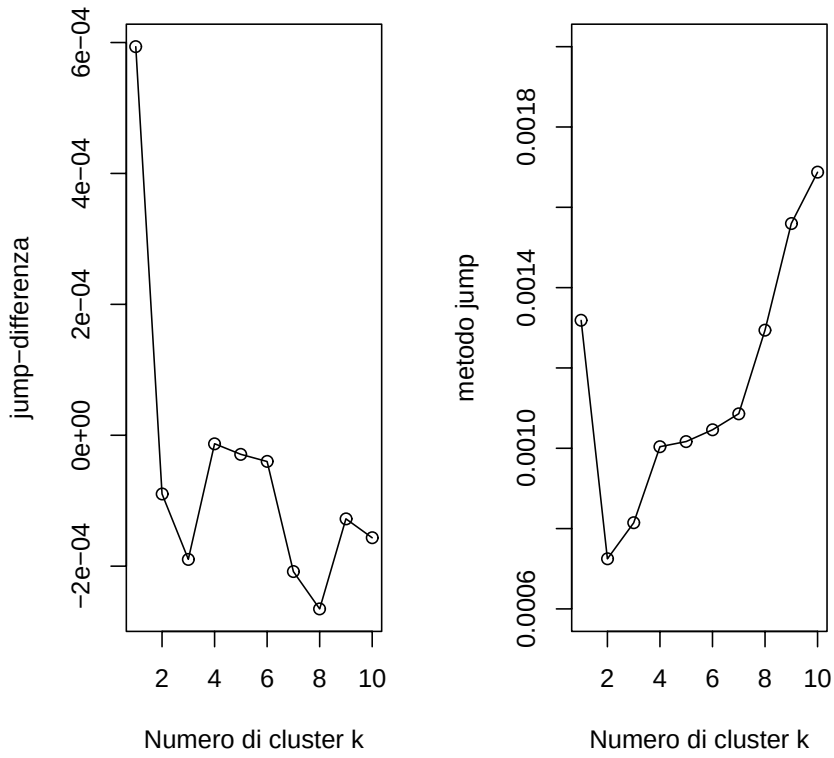

Figura 3.1: metodi jump-differenza e jump. Simulato 1 cluster in p=6 dimensioni, dalla distribuzione $N(0, 9)$ ($n = 200$)

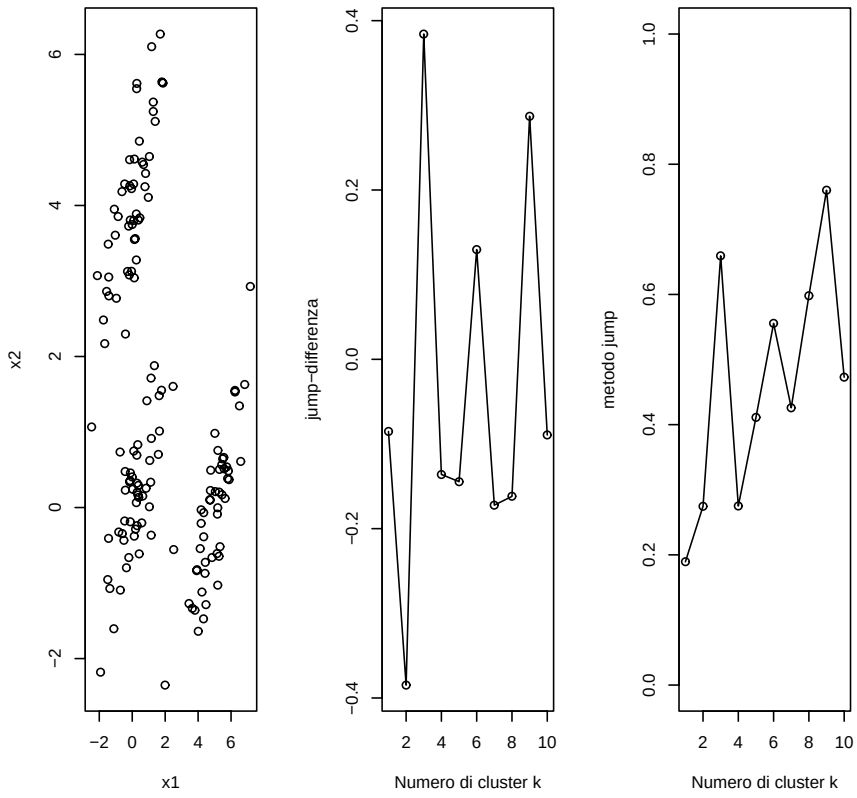

Figura 3.2: metodi jump-differenza e jump. Simulati 3 cluster da distribuzione normale con medie $(0,0)'$, $(5,0)'$ e $(0,4)'$ e matrice di covarianza Σ, dove la varianza $\sigma_{ii} = 1$ per $1 \leq i \leq 2$, e $\sigma_{ij} = 0.8$ per $1 \leq i \neq j \leq 2$ ($n = 150$)

Nei due esempi riportati, si può notare come il metodo jump-differenza, diversamente dal metodo jump, identifichi correttamente il numero di cluster k^*.

3.2 Metodo jump con Γ stimata

Sugar e James (2003), nell'implementare il metodo jump (cfr. paragrafo 2.7), assumono per convenienza che $\Gamma = I_p$. É proprio a partire da questa osservazione che si è deciso di considerare una stima della matrice Γ, anzichè utilizzare la matrice identità I_p. Questo cambiamento dovrebbe permettere di superare il problema legato alla correlazione tra le variabili.
La procedura per realizzare il metodo jump usando la stima $\widehat{\Gamma}$ diventa quindi:

Per $r = 1, ..., R$ (nelle simulazioni si è usato $R = 7$),

a) applicare l'algoritmo $r-$means, e ottenere la matrice dei centri cluster C_{ij}, dove $i = 1, ..., r$ e $j = 1, ..., p$.

b) Stimare la matrice di covarianza comune Γ_r, considerando i gruppi ottenuti in a), come
$$\widehat{\Gamma}_r = \frac{\sum_{m=1}^{r}(n_m - 1)Sx_m}{n - r}$$
dove Sx_m è la matrice di covarianza del gruppo $m-$esimo, n_m la numerosità del gruppo $m-$esimo e n il numero totale di osservazioni.

c) Calcolare la $tr\mathbf{B}_r$.

d) Applicare l'algoritmo $k-$means per differenti valori di k, e calcolare la corrispondente distorsione d_k, considerando la distanza di Mahalanobis. Per il calcolo di questa distanza, si utilizzano i centri dei cluster trovati al punto a), e la stima della matrice di covarianza $\widehat{\Gamma}_r$, determinata al punto b).

e) Si procede come i passi 2 e 3, del metodo jump.

Per $k \geq 1$, il numero di cluster ottimo è il valore di k che massimizza J_k. Naturalmente si ottengono R valori k, ognuno riferito ad una stima diversa di $\widehat{\Gamma}_r$.

Per scegliere quale di questi k considerare, è stato deciso di utilizzare il seguente semplice criterio:

$$k = k_r : \max_{r=1,\dots,R} \frac{tr\mathbf{B}_r}{k_r} \tag{3.2}$$

dove k_r è il valore di k stimato considerando la matrice di covarianza comune $\widehat{\Gamma}_r$. Tale rapporto può essere interpretato come la distanza media tra i cluster. Viene preso il valore massimo perchè, come già osservato precedentemente, ci si aspetta che la distanza tra i cluster sia grande, mentre quella entro i cluster sia piccola.
Nelle figure 3.3 e 3.4, si riprongono le stesse situazioni presentate per il metodo jump-differenza.

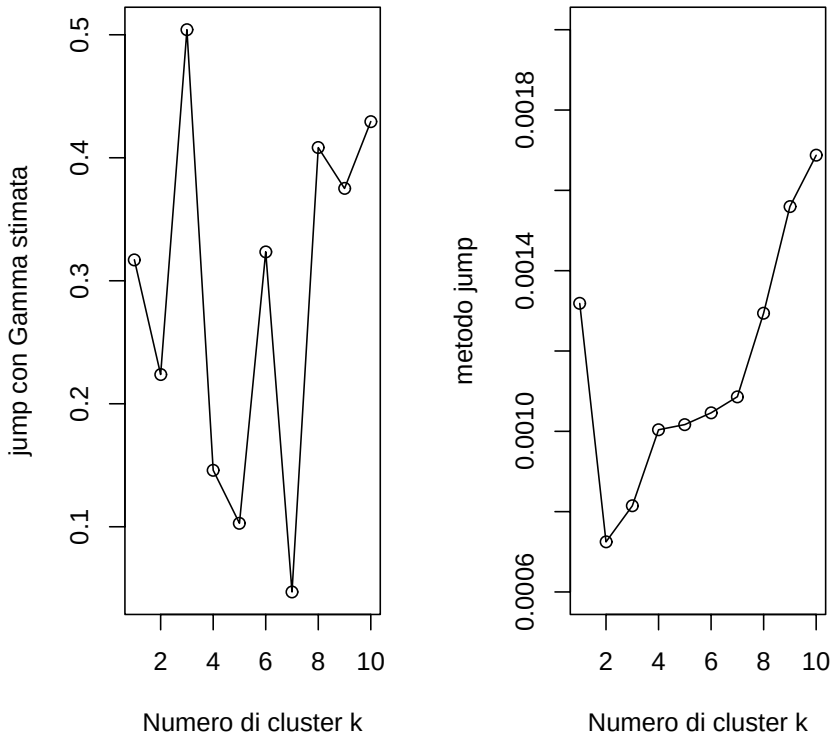

Figura 3.3: metodi jump con Γ stimata e jump. 1 cluster in $p = 6$ dimensioni

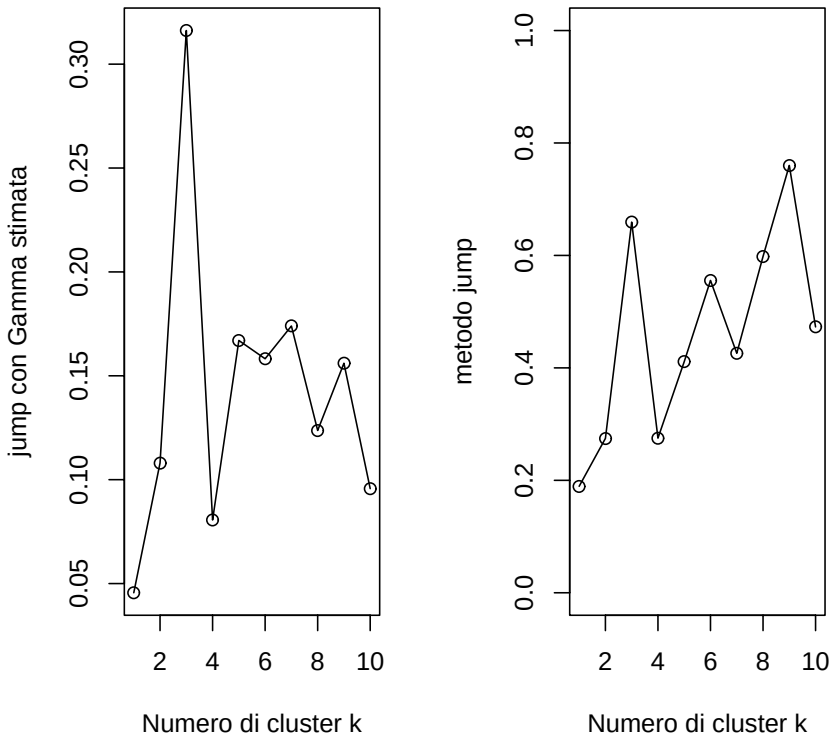

Figura 3.4: metodi jump con Γ stimata e jump. 3 cluster in 2 dimensioni

Nella prima situazione, il nuovo metodo introdotto identifica $k = 3$ piuttosto che $k = 1$. Tuttavia, anche se non rappresenta un buon risultato, sovrastima in misura inferiore il numero di cluster rispetto al metodo jump. Nella seconda situazione, determina invece correttamente il valore k^*.
I valori dei salti riportati nei grafici per il metodo jump con Γ stimata, si riferiscono al valore k scelto dal criterio (3.2).

3.3 Metodo jump-differenza con Γ stimata

Tale metodo è un estensione del criterio illustrato al paragrafo precedente, usando le differenze successive come nel caso descritto al paragrafo 3.1.
Ora, oltre a calcolare $J_k = d_k^{-Y} - d_{k-1}^{-Y}$, sarà calcolato anche $J_{k+1} = d_{k+1}^{-Y} - d_k^{-Y}$.
In questo modo è possibile ottenere

$$DJR_k = J_k - J_{k+1} \tag{3.3}$$

Per $k \geq 1$, il numero di cluster ottimo è il valore di k che massimizza DJR_k. Naturalmente, anche in questo caso, si ottengono R valori k, dato che Γ viene stimata R volte. Per determinare quale k scegliere si utilizza il criterio definito in (3.2).
Il comportamento di questo nuovo metodo si può osservare negli esempi riportati nelle figure 3.5 e 3.6.

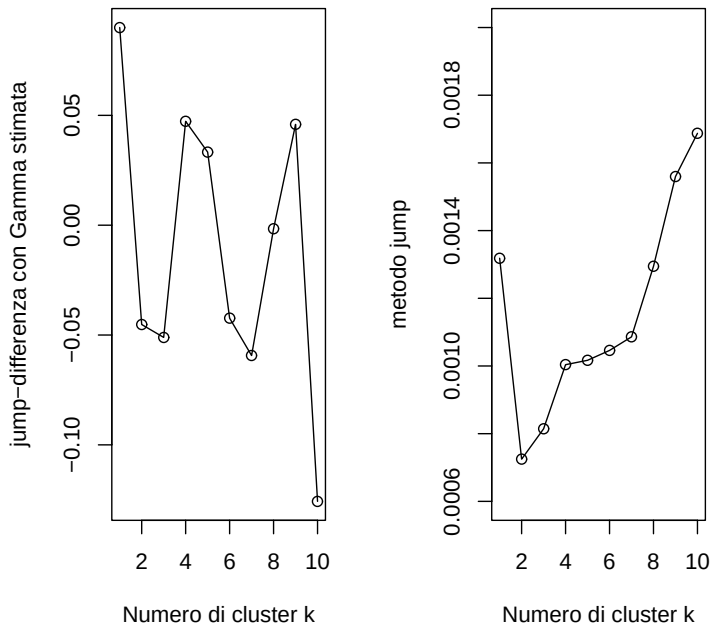

Figura 3.5: metodi jump-differenza con Γ stimata e jump. 1 cluster in $p = 6$ dimensioni

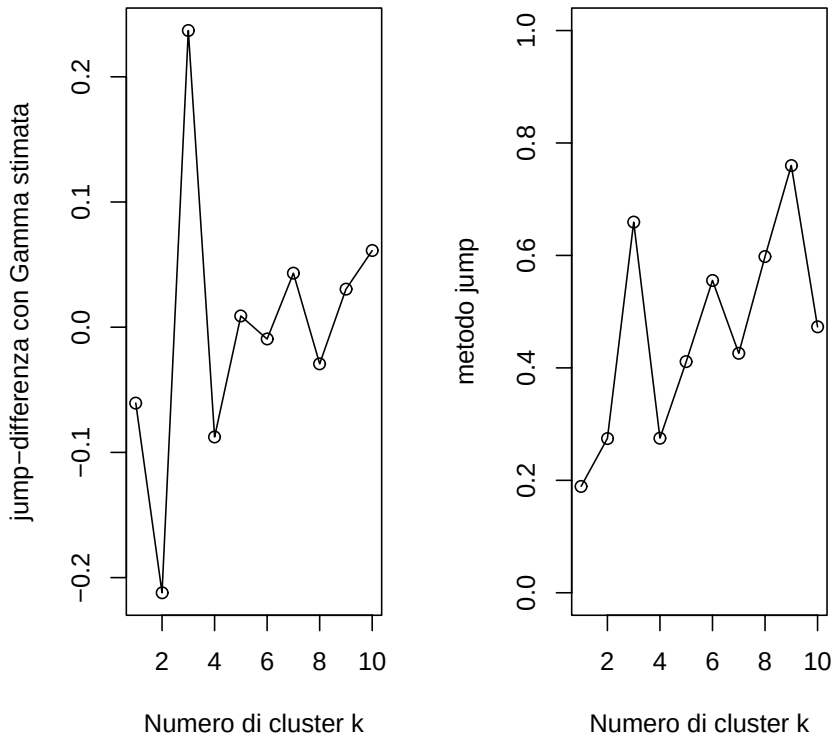

Figura 3.6: metodi jump-differenza con Γ stimata e jump. 3 cluster in 2 dimensioni

In entrambe le situazioni, il nuovo metodo identifica correttamente il numero di cluster k^*.
I valori dei salti rappresentati nelle due figure per il metodo jump-differenza con Γ stimata, si riferiscono al valore k scelto dal criterio (3.2).

Capitolo 4

Simulazioni

In questo capitolo vengono presentate le performance dei vari metodi descritti nei capitoli 2 e 3 per diversi scenari di simulazione e due data set reali.
Tutti i metodi e gli scenari di simulazione sono stati implementati con il linguaggio di programmazione *R* 2.7.1. Come algoritmo di partizionamento è stato utilizzato il $k-$means, in quanto è uno dei più conosciuti. Dato che tale algoritmo individua un ottimo locale, è stato ripetuto 20 volte, per ogni data set e per ogni k considerato, per poi scegliere la configurazione che minimizzava la somma dei quadrati entro i cluster. Sia per motivi computazionali, sia perchè nelle varie prove effettuate si sono ottenuti comunque risulati simili a quelli riportati dai diversi autori, per la statistica Clest e per la prediction strength l'algoritmo $k-$means è stato eseguito una sola volta.
Nel paragrafo 4.1 viene presentata una tabella per ogni scenario che contiene, in riferimento ai valori k indicati nella prima riga, le frequenze ottenute per ciascun metodo. Dopo aver individuato in 4.1 i metodi migliori, nel paragrafo 4.2 viene analizzato il loro comportamento, considerando un singolo data set per ogni scenario. É stato deciso di fare questo confronto diretto, perchè nella realtà si ha a disposizione un solo insieme di dati per individuare la presenza o meno di k gruppi, avvalendosi di uno dei metodi considerati. In aggiunta a questo, nel paragrafo 4.3 viene esaminato il comportamento dei metodi migliori su due data set reali noti in letteratura.
Per semplicità di esposizione nella presentazione dei risultati viene usata la notazione indicata in tabella 4.1.

CH	Indice di Calinski e Harabasz
Hartigan	Indice di Hartigan
KL	Indice di Krzanowski e Lai
GapUnif	Statistica Gap (distribuzione uniforme)
GapPC	Statistica Gap (componenti principali)
WGapUnif	Statistica Gap pesata (distribuzione uniforme)
WGapPC	Statistica Gap pesata (componenti principali)
DDGapUnif	Statistica DD-Gap pesata (distribuzione uniforme)
DDGapPC	Statistica DD-Gap pesata (componenti principali)
Sil	Silhouette
SSM	Split Silhouette Media
PS	Prediction strength
Clest	Statistica Clest
JM	Metodo jump
D-JM	Metodo jump-differenza
Cov JM	Metodo jump con Γ stimata
Cov D-JM	Metodo jump-differenza con Γ stimata

Tabella 4.1: notazione per i metodi

4.1 Scenari

Gli scenari sono stati scelti tra quelli già considerati in letteratura. Per ciascuno di questi, sono stati simulati 50 data set differenti.

Nella presentazione dei risultati ottenuti si sono evidenziati in grasseto il valore k^* di ogni scenario e, per ciascun metodo, la frequenza più alta.

4.1.1 1 cluster in 10 dimensioni

Le osservazioni sono uniformemente distribuite sull'iper-cubo di lato unitario in $p = 10$ dimensioni.

k	**1***	2	3	4	5	6	7	8	9	10	> 10
Hartigan	0	0	0	6	**34**	9	1	0	0	0	0
GapUnif	**42**	8	0	0	0	0	0	0	0	0	0
GapPC	**50**	0	0	0	0	0	0	0	0	0	0
WGapUnif	**49**	1	0	0	0	0	0	0	0	0	0
WGapPC	**50**	0	0	0	0	0	0	0	0	0	0
SSM	**49**	1	0	0	0	0	0	0	0	0	0
Clest	**48**	1	0	1	0	0	0	0	0	0	0
PS	**50**	0	0	0	0	0	0	0	0	0	0
JM	0	0	0	0	0	0	1	5	14	**30**	0
D-JM	**41**	0	0	0	0	0	2	3	4	0	0
Cov JM	0	2	**13**	10	6	5	4	4	5	1	0
Cov D-JM	4	3	16	**18**	6	3	0	0	0	0	0

Tabella 4.2: Primo scenario. 1 cluster in 10 dimensioni ($n = 200$)

Nella tabella 4.2 vengono posti a confronto solo i metodi definiti per $k \geq 1$. L'indice di Hartigan e i metodi JM e Cov JM, non individuano in nessuna situazione il valore k^*, mentre Cov D-JM lo individua solo nell'8% dei casi. Tutti gli altri metodi mostrano invece un buon comportamento.

Inoltre, dai risultati riportati si può notare che D-JM è il metodo più adatto, tra quelli nuovi, per indiviudare l'assenza di struttura nei dati.

4.1.2 3 cluster in 2 dimensioni (covarianze uguali)

Le osservazioni in ognuno dei 3 cluster sono normali bivariate indipendenti con vettori medi $(0,0)'$, $(0,5)'$, $(5,-3)'$ e con matrice di covarianza identità I_2. La numerosità per ogni cluster è rispettivamente 25, 25 e 50 osservazioni.

k	1	2	**3***	4	5	6	7	8	9	10	> 10
CH	-	0	**50**	0	0	0	0	0	0	0	0
KL	-	0	**31**	6	1	1	1	0	3	2	5
DDGapUnif	-	1	**46**	0	0	0	0	1	0	0	2
DDGapPC	-	0	**47**	0	0	0	0	1	0	0	2
Sil	-	0	**50**	0	0	0	0	0	0	0	0
Hartigan	0	0	0	0	0	0	1	6	9	15	**19**
GapUnif	0	0	**50**	0	0	0	0	0	0	0	0
GapPC	0	0	**50**	0	0	0	0	0	0	0	0
WGapUnif	0	0	**50**	0	0	0	0	0	0	0	0
WGapPC	0	0	**50**	0	0	0	0	0	0	0	0
SSM	0	0	**40**	3	4	1	0	0	0	2	0
Clest	0	0	**50**	0	0	0	0	0	0	0	0
PS	0	4	**46**	0	0	0	0	0	0	0	0
JM	0	0	**50**	0	0	0	0	0	0	0	0
D-JM	0	0	**50**	0	0	0	0	0	0	0	0
Cov JM	0	10	**40**	0	0	0	0	0	0	0	0
Cov D-JM	0	0	**50**	0	0	0	0	0	0	0	0

Tabella 4.3: Secondo scenario. 3 cluster in 2 dimensioni (cov. uguali; $n = 100$)

Nella tabella 4.3 si può notare che tutti i metodi tranne l'indice di Hartigan, che sovrastima il numero di cluster in ogni data set, individuano almeno nel 62% dei casi la partizione corretta. Ci sono ben 10 metodi che la individuano sempre (100% dei casi).

Questo tipo di risultato era prevedibile, dato che la simulazione riguardava tre cluster ben distinti.

4.1.3 3 cluster in 2 dimensioni (covarianze diverse)

Sono state generate 50 osservazioni da tre diverse distribuzioni normali bivariate con i seguenti vettori medi e matrici di covarianza:

$$\mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \mu_2 = \begin{pmatrix} 0 \\ 3 \end{pmatrix} \qquad \mu_3 = \begin{pmatrix} 4 \\ 3 \end{pmatrix}$$

$$\Sigma_1 = \begin{pmatrix} 4 & 1.7 \\ 1.7 & 1 \end{pmatrix} \quad \Sigma_2 = \begin{pmatrix} 0.25 & 0 \\ 0 & 0.25 \end{pmatrix} \quad \Sigma_3 = \begin{pmatrix} 4 & -1.7 \\ -1.7 & 1 \end{pmatrix}$$

Questo data set è stato proposto da Diday e Govaert (1977).

k	1	2	**3***	4	5	6	7	8	9	10	> 10
CH	-	0	0	0	2	2	6	8	3	8	**21**
KL	-	0	12	2	**13**	2	7	5	3	4	2
DDGapUnif	-	0	**20**	6	3	1	3	1	5	5	6
DDGapPC	-	0	**16**	4	5	3	3	2	4	6	7
Sil	-	0	10	10	10	**13**	7	0	0	0	0
Hartigan	0	0	0	0	0	0	0	0	0	1	**49**
GapUnif	**47**	0	3	0	0	0	0	0	0	0	0
GapPC	**49**	0	1	0	0	0	0	0	0	0	0
WGapUnif	**45**	0	2	3	0	0	0	0	0	0	0
WGapPC	**48**	0	2	0	0	0	0	0	0	0	0
SSM	0	0	0	1	1	2	10	8	5	8	**15**
Clest	0	0	**22**	10	18	0	0	0	0	0	0
PS	**22**	13	15	0	0	0	0	0	0	0	0
JM	0	0	0	0	**16**	1	11	7	4	11	0
D-JM	0	0	7	1	**19**	3	12	6	2	0	0
Cov JM	0	0	**17**	1	14	0	12	1	3	2	0
Cov D-JM	2	0	**31**	1	12	0	4	0	0	0	0

Tabella 4.4: Terzo scenario. 3 cluster in 2 dimensioni (cov. diverse; $n = 150$)

I risultati della tabella 4.4, riferiti ad uno scenario più complesso rispetto al precedente, dove i gruppi erano separati in maniera evidente, mostrano la difficoltà di tutti i metodi nell'individuare la partizione corretta k^*. Le statistiche GapUnif, GapPC, WGapUnif e WGapPC nella maggior parte dei casi, non riescono

a cogliere la struttura presente nei dati. Un comportamento simile, ma meno accentuato (44% dei casi), è tenuto anche da PS. I nuovi metodi D-JM, Cov JM e Cov D-JM funzionano meglio rispetto a JM da cui derivano, ma solo Cov D-JM riesce ad ottenere una performance del 62%, che è la migliore tra tutti i metodi. I metodi DDGapUnif, DDGapPC e la statistica Clest, individuano k^* almeno nel 32% dei casi. Per quanto riguarda invece gli altri metodi, si può notare una sovrastima del numero di cluster.

4.1.4 2 cluster allungati in 3 dimensioni

Il cluster 1 contiene 100 osservazioni da una distribuzione trivariata generata come segue: sia t il generico di 100 valori equispaziati nell'intervallo $[-0.5, 0.5]$ e si ponga inizialmente $x_1 = x_2 = x_3 = t$. Ad ogni componente è aggiunto un rumore gaussiano con deviazione standard 0.1. Il cluster 2 è generato allo stesso modo, considerando come intervallo di riferimento $[9.5, 10.5]$.

k	1	**2***	3	4	5	6	7	8	9	10	> 10
CH	-	0	0	6	1	**32**	0	11	0	0	0
KL	-	**50**	0	0	0	0	0	0	0	0	0
DDGapUnif	-	**50**	0	0	0	0	0	0	0	0	0
DDGapPC	-	**50**	0	0	0	0	0	0	0	0	0
Sil	-	**50**	0	0	0	0	0	0	0	0	0
Hartigan	0	0	0	0	0	0	0	0	0	2	**48**
GapUnif	0	0	5	11	2	**26**	3	3	0	0	0
GapPC	0	**50**	0	0	0	0	0	0	0	0	0
WGapUnif	0	0	0	0	12	**25**	7	6	0	0	0
WGapPC	0	**50**	0	0	0	0	0	0	0	0	0
SSM	0	0	0	0	0	0	0	6	9	15	**20**
Clest	0	**48**	2	0	0	0	0	0	0	0	0
PS	0	**50**	0	0	0	0	0	0	0	0	0
JM	0	0	0	1	0	**27**	0	18	1	3	0
D-JM	0	0	0	12	0	**22**	0	15	1	0	0
Cov JM	0	**50**	0	0	0	0	0	0	0	0	0
Cov D-JM	0	**50**	0	0	0	0	0	0	0	0	0

Tabella 4.5: Quarto scenario. 2 cluster in 3 dimensioni ($n = 200$)

Nella tabella 4.5 si può notare che i metodi CH, Hartigan, GapUnif, WGapUnif, SSM, JM e D-JM sovrastimano il numero di cluster, in quanto inducono a suddividere ognuno dei due cluster in più di 2 gruppi. Gli altri indici dimostrano invece un ottimo comportamento. Le statistiche Gap e Gap pesata, considerando l'orientamento secondo le componenti principali, ottengono risultati decisamente migliori.

4.1.5 3 cluster sovrapposti in 13 dimensioni

Ogni cluster contiene 50 osservazioni. Le prime tre variabili hanno una distribuzione normale multivariata con vettori medi $(0,0,0)'$, $(2,-2,2)'$, $(-2,2,-2)'$ e matrice di covarianza Σ, dove $\sigma_{ii} = 1$ per $1 \leq i \leq 3$, e $\sigma_{ij} = 0.5$ per $1 \leq i \neq j \leq 3$. Le rimanenti 10 variabili sono simulate indipendentemente dalla distribuzione $N(0_{10}, I_{10})$.

k	1	2	**3***	4	5	6	7	8	9	10	> 10
CH	-	**50**	0	0	0	0	0	0	0	0	0
KL	-	7	**30**	1	0	3	1	2	2	0	4
DDGapUnif	-	**50**	0	0	0	0	0	0	0	0	0
DDGapPC	-	**46**	1	0	0	0	0	1	0	1	1
Sil	-	**50**	0	0	0	0	0	0	0	0	0
Hartigan	0	0	**49**	1	0	0	0	0	0	0	0
GapUnif	0	4	**46**	0	0	0	0	0	0	0	0
GapPC	0	2	**48**	0	0	0	0	0	0	0	0
WGapUnif	0	23	**26**	1	0	0	0	0	0	0	0
WGapPC	0	10	**37**	3	0	0	0	0	0	0	0
SSM	0	0	17	**19**	8	6	0	0	0	0	0
Clest	0	11	**38**	1	0	0	0	0	0	0	0
PS	**35**	2	13	0	0	0	0	0	0	0	0
JM	0	0	0	0	0	0	0	2	20	**28**	0
D-JM	0	0	**27**	0	1	6	3	5	8	0	0
Cov JM	0	8	**36**	0	0	3	0	1	2	0	0
Cov D-JM	0	22	**28**	0	0	0	0	0	0	0	0

Tabella 4.6: Quinto scenario. 3 cluster in 13 dimensioni ($n = 150$)

I risultati esposti nella tabella 4.6 mostrano, ancora una volta, il netto miglioramento di performance dei nuovi metodi D-JM, Cov JM e Cov D-JM, rispetto a JM che sovrastima il numero di cluster. I metodi CH, DDGapUnif, DDGapPC e Sil individuano in 2 cluster la partizione migliore piuttosto che 3. Solo PS, nel 70% dei casi non individua nessuna struttura presente nei dati. La statistica Gap mostra un comportamento migliore rispetto alla Gap pesata, per entrambi i tipi di scelta. La SSM si orienta, nella maggior parte dei casi, su una suddi-

visione in 4 gruppi. I metodi che funzionano meglio, per questo scenario, sono Hartigan, GapPC e GapUnif.

4.1.6 2 cluster in 2 dimensioni

Entrambi i cluster sono stati generati da una distribuzione normale bivariata. Il cluster 1 ha 100 osservazioni con vettore medio $(0, 0)'$ e matrice di covarianza identità I_2. Il cluster 2 contiene 15 osservazioni con vettore medio $(5, 0)'$ e matrice di covarianza $0.1I_2$.

k	1	**2***	3	4	5	6	7	8	9	10	> 10
CH	-	7	0	1	0	1	3	2	9	7	**20**
KL	-	**26**	2	4	5	3	1	1	3	1	4
DDGapUnif	-	**44**	2	0	0	0	0	0	0	1	3
DDGapPC	-	**45**	1	0	0	0	0	0	0	1	3
Sil	-	**50**	0	0	0	0	0	0	0	0	0
Hartigan	0	0	0	0	0	0	0	0	0	11	**39**
GapUnif	2	12	**21**	15	0	0	0	0	0	0	0
GapPC	2	11	**22**	15	0	0	0	0	0	0	0
WGapUnif	0	**45**	5	0	0	0	0	0	0	0	0
WGapPC	0	**44**	6	0	0	0	0	0	0	0	0
SSM	0	**20**	5	9	3	5	1	1	2	0	4
Clest	**28**	16	4	1	1	0	0	0	0	0	0
PS	4	**46**	0	0	0	0	0	0	0	0	0
JM	0	0	0	3	2	5	11	1	**15**	13	0
D-JM	0	4	1	11	5	8	8	1	**12**	0	0
Cov JM	0	0	6	8	9	9	7	3	7	1	0
Cov D-JM	0	19	**20**	8	2	1	0	0	0	0	0

Tabella 4.7: Sesto scenario. 2 cluster in 2 dimensioni ($n = 115$)

La particolarità di questo scenario, rispetto agli altri, è la netta distinzione del numero di osservazioni tra i due cluster. Nella tabella 4.7, si può osservare che Sil, DDGapUnif, DDGapPC, WGapUnif, WgapPC e PS, sono i metodi che almeno nell'88% dei casi, individuano la partizione corretta k^*. La statistica Clest, solo nel 32% dei casi, riesce ad individuare la distinzione tra i due cluster,

mentre il metodo Cov D-JM nel 38% dei casi. Nonostante quest'ultimo dato non sia così soddisfacente, è il migliore tra i nuovi metodi. I metodi CH, Hartigan e JM tendono invece a sovrastimare il numero di cluster. Per quanto riguarda le statistiche GapUnif e GapPC, entrambe si orientano ad individuare, la maggior parte delle volte, in 3 la partizione corretta piuttosto che 2.

4.1.7 5 cluster in 2 dimensioni

Le osservazioni in ognuno dei 5 cluster sono normali bivariate indipendenti con vettori medi $(0,0)'$, $(2.5,2.5)'$, $(5,5)'$, $(-2.5,2.5)'$, $(-5,-5)'$ e con matrice di covarianza identità I_2. La numerosità è la stessa per ogni cluster.

k	1	2	3	4	**5***	6	7	8	9	10	> 10
CH	-	0	0	0	**49**	1	0	0	0	0	0
KL	-	0	8	0	**19**	0	7	5	4	3	4
DDGapUnif	-	**33**	15	0	0	0	0	0	0	1	1
DDGapPC	-	0	**28**	1	15	0	0	0	0	3	3
Sil	-	15	**17**	2	16	0	0	0	0	0	0
Hartigan	0	0	0	0	0	1	3	5	11	11	**13**
GapUnif	0	0	**35**	0	15	0	0	0	0	0	0
GapPC	**49**	0	1	0	0	0	0	0	0	0	0
WGapUnif	0	3	**45**	0	2	0	0	0	0	0	0
WGapPC	15	0	12	6	**17**	0	0	0	0	0	0
SSM	0	1	0	4	**25**	12	3	1	2	0	2
Clest	0	0	**29**	1	20	0	0	0	0	0	0
PS	**23**	0	19	1	7	0	0	0	0	0	0
JM	0	0	1	2	**47**	0	0	0	0	0	0
D-JM	0	0	1	0	**49**	0	0	0	0	0	0
Cov JM	0	0	2	**32**	16	0	0	0	0	0	0
Cov D-JM	0	0	1	11	**38**	0	0	0	0	0	0

Tabella 4.8: Settimo scenario. 5 cluster in 2 dimensioni ($n = 100$)

I risultati della tabella 4.8 mostrano che solo i metodi CH, JM e D-JM funzionano in maniera soddisfacente. Vi sono ben 10 metodi che sottostimano, la maggior parte delle volte, il numero di cluster. Tra questi c'è da segnalare

che GapPC non individua praticamente mai la presenza di struttura nei dati. In misura minore, anche PS e WGapPC hanno un comportamento simile, però a differenza di GapPC, riescono ad individuare la partizione corretta k^*, rispettivamente nel 14% e nel 34% dei casi.

4.1.8 Riepilogo

Dalla tabella 4.9, che riassume le performance dei metodi per ogni scenario, in riferimento alla partizione corretta k^*, emerge che i risultati migliori, in media, sono stati ottenuti dalle statistiche WGapPC, Clest, PS e Cov D-JM.

	Scenari							
	Primo	Secondo	Terzo	Quarto	Quinto	Sesto	Settimo	Media
CH	-	100	0	0	0	14	98	35.33
KL	-	62	24	100	60	52	38	56
DDGapUnif	-	92	40	100	0	88	0	53.33
DDGapPC	-	94	32	100	2	90	30	58
Sil	-	100	20	100	0	100	32	58.67
Hartigan	0	0	0	0	98	0	0	14
GapUnif	84	100	6	0	92	24	30	48
GapPC	100	100	2	100	96	22	0	60
WGapUnif	98	100	4	0	52	90	4	49.71
WGapPC	100	100	4	100	74	88	34	71.43
SSM	98	80	0	0	34	40	50	43.14
Clest	96	100	44	96	76	32	40	69.14
PS	100	92	30	100	26	92	14	64.86
JM	0	100	0	0	0	0	94	27.71
D-JM	82	100	14	0	54	8	98	50.86
Cov JM	0	80	34	100	72	0	32	45.43
Cov D-JM	8	100	62	100	56	38	76	62.86

Tabella 4.9: percentuali di individuazione del numero corretto di cluster. La media per i metodi CH, KL, DDGapUnif, DDGapPC e Sil è stata fatta considerando gli scenari dal secondo al settimo.

4.2 Prove singole

Nella realtà, un utente dispone di un solo insieme di dati per verificare, tramite la cluster analisi, la presenza di uno o più gruppi. Partendo da questo fatto, è quindi importante valutare come i migliori metodi, individuati dall'analisi sin qui condotta, si comportano in tale situazione.

		Metodi			
Scenari	$\mathbf{k}^*$	Clest	WGapPC	PS	Cov D-JM
Primo	**1**	1	1	1	2
Secondo	**3**	3	3	3	3
Terzo	**3**	3	1	3	3
Quarto	**2**	2	2	2	2
Quinto	**3**	3	3	1	3
Sesto	**2**	1	2	2	3
Settimo	**5**	3	4	3	5

Tabella 4.10: Performance dei metodi migliori

I valori di k ottenuti, simulando un data set da ogni scenario descritto nel paragrafo 4.1, sono contenuti nella tabella 4.10. Si può osservare che ciascun metodo non individua il numero corretto di cluster in due casi che variano a seconda della metodologia utilizzata. Si nota, inoltre, che essi non individuano correttamente k^* proprio negli scenari dove manifestavano la performance più bassa (cfr. tabella 4.9). Nell'ultimo, solo il nuovo metodo Cov D-JM riesce ad individuare correttamente k^*.

4.3 Data set reali

In questo paragrafo, vengono considerati due data set reali. Nella tabella 4.11, vengono riportate le stime del "vero" numero di cluster k^*, per ciascuno dei metodi migliori.

		Metodi			
Dataset	$\mathbf{k}^*$	Clest	WGapPC	PS	Cov D-JM
Iris	**2/3**	3	4	2	3
Breast cancer	**2**	4	2	3	2

Tabella 4.11: Performance dei metodi migliori

Il data set Iris, studiato da Fisher (1936), contiene 150 osservazioni misurate su 4 variabili. Tali osservazioni sono classificate in 3 gruppi: setosa, versicolor e virginica. Tuttavia, è noto, che mentre il gruppo setosa è ben distinto dagli altri, versicolor e virginica sono alquanto sovrapposti. É quindi ragionevole attendersi, che il data set venga considerato come contenente 3 cluster, o almeno 2, dai metodi migliori.
Il data set Wisconsin breast cancer (Wolberg e Mangasarian, 1990) contiene, invece, le misurazioni su 9 variabili registrate per ognuno dei 683 pazienti affetti da cancro. Di questi, 444 erano benigni, mentre i rimanenti 239 erano maligni. Si identificano quindi, 2 cluster ben distinti.
Dai risultati ottenuti emerge che solo il nuovo metodo Cov D-JM è in grado, in entrambe le situazioni, di individuare il k^* corretto. La statistica Clest stima correttamente k^* in Iris, ma non nell'altra situazione. WGapPc si comporta, invece, nella maniera inversa. Per quanto riguarda il metodo PS, è l'unico che non riesce a distinguere versicolor da virginica, ed individua in 2 il numero di cluster.

Conclusioni

In questo lavoro, sono stati presentati tre nuovi metodi per individuare il numero di cluster in un dataset:

- jump-differenza (D-JM);
- jump con Γ stimata (Cov JM);
- jump-differenza con Γ stimata (Cov D-JM).

Rispetto al metodo jump da cui derivano, i metodi proposti hanno mostrato una performance migliore. La tabella che segue, ripropone i risultati ottenuti, limitatamente al confronto tra questi, e il metodo jump.

	Scenari						
	Primo	Secondo	Terzo	Quarto	Quinto	Sesto	Settimo
JM	0	100	0	0	0	0	94
D-JM	82	100	14	0	54	8	98
Cov JM	0	80	34	100	72	0	32
Cov D-JM	8	100	62	100	56	38	76

Tabella 4.12: percentuali di individuazione del numero corretto di cluster

Si può notare che in 5 dei 7 scenari proposti, il metodo JM non riesce mai a individuare il valore corretto k^*. Nel primo scenario, dove era stato simulato un unico cluster in 10 dimensioni, solo il metodo D-JM riesce ad ottenere dei buoni risultati. Per quanto riguarda gli altri scenari, è quasi sempre il metodo Cov D-JM ad ottenere i risultati migliori. JM è in grado di individuare in modo soddisfacente la partizione corretta, solo nel secondo e nel settimo scenario, in quanto la struttura dei dati simulati era abbastanza semplice.

Si può quindi pensare di utilizzare Cov D-JM, dopo aver verificato preliminarmente, ad esempio con D-JM, che $k \neq 1$. La difficoltà per $k = 1$ deriva dalla definizione del criterio $\max_{r=1,\ldots,R} \frac{tr\mathbf{B}_r}{k_r}$, per scegliere il valore k.

$tr\mathbf{B}_r$	k_r	$tr\mathbf{B}_r/k_r$
0	1	0
17.226	**2**	8.613
28.327	7	4.047
35.735	7	5.105
42.270	5	8.454
47.976	6	7.996
51.975	7	7.425

Tabella 4.13: scelta di k (primo scenario)

Come si vede in tabella 4.13, il valore massimo del rapporto si ha per $k = 2$. Ciò non sorprende in quanto il criterio è stato definito con l'intento di massimizzare la distanza media tra cluster, il che implica che k sia almeno pari a 2.

Nella tabella 4.9 che riassume le performance di tutti i metodi considerati, si è visto che i risultati migliori, in media, sono stati ottenuti dalle statistiche WGapPC, Clest, PS e dal nuovo metodo Cov D-JM.

Nel primo scenario Clest, WGapPC e PS hanno ottenuto dei risultati molto soddisfacenti rispetto a Cov D-JM. Tuttavia andando a valutare il comportamento di questi metodi negli altri scenari (dal secondo in poi), si può osservare che Cov D-JM ottiene una performance media un pò superiore, come si può vedere dalla tabella 4.14.

Cov D-JM	72
WGapPC	66.67
Clest	64.67
PS	59

Tabella 4.14: percentuali medie di individuazione del numero corretto di cluster (nella media non viene considerato il primo scenario)

Queste performance sono riferite alla simulazione di 50 data set per ogni scenario. Per valutare la bontà di tali risultati, si è provato a raddoppiare il numero di data set, e si è visto che le performance ottenute non sono significativamente diverse da quelle riportate in tabella 4.14.
Inoltre, analizzando attentamente il comportamento complessivo dei metodi migliori, c'è da segnalare che WGapPC e PS in alcuni scenari hanno individuato in un certo numero di casi $k = 1$, quando il vero valore era $k^* > 1$. Ciò è accaduto, in misura minore, anche per la statistica Clest, nel sesto scenario e per Cov D-JM, nel terzo (in misura trascurabile). Si fa notare questo perchè, dato un insieme di dati, il primo passo da compiere in una cluster analisi è capire se possono essere presenti o meno più gruppi. L'applicazione di uno di questi metodi potrebbe quindi portare a selezionare $k = 1$, anche quando è effettivamente presente una certa struttura. Di conseguenza, si propongono le seguenti due procedure:

1. - **Procedura A**: applicare la statistica Split Silhouette Media (SSM) e poi procedere con i passi 2, 3 e 4.
   - **Procedura B**: applicare il metodo D-JM e poi procedere con i passi 2, 3 e 4.

2. Se si ottiene $k = 1$, fermarsi e concludere che non vi è la presenza di gruppi nei dati.

3. Se $k \neq 1$, applicare il metodo Cov D-JM.

4. Il valore ottenuto è la stima di k^*.

La scelta dei metodi proposti per la fase 1, per valutare l'assenza di struttura nei dati, potrebbe sembrare, per quanto detto finora, non appropriata, in quanto non si tratta dei metodi migliori. In realtà, valutando nel complesso le performance di entrambi i metodi, emerge che negli scenari dove era presente una certa struttura, cioè $k^* > 1$, questi non hanno mai individuato $k = 1$. Questo è un aspetto molto importante, perchè il primo passo da compiere in una cluster analisi, è capire se c'è o meno una struttura nei dati.
Per valutare il comportamento delle procedure proposte sono stati simulati 50 data set da ogni scenario. I risultati ottenuti sono riportati nella tabella 4.15.

		k										
Scenari	$\mathbf{k}^*$	1	2	3	4	5	6	7	8	9	10	> 10
Primo A (SSM)	**1**	**100**	0	0	0	0	0	0	0	0	0	0
Primo B (D-JM)	**1**	**86**	2	4	0	6	2	0	0	0	0	0
Secondo	**3**	0	0	**100**	0	0	0	0	0	0	0	0
Terzo	**3**	2	0	**72**	4	16	2	4	0	0	0	0
Quarto	**2**	0	**100**	0	0	0	0	0	0	0	0	0
Quinto	**3**	0	40	**60**	0	0	0	0	0	0	0	0
Sesto	**2**	0	**42**	42	14	0	2	0	0	0	0	0
Settimo	**5**	0	0	0	24	**76**	0	0	0	0	0	0

Tabella 4.15: percentuali di individuazione di k cluster

La performance media per la procedura A è del 78.57% mentre per la procedura B è del 76.57%. In entrambi i casi quindi si consegue un miglioramento della performance media rispetto ai due metodi sin qui risultati migliori WGapPC (71.43%) e Clest (69.14%).
A conclusione di queste brevi considerazioni sulle due procedure proposte, pare opportuno richiamare l'attenzione su due aspetti:

- la procedura A risulta preferibile sul piano della performance (+2%);
- la procedura B garantisce una struttura più semplice e meno onerosa dal punto di vista computazionale.

Sarà quindi l'utilizzatore a scegliere di volta in volta quale tra le due procedure risponde meglio alle sue esigenze.

Appendice A

Codice R delle funzioni utilizzate

A.1 Calinski e Harabasz

```
ch <- function(x)
{
   #x: dati in input
   k <- 11
   valCH <- NULL #vettore per contenere i valori dell'indice
   for(i in 2:k)
   {
      cl <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      wi <- sum(cl$withinss)

      for(z in 1:19)
      {
         ab <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
         if(wi>sum(ab$withinss))
         {
            cl <- ab
            wi <- sum(ab$withinss)
         }
      }
      #La funzione index.G1 si trova nella libreria ‘‘clusterSim’’
      val <- index.G1(x,cl$cluster) #calcolo indice di Calinski e Harabasz
      valCH <- c(valCH,val)
   }

   max <- valCH[1]
   pos <- 2
   for(j in 2:length(valCH))
   {
      if(valCH[j]>max)
      {
         max <- valCH[j]
         pos <- j+1
```

```
      }
   }
   pos #valore k scelto
}
```

A.2 Indice di Hartigan

```
h <- function(x)
{
   #x: dati in input
   k <- 11
   valH <- NULL #vettore per contenere i valori dell'indice
   for(i in 1:k)
   {
      if(i==1)
         cl1 <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      else
         cl1 <- cl2
      cl2 <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
      wi1 <- sum(cl1$withinss)
      wi2 <- sum(cl2$withinss)

      for(z in 1:19)
      {
         if(i==1)
         {
            cl1b <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
            if(wi1>sum(cl1b$withinss))
            {
               cl1 <- cl1b
               wi1 <- sum(cl1b$withinss)
            }
         }
         cl2b <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
         if(wi2>sum(cl2b$withinss))
         {
            cl2 <- cl2b
            wi2 <- sum(cl2b$withinss)
         }
      }
      clall <- cbind(cl1$cluster,cl2$cluster)
      #La funzione index.H si trova nella libreria ‘‘clusterSim’’
      val <- index.H(x,clall) #calcolo indice di Hartigan
      valH <- c(valH,val)
   }

   if(min(valH)<10)
   {
      for(z in 1:length(valH))
      {
```

```
         if(valH[z]<=10)
         {
            pos <- z
            break
         }
      }
   }
   else
      pos <- 12 #assegnato valore 12 per convenienza!

   pos #valore k scelto
}
```

A.3 Indice di Krzanowski e Lai

```
kl <- function(x)
{
   #x: dati in input
   k <- 11
   valKL <- NULL #vettore per contenere i valori dell'indice
   for(i in 2:k)
   {
      if(i==2)
      {
         cl1 <- rep(1,nrow(x))
         cl2 <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      }
      else
      {
         cl1 <- cl2
         cl2 <- cl3
      }
      cl3 <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
      wi2 <- sum(cl2$withinss)
      wi3 <- sum(cl3$withinss)

      for(z in 1:19)
      {
         if(i==2)
         {
            cl2b <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
            if(wi2>sum(cl2b$withinss))
            {
               cl2 <- cl2b
               wi2 <- sum(cl2b$withinss)
            }
         }
         cl3b <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
         if(wi3>sum(cl3b$withinss))
         {
```

```
            cl3 <- cl3b
            wi3 <- sum(cl3b$withinss)
         }
      }
      if(i==2)
         clall <- cbind(cl1,cl2$cluster,cl3$cluster)
      else
         clall <- cbind(cl1$cluster,cl2$cluster,cl3$cluster)
      #La funzione index.KL si trova nella libreria ‘‘clusterSim’’
      val <- index.KL(x,clall) #calcolo indice di Krzanowski e Lai
      valKL <- c(valKL,val)
   }
   max <- valKL[1]
   pos <- 2
   for(z in 2:length(valKL))
   {
      if(valKL[z]>max)
      {
         max <- valKL[z]
         pos <- z+1
      }
   }
   pos #valore k scelto
}
```

A.4 Statistica Gap

La funzione `gap.unif.pc` riportata di seguito, restituisce la stima del valore k per le statistiche GapUnif e GapPC.

```
gap.unif.pc <- function(x)
{
   #x: dati in input
   k <- 11
   indGapUnif <- NULL #vettore per contenere i valori della statistica GapUnif
   indGapPc <- NULL #vettore per contenere i valori della statistica GapPC

   for(i in 1:k)
   {
      if(i==1)
         cl1 <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      else
         cl1 <- cl2
      cl2 <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
      wi1 <- sum(cl1$withinss)
      wi2 <- sum(cl2$withinss)

      for(z in 1:19)
      {
```

```
        if(i==1)
        {
           cl1b <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
           if(wi1>sum(cl1b$withinss))
           {
              cl1 <- cl1b
              wi1 <- sum(cl1b$withinss)
           }
        }

        cl2b <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
        if(wi2>sum(cl2b$withinss))
        {
           cl2 <- cl2b
           wi2 <- sum(cl2b$withinss)
        }
     }
     clall <- cbind(cl1$cluster,cl2$cluster)
     #La funzione index.Gap si trova nella libreria ‘‘clusterSim’’
     #Dal prompt di R: fix(index.Gap)
     #               - Per l’algoritmo kmeans, usare quello di ‘‘MacQueen’’
     #               - Far ritornare solo il valore ‘‘diffu’’
     #calcolo statistica GapUnif
     valUnif <- index.Gap(x,clall,reference.distribution="unif",B=30,method="k-means")
     #calcolo statistica GapPC
     valPc <- index.Gap(x,clall,reference.distribution="pc",B=30,method="k-means")
     indGapUnif <- c(indGapUnif,valUnif)
     indGapPc <- c(indGapPc,valPc)
  }

  # Scelta del valore k
  for(z in 1:length(indGapUnif))
  {
     if(indGapUnif[z]>0)
     {
        kGapUnif <- z
        break
     }
  }
  for(r in 1:length(indGapPc))
  {
     if(indGapPc[r]>0)
     {
        kGapPc <- r
        break
     }
  }
  pos <- list(kGapUnif=kGapUnif,kGapPc=kGapPc)
  pos #restituisce il valore k per le statistiche GapUnif e GapPC
}
```

A.5 Statistica Silhouette media

```
sil.tte <- function(x)
{
   #x: dati in input
   k <- 11
   valS <- NULL #vettore per contenere i valori della statistica

   deq <- dist(x)
   for(i in 2:k)
   {
      cl <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      wi <- sum(cl$withinss)

      for(z in 1:19)
      {
         ab <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
         if(wi>sum(ab$withinss))
         {
            cl <- ab
            wi <- sum(ab$withinss)
         }
      }
      #La funzione silhouette si trova nella libreria ‘‘cluster’’
      sil <- silhouette(cl$cluster,deq) #calcolo della statistica silhouette
      val <- mean(sil[,3]) #media di tutti i valori s(i)
      valS <- c(valS,val)
   }

   max <- valS[1]
   pos <- 2
   for(j in 2:length(valS))
   {
      if(valS[j]>max)
      {
         max <- valS[j]
         pos <- j+1
      }
   }
   pos #valore k scelto
}
```

A.6 Statistica Split Silhouette Media

La parte compresa tra BEGIN...END, è stata scritta dopo aver analizzato la funzione `labelstomss` contenuta nella libreria `hopach` che calcola la split silhouette.

```
mss <- function(x,dist)
```

```
{
   #x: dati in input
   #dist: matrice delle distanze
   #la matrice dist si ottiene dalla funzione distancematrix, contenuta nella libreria ‘‘hopach’’)
   k <- 11
   valS <- NULL #vettore per contenere i valori della statistica silhouette
   valMSS <- NULL #vettore per contenere i valori della statistica SSM

   deq <- dist(x)
   for(i in 2:k)
   {
      cl <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      wi <- sum(cl$withinss)

      for(z in 1:19)
      {
         ab <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
         if(wi>sum(ab$withinss))
         {
            cl <- ab
            wi <- sum(ab$withinss)
         }
      }
      #La funzione silhouette si trova nella libreria ‘‘cluster’’
      sil <- silhouette(cl$cluster,deq)
      val <- mean(sil[,3]) #media di tutti i valori s(i)
      valS <- c(valS,val)
      #BEGIN
      ssi <- NULL
      p <- dist@Size
      unlabels <- sort(unique(cl$cluster))
      vk <- length(unlabels)
      for(j in 1:vk)
      {
         labs <- (1:p)[cl$cluster==unlabels[j]] #indice delle osservazioni
         pp <- length(labs)
         if(pp<3)
            ssi[j] <- NA
         else
         {
            #La funzione silh.tte è riportata di seguito
            bestk <- silh.tte(x,min(9,(length(labs)-1),na.rm=TRUE))
            dati <- x[labs,]
            #La funzione ss è riportata di seguito
            ssi[j] <- ss(dati,bestk)
         }
      }
      if(sum(is.na(ssi))==k)
         valMSS <- c(valMSS,NA)
      else
         valMSS <- c(valMSS,mean(ssi,na.rm=TRUE))
      #END
```

```
   }
   maxS1 <- max(valS)
   valMSS <- c(maxS1,valMSS)
   valMin <- valMSS[1]
   pos <- 1
   for(z in 2:length(valMSS))
   {
      if(valMSS[z]<valMin)
      {
         valMin <- valMSS[z]
         pos <- z
      }
   }
   pos #valore k scelto
}

silh.tte <- function(x,k)
{
   #x: dati
   #k: valore per k
   valS <- NULL
   deq <- dist(x)
   for(i in 2:k)
   {
      cl <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      #La funzione silhouette si trova nella libreria ‘‘cluster’’
      sil <- silhouette(cl$cluster,deq)
      val <- mean(sil[,3]) #media di tutti i valori s(i)
      valS <- c(valS,val)
   }
   max <- valS[1]
   pos <- 2
   if(length(valS)>1)
   {
      for(j in 2:length(valS))
      {
         if(valS[j]>max)
         {
            max <- valS[j]
            pos <- j+1
         }
      }
   }
   pos #valore k scelto
}

ss <- function(x,k)
{
   #x: dati
   #k: valore per k
   valS <- NULL
   deq <- dist(x)
```

```
   for(i in 2:k)
   {
      cl <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      sil <- silhouette(cl$cluster,deq)
      val <- mean(sil[,3]) #media di tutti i valori s(i)
      valS <- c(valS,val)
   }
   maxS <- max(valS)
   maxS
}
```

A.7 Metodo jump

```
jump.method <- function (x)
{
   #x: dati in input
   k <- 11
   p <- ncol(x)
   y <- p/2

   dk <- NULL # vettore che contiene i valori della distorsione
   for(i in 1:k)
   {
      cl <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      wi <- sum(cl$withinss)
      n <- sum(cl$size)
      for(z in 1:19)
      {
         ab <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
         if(wi>sum(ab$withinss))
         {
            cl <- ab
            wi <- sum(ab$withinss)
            n <- sum(ab$size)
         }
      }
      val <- (wi/n)/p
      dk <- c(dk,val)
   }

   dky <- dk^(-y) #vettore della distorsione trasformata
   jump <- NULL #vettore dei salti
   for(i in 1:k)
   {
      if(i==1)
         jump <- c(jump,dky[i]) #ossia jump<-c(jump,(dk[i]-dk0)) dove dk0=0
      jump <- c(jump,(dky[i]-dky[i-1]))
   }

   max <- jump[1]
```

```
   pos <- 1
   for(i in 2:length(jump))
   {
      if(jump[i]>max)
      {
         max <- jump[i]
         pos <- i
      }
   }
   pos #valore k scelto
}
```

A.8 Statistica Gap pesata

La funzione `w.gap` riportata di seguito è stata ottenuta modificando la funzione `index.Gap`, contenuta nella libreria `clusterSim`.
Tale funzione calcola il valore definito nella formula (2.11).

```
w.gap <- function (x, clall, reference.distribution = "unif", B = 10, method = "pam")
{
   #x: dati in input
   #clall: due vettori di interi, che indicano il cluster al quale ogni oggetto
   #       è allocato nelle partizioni in k, e k+1 cluster

   GAP <- function(X, cl, referenceDistribution, B, method)
   {
      simgap <- function(Xvec)
      {
         ma <- max(Xvec)
         mi <- min(Xvec)
         Xout <- runif(length(Xvec), min = mi, max = ma)
         return(Xout)
      }

      pcsim <- function(X)
      {
         Xmm <- apply(X, 2, mean)
         for (k in (1:dim(X)[2]))
         {
            X[, k] <- X[, k] - Xmm[k]
         }
         ss <- svd(X)
         Xs <- X %*% ss$v
         Xnew <- apply(Xs, 2, simgap)
         Xt <- Xnew %*% t(ss$v)
         for (k in (1:dim(X)[2]))
         {
            Xt[, k] <- Xt[, k] + Xmm[k]
         }
```

```
   return(Xt)
}

ClassNr <- max(cl)
Wk0 <- 0
WkB <- matrix(0, 1, B)

for (bb in (1:B))
{
   if (reference.distribution == "unif")
      Xnew <- apply(X, 2, simgap)
   else if (reference.distribution == "pc")
      Xnew <- pcsim(X)
   else stop("Wrong reference distribution type")

   if (bb == 1)
   {
      pp <- cl
      if (ClassNr == length(cl))
         pp2 <- 1:ClassNr
      else if (method == "pam")
         pp2 <- pam(Xnew, ClassNr)$cluster
      else if (method == "k-means")
         pp2 <- kmeans(Xnew, ClassNr, 100, algorithm="MacQueen")$cluster
      else stop("Wrong clustering method")

      if (ClassNr > 1)
      {
         for (zz in (1:ClassNr))
         {
            Xuse <- X[pp == zz, ]
            Wk0 <- Wk0 + sum(diag(var(Xuse)))
            Xuse2 <- Xnew[pp2 == zz, ]
            WkB[1,bb] <- WkB[1,bb]+sum(diag(var(Xuse2)))
         }
      }

      if (ClassNr == 1)
      {
         Wk0 <- sum(diag(var(x)))
         WkB[1, bb] <- sum(diag(var(Xnew)))
      }
   }

   if (bb > 1)
   {
      if (ClassNr == length(cl))
         pp2 <- 1:ClassNr
      else if (method == "pam")
         pp2 <- pam(Xnew, ClassNr)$cluster
      else if (method == "k-means")
         pp2 <- kmeans(Xnew, ClassNr, 100, algorithm="MacQueen")$cluster
```

```
            else stop("Wrong clustering method")

            if (ClassNr > 1)
            {
               for (zz in (1:ClassNr))
               {
                  Xuse2 <- Xnew[pp2 == zz, ]
                  WkB[1,bb] <- WkB[1,bb]+sum(diag(var(Xuse2)))
               }
            }

            if (ClassNr == 1)
            {
               WkB[1, bb] <- sum(diag(var(Xnew)))
            }
         }
      }

      Wgap <- mean(log(WkB[1, ])) - log(Wk0)
      Wdgap <- sqrt(1 + 1/B) * sqrt(var(log(WkB[1, ]))) * sqrt((B - 1)/B)
      resul <- list(Wgap = Wgap, Wdgap = Wdgap)
      #resul <- list(Wgap = Wgap, Wk0 = Wk0, Wkb = WkB)
      resul
   }

   X <- as.matrix(x)
   gap1 <- GAP(X, clall[, 1], reference.distribution, B, method)
   gap <- gap1$Wgap
   gap2 <- GAP(X, clall[, 2], reference.distribution, B, method)
   diffu <- gap - (gap2$Wgap - gap2$Wdgap)
   diffu
}
```

`w.gap.unif.pc` restituisce il valore k per le statistiche WGapUnif e WGapPC.

```
w.gap.unif.pc <- function(x)
{
   #x: dati in input
   k <- 11
   indWGapUnif <- NULL #vettore per contenere i valori della statistica WGapUnif
   indWGapPc <- NULL #vettore per contenere i valori della statistica WGapPC

   for(i in 1:k)
   {
      if(i==1)
         cl1 <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      else
         cl1 <- cl2
      cl2 <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
      wi1 <- sum(cl1$withinss)
      wi2 <- sum(cl2$withinss)
```

```
      for(z in 1:19)
      {
         if(i==1)
         {
            cl1b <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
            if(wi1>sum(cl1b$withinss))
            {
               cl1 <- cl1b
               wi1 <- sum(cl1b$withinss)
            }
         }
         cl2b <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
         if(wi2>sum(cl2b$withinss))
         {
            cl2 <- cl2b
            wi2 <- sum(cl2b$withinss)
         }
      }
      clall <- cbind(cl1$cluster,cl2$cluster)
      valUnif <- w.gap(x,clall,reference.distribution="unif",B=30,method="k-means")
      valPc <- w.gap(x,clall,reference.distribution="pc",B=30,method="k-means")
      indWGapUnif <- c(indWGapUnif,valUnif)
      indWGapPc <- c(indWGapPc,valPc)
   }

   #Scelta del valore k
   for(z in 1:length(indWGapUnif))
   {
      if(indWGapUnif[z]>0)
      {
         kWGapUnif <- z
         break
      }
   }
   for(r in 1:length(indWGapPc))
   {
      if(indWGapPc[r]>0)
      {
         kWGapPc <- r
         break
      }
   }
   pos <- list(kWGapUnif=kWGapUnif,kWGapPc=kWGapPc)
   pos #restituisce il valore k per le statistiche WGapUnif e WGapPC
}
```

A.9 Statistica DD-Gap pesata

Si modifica la funzione `w.gap` nel seguente modo: alla fine, al posto di usare

```
gap1 <- GAP(X, clall[, 1], reference.distribution, B, method)
gap <- gap1$Wgap
gap2 <- GAP(X, clall[, 2], reference.distribution, B, method)
diffu <- gap - (gap2$Wgap - gap2$Wdgap)
diffu
```

si usano solo le istruzioni

```
gap1 <- GAP(X, clall, reference.distribution, B, method)
#clall: è un vettore di interi che indica il cluster al quale un oggetto è allocato
gap <- gap1$Wgap
gap
```

La funzione `d.dd.gap` restituisce il valore k per le statistiche DDGapUnif e DDGapPC.

```
d.dd.gap <- function (x)
{
   #x: dati in input
   k <- 11
   p <- dim(x)[2]
   ddgapUnif <- NULL #vettore per contenere i valori della statistica DDGapUnif
   ddgapPc <- NULL #vettore per contenere i valori della statistica DDGapPC

   for(i in 2:k)
   {
      if(i==2)
      {
         cl1 <- rep(1,nrow(x))
         cl2 <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      }
      else
      {
         cl1 <- cl2
         cl2 <- cl3
      }
      cl3 <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
      wi2 <- sum(cl2$withinss)
      wi3 <- sum(cl3$withinss)

      for(z in 1:19)
      {
         if(i==2)
         {
            cl2b <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
            if(wi2>sum(cl2b$withinss))
```

```
         {
            cl2 <- cl2b
            wi2 <- sum(cl2b$withinss)
         }
      }
      cl3b <- kmeans(x,(i+1),iter.max=100,nstart=(i+1),algorithm="MacQueen")
      if(wi3>sum(cl3b$withinss))
      {
         cl3 <- cl3b
         wi3 <- sum(cl3b$withinss)
      }
   }

   if(i==2)
   {
      clu1 <- cl1
      clu2 <- cl2$cluster
      clu3 <- cl3$cluster
   }
   else
   {
      clu1 <- cl1$cluster
      clu2 <- cl2$cluster
      clu3 <- cl3$cluster
   }

   k0 <- w.gap(x,clu1,reference.distribution="unif",B=30,method="k-means") #k-1
   k1 <- w.gap(x,clu2,reference.distribution="unif",B=30,method="k-means") #k
   k2 <- w.gap(x,clu3,reference.distribution="unif",B=30,method="k-means") #k+1
   dk <- k1-k0
   dk1 <- k2-k1
   ddk <- dk-dk1

   k0pc <- w.gap(x,clu1,reference.distribution="pc",B=30,method="k-means") #k-1
   k1pc <- w.gap(x,clu2,reference.distribution="pc",B=30,method="k-means") #k
   k2pc <- w.gap(x,clu3,reference.distribution="pc",B=30,method="k-means") #k+1
   dkpc <- k1pc-k0pc
   dk1pc <- k2pc-k1pc
   ddkpc <- dkpc-dk1pc

   ddgapUnif <- c(ddgapUnif,ddk)
   ddgapPc <- c(ddgapPc,ddkpc)
}
#Scelta del valore di k
mddUnif <- ddgapUnif[1]
kddGapUnif <- 2
mddPc <- ddgapPc[1]
kddGapPc <- 2
if(length(ddgapUnif)==length(ddgapPc))
{
   for (j in 2:length(ddgapUnif))
   {
```

```
         if(ddgapUnif[j]>mddUnif)
         {
            mddUnif <- ddgapUnif[j]
            kddGapUnif <- j+1
         }
         if(ddgapPc[j]>mddPc)
         {
            mddPc <- ddgapPc[j]
            kddGapPc <- j+1
         }
      }
   }
   else stop("Wrong: qualcosa non và!")

   pos <- list(kddGapUnif=kddGapUnif,kddGapPc=kddGapPc)
   pos #restituisce il valore k per le statistiche DDGapUnif e DDGapPC
}
```

A.10 Prediction strength

```
ps.ps <- function(x,B=30)
{
   #x: dati in input
   #B: numero di divisioni casuali
   k <- 7
   n <- nrow(x)
   n1 <- round(n*0.50)
   ps <- NULL #vettore per contenere i valori della statistica PS
   for(j in 2:k)
   {
      ps.ave <- NULL
      for(i in 1:B)
      {
         permuta <- sample(1:n,n)
         stima <- sort(permuta[1:n1])
         verifica <- sort(permuta[(n1+1):n])
         x.stima <- x[stima,] #dataset per la stima
         x.test <- x[verifica,] #dataset per la verifica

         cl1 <- kmeans(x.stima,j,iter.max=100,nstart=j,algorithm=''MacQueen'')
         cl2 <- kmeans(x.test,j,iter.max=100,nstart=j,algorithm=''MacQueen'')
         #La funzione assign.clusters è riportata di seguito
         cl3 <- assign.clusters(x.test,cl1$centers)
         ClassNr <- max(cl2$cluster)
         ff <- 0
         ii <- 1
         psk <- NULL
         for(r in 1:ClassNr)
         {
            nkr <- cl2$size[r]
```

```
            if(r==1)
            {
               ff <- ff+cl2$size[r]
               clu1 <- cl2$cluster[1:ff]
               clu2 <- cl3[1:ff]
               Cone <- matrix(clu1,nr=nkr,nc=nkr)==matrix(clu1,nr=nkr,nc=nkr,byrow=TRUE)
               Ctwo <- matrix(clu2,nr=nkr,nc=nkr)==matrix(clu2,nr=nkr,nc=nkr,byrow=TRUE)
               Cint <- Cone==Ctwo
               diag(Cint) <- 0
            }
            else
            {
               ii <- ii+cl2$size[r-1]
               ff <- ff+cl2$size[r]
               clu1 <- cl2$cluster[ii:ff]
               clu2 <- cl3[ii:ff]
               Cone <- matrix(clu1,nr=nkr,nc=nkr)==matrix(clu1,nr=nkr,nc=nkr,byrow=TRUE)
               Ctwo <- matrix(clu2,nr=nkr,nc=nkr)==matrix(clu2,nr=nkr,nc=nkr,byrow=TRUE)
               Cint <- Cone==Ctwo
               diag(Cint) <- 0
            }
            if(nkr>1)
            {
               val <- sum(Cint)/(nkr*(nkr-1))
               psk <- c(psk,val)
            }
         }
         ps.ave <- c(ps.ave,min(psk,na.rm=TRUE))
      }
      mean.ps.ave <- mean(ps.ave,na.rm=TRUE)
      ps <- c(ps,mean.ps.ave)
   }
   ps <- c(1,ps)
   ma <- 0
   if(length(ps)==k)
   {
      for(z in 1:length(ps))
      {
         if(ps[z]>0.85)
         {
            pos <- z
         }
      }
   }
   else stop("Qualcosa non và...")
   pos #valore k scelto
}

assign.clusters <- function(X,centers)
{
   n <- nrow(X)
   p <- ncol(X)
```

```
   c <- dim(centers)[1]

   assign <- NULL
   for(i in 1:n)
   {
      min.dist <- -1
      assign[i] <- 0
      for(k in 1:c)
      {
         dist.k <- sqr.euclid(X[i,],centers[k,])
         if(min.dist == -1 || min.dist > dist.k)
         {
            min.dist <- dist.k
            assign[i] <- k
         }
      }
   }
   return(assign)
}

dove sqr.euclid <- function(x,y)
                   {
                      n <- length(x)
                      if(n > length(y))
                         n <- length(y)
                      return(sum((x[1:n] - y[1:n])^2))
                   }
```

A.11 Statistica Clest

```
clest <- function(x,B=10,B0=10)
{
   #x: dati in input
   #DISTRIBUZIONE UNIFORME
   simUNIF <- function(Xvec)
   {
      ma <- max(Xvec)
      mi <- min(Xvec)
      Xout <- runif(length(Xvec), min = mi, max = ma)
      return(Xout)
   }

   k <- 5 #suggerimento indicato dagli autori
   n <- nrow(x)
   n1 <- round(n*0.75)
   tk <- NULL #vettore dei valori delle mediane
   dk <- NULL #vettore delle differenze
   pk <- NULL #vettore dei p-value
   for(j in 2:k)
   {
```

```
skb <- NULL #vettore delle similarità
for(i in 1:B)
{
   permuta <- sample(1:n,n)
   stima <- sort(permuta[1:n1])
   verifica <- sort(permuta[(n1+1):n])
   x.stima <- x[stima,] #dataset per la stima
   x.test <- x[verifica,] #dataset per la verifica

   cl1 <- kmeans(x.stima,j,iter.max=100,nstart=j,algorithm="MacQueen")
   cl2 <- kmeans(x.test,j,iter.max=100,nstart=j,algorithm="MacQueen")
   #Vedi la funzione stat.diag.da di seguito
   clpred <- stat.diag.da(x.stima,cl1$cluster,x.test,pool=1,cl1$size)$pred #classifier DLDA
   #caricare la libreria ‘‘clv’’
   std <- std.ext(cl2$cluster,clpred)
   skb[i] <- clv.Folkes.Mallows(std)
}
tk[j-1] <- median(skb) #similarità mediana per la partizione k

tk0 <- NULL
for(z in 1:B0)
{
   skb0 <- NULL
   xnew <- apply(x,2,simUNIF)
   for(zz in 1:B)
   {
      permuta <- sample(1:n,n)
      stima <- sort(permuta[1:n1])
      verifica <- sort(permuta[(n1+1):n])
      x.stima <- xnew[stima,] #dataset per la stima
      x.test <- xnew[verifica,] #dataset per la verifica

      cl1 <- kmeans(x.stima,j,iter.max=100,nstart=j,algorithm="MacQueen")
      cl2 <- kmeans(x.test,j,iter.max=100,nstart=j,algorithm="MacQueen")
      #Vedi la funzione stat.diag.da di seguito
      clpred <- stat.diag.da(x.stima,cl1$cluster,x.test,pool=1,cl1$size)$pred #classifier DLDA
      #caricare la libreria ‘‘clv’’
      std <- std.ext(cl2$cluster,clpred)
      skb0[zz] <- clv.Folkes.Mallows(std)
   }
   tk0[z] <- median(skb0)
}
tk0mean <- mean(tk0)
ind <- 0
if(length(tk0)==B0)
{
   for(y in 1:B0)
   {
      if(tk0[y]>=tk[j-1])
      ind <- ind+1
   }
}
```

```
      else stop("Qualcosa non và...!")

      pk[j-1] <- ind/B0
      dk[j-1] <- tk[j-1]-tk0mean
   }
   pmax <- 0.05
   dmin <- 0.05

   #Scelta del valore k
   ok <- 0
   for(w in 1:length(pk))
   {
      if((pk[w]<=pmax)&&(dk[w]>=dmin))
      {
         ok <- 1
         break
      }
   }
   if(ok==0)
      pos <- 1
   else
   {
      ma <- dk[1]
      pos <- 2
      for(r in 2:length(dk))
      {
         if(dk[r]>ma)
         {
            ma <- dk[r]
            pos <- r+1
         }
      }
   }
   pos #valore k scelto
}

#Caricare la libreria ‘‘sma’’. Sono state apportate alcune modifiche
#per migliorarne il funzionamento
stat.diag.da <- function (ls, cll, ts, pool = 1,size)
{
   #ls: dati dell'insieme di stima (x.stima)
   #cll: vettore di interi che indica le etichette di classe per l'insieme stima (cl1$cluster)
   #ts: dati dell'insieme test (x.test)
   #pool: per predittore lineare mettere 1, per quello quadratico 0
   #size: vettore che contiene il numero di osservazioni per cluster
   ls <- as.matrix(ls)
   ts <- as.matrix(ts)
   n <- nrow(ls)
   p <- ncol(ls)
   K <- max(cll)
   nk <- rep(0,K)
   m <- matrix(0, K, p)
```

```
   v <- matrix(0, K, p)
   disc <- matrix(0, nrow(ts), K)
   for (k in (1:K))
   {
      which <- (1:n)[cll==k]
      nk[k] <- size[k]
      if(nk[k]>1)
      {
         m[k, ] <- apply(ls[which, ], 2, mean.na)
         v[k, ] <- apply(ls[which, ], 2, var.na)
      }
   }
   vp <- apply(v, 2, function(z) sum.na((nk - 1) * z)/(n - K))
   if (pool == 1)
   {
      for (k in (1:K)) disc[, k] <- apply(ts, 1, function(z) sum.na((z -
                                    m[k, ])^2/vp))
   }
   if (pool == 0)
   {
      for (k in (1:K)) disc[, k] <- apply(ts, 1, function(z) (sum.na((z -
                                    m[k, ])^2/v[k, ]) + sum.na(log(v[k, ]))))
   }
   pred <- apply(disc, 1, function(z) (min(cll):max(cll))[order.na(z)[1]])
   list(pred = pred)
}
```

A.12 Metodo jump-differenza

Per l'implementazione della funzione `jm.diff`, si fà riferimento al codice del metodo jump (pag. 59).
L'unica differenza, è cambiare le seguenti istruzioni

```
for(i in 1:k)
{
   if(i==1)
      jump <- c(jump,dky[i]) #ossia jump<-c(jump,(dk[i]-dk0)) dove dk0=0
   jump <- c(jump,(dky[i]-dky[i-1]))
}
```

con queste

```
for(i in 1:(k-1))
{
   if(i==1)
      jk <- dky[i] #ossia jump<-c(jump,(dk[i]-dk0)) dove dk0=0
   else
      jk <- dky[i]-dky[i-1]
```

```
   jk1 <- dky[i+1]-dky[i]
   jump <- c(jump,(jk-jk1))
}
```

A.13 Metodo jump con Γ stimata

```
jm.mala <- function (x,varCov)
{
   #x: dati in input
   #varCov: matrice di covarianza comune
   k <- 11
   p <- ncol(x)
   y <- p/2

   dk <- NULL # vettore che contiene i valori della distorsione
   for(i in 1:k)
   {
      d <- NULL #vettore
      cl <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
      with <- sum(cl$withinss)
      for(z in 1:19)
      {
         ab <- kmeans(x,i,iter.max=100,nstart=i,algorithm="MacQueen")
         if(with>sum(ab$withinss))
         {
            cl <- ab
            with <- sum(ab$withinss)
         }
      }
      for(j in 1:i)
      {
         distanza <- mahalanobis(x[cl$cluster==j,],cl$centers[j,],varCov)
         val <- sum(distanza)
         d <- c(d,val)
      }
      wi <- sum(d)
      nn <- sum(cl$size)
      val <- (wi/nn)/p
      dk <- c(dk,val)
   }

   dky <- dk^(-y) #vettore della distorsione trasformata
   jump <- NULL #vettore dei salti
   for(i in 1:k)
   {
      if(i==1)
         jump <- c(jump,dky[i]) #ossia jump<-c(jump,(dk[i]-dk0)) dove dk0=0
      jump <- c(jump,(dky[i]-dky[i-1]))
   }
```

```
   max <- jump[1]
   pos <- 1
   for(i in 2:length(jump))
   {
      if(jump[i]>max)
      {
         max <- jump[i]
         pos <- i
      }
   }
   pos #valore k scelto
}
```

A.14 Metodo jump-differenza con Γ stimata

Per l'implementazione della funzione `jm.mala.diff`, si fà riferimento al codice del metodo jump con Γ stimata, riportato nel paragrafo precedente.
L'unica differenza, è cambiare le seguenti istruzioni

```
for(i in 1:k)
{
   if(i==1)
      jump <- c(jump,dky[i]) #ossia jump<-c(jump,(dk[i]-dk0)) dove dk0=0
   jump <- c(jump,(dky[i]-dky[i-1]))
}
```

con queste

```
for(i in 1:(k-1)) #use i+1...
{
   if(i==1)
      jk <- dky[i] #ossia jump<-c(jump,(dk[i]-dk0)) dove dk0=0
   else
      jk <- dky[i]-dky[i-1]
   jk1 <- dky[i+1]-dky[i]
   jump <- c(jump,(jk-jk1))
}
```

A.15 Calcolo dei valori k

Le istruzioni utilizzate per ottenere da ogni metodo la stima del valore k per ogni data set simulato sono riportate di seguito. Le librerie da caricare sono: `MASS`, `cluster`, `clv`, `clusterSim`, `sma`, e `hopach`

```
#COMUNI A TUTTI I METODI
rip <- 50 #numero di dataset
noss <- 100 #da cambiare in base al numero di osservazioni del dataset

#AL POSTO DI ‘‘NomeMetodo’’, si sostituisce una delle seguenti funzioni:
# ch(dati), h(dati), kl(dati), sil.tte(dati), jump.method(dati), jm.diff(dati),
# ps.ps(dati), clest(dati,20,20), mss(dati,mydist)
valori <- NULL
for(i in 1:rip)
{
   if(i==1)
      ii <- 1
   else
      ii <- 1+(noss*(i-1))

   dati <- x[ii:(noss*i),] #x contiene tutti i dataset simulati
   #mydist <- distancematrix(dati,d="euclid") #SOLO PER IL METODO mss
   #al posto di ‘‘NomeMetodo’’, và inserito uno dei metodi citati sopra
   f <- NomeMetodo #restituisce il valore k
   valori <- c(valori,f)
}
### Vettore con i valori di k
fVAL <- NULL
for(k in 1:11)
{
   fVAL <- c(fVAL,length(valori[valori==k]))
}
fVAL
```

`fVAL` è quindi il vettore che contiene la performance ottenuta da un determinato metodo. Il primo valore corrisponde al numero di volte che il metodo ha stimato $k = 1$, il secondo valore al numero di volte che ha stimato $k = 2$, e così via.
Per le funzioni `gap.unif.pc`, `w.gap.unif.pc` e `d.dd.gap`, che restituiscono due valori k, si sono utilizzate le seguenti istruzioni:

```
#AL POSTO DI ‘‘NomeMetodo’’, si sostituisce una delle seguenti funzioni:
# gap.unif.pc(dati), w.gap.unif.pc(dati), d.dd.gap(dati)
valoriUnif <- NULL
valoriPC <- NULL
for(i in 1:rip)
{
   if(i==1)
```

```
      ii <- 1
   else
      ii <- 1+(noss*(i-1))

   dati <- x[ii:(noss*i),] #x contiene tutti i dataset simulati
   #al posto di ‘‘NomeMetodo’’, và inserito uno dei metodi citati sopra
   f <- NomeMetodo #restituisce il valore k
   valoriUnif <- c(valoriUnif,f$kGapUnif) #f$... varia in base al metodo usato
   valoriPC <- c(valoriPC,f$kGapPc) #f$... varia in base al metodo usato
}
### Vettore con i valori di k
fVALUnif <- NULL
fVALPc <- NULL
for(k in 1:11)
{
   fVALUnif <- c(fVALUnif,length(valoriUnif[valoriUnif==k]))
   fVALPc <- c(fVALPc,length(valoriPC[valoriPC==k]))
}
fVALUnif
fVALPc
```

Per le funzioni `jm.mala` e `jm.mala.diff`, che richiedono in input la matrice di covarianza comune stimata, si procede nel seguente modo:

```
valk <- 7 #nella teoria è il valore R
valori <- NULL
for(i in 1:rip)
{
   if(i==1)
      ii <- 1
   else
      ii <- 1+(noss*(i-1))
   dati <- x[ii:(noss*i),]

   valuesK <- NULL
   bgssValues <- NULL #vettore delle varianze tra i gruppi
   for(k in 1:valk)
   {
      cl <- kmeans(dati,k,iter.max=100,nstart=k,algorithm="MacQueen")
      with <- sum(cl$withinss)
      for(j in 1:19)
      {
         ab <- kmeans(dati,k,iter.max=100,nstart=k,algorithm="MacQueen")
         if(with>sum(ab$withinss))
         {
            cl <- ab
            with <- sum(ab$withinss)
         }
      }
      #La funzione cov.comune si trova di seguito
      varCov <- cov.comune(dati,cl,k) #stima della matrice di covarianza comune
```

```
      #La funzione gss si trova di seguito
      rv <- gss(dati,cl$size,cl$withinss)
      bgssValues <- c(bgssValues,rv$b)
      f <- jm.mala.diff(dati,varCov) #restituisce il valore di k
      #f <- jm.mala(dati,varCov) #restituisce il valore di k
      valuesK <- c(valuesK,f)
   }

   #La funzione trovaK si trova di seguito
   fK <- trovaK(bgssValues,valuesK) #scelta del valore k con il criterio definito

   valori <- c(valori,fK)
}
### Vettore con i valori di k
fVAL <- NULL
for(k in 1:11)
{
   fVAL <- c(fVAL,length(valori[valori==k]))
}
fVAL

cov.comune <- function(x,cl,i)
{
   #x: dati
   #cl: risultato dell'algoritmo di clustering
   #i: valore di k corrente
   n <- nrow(x)
   cm <- 0 #covarianza comune
   for(j in 1:i)
   {
      ni <- length(cl$cluster[cl$cluster==j])
      dati <- x[cl$cluster==j,]
      if(ni>1)
         covarianza_i <- var(dati)*(ni-1)
      cm <- cm+covarianza_i
   }
   cm <- cm/(n-i)
   cm
}

gss <- function(x,clsize,withins)
{
   n <- sum(clsize)
   k <- length(clsize)
   allmean <- apply(x,2,mean)
   dmean <- sweep(x,2,allmean,"-")
   allmeandist <- sum(dmean^2)
   wgss <- sum(withins)
   bgss <- allmeandist-wgss
   result <- list(b=bgss,w=wgss,t=(bgss+wgss))
   result
}
```

```
trovaK <- function(Bv,vk)
{
   #Bv: vettore della varianza tra i gruppi
   #vk: vettore dei valori k
   if(length(Bv)!=length(vk))
      stop ("Non funziona correttamente")

   ratiok <- Bv/vk
   ris <- as.matrix(cbind(vk,ratiok))
   maxRatiok <- max(ratiok)
   for(i in 1:nrow(ris))
   {
      if(ris[i,2]==maxRatiok)
      {
         pos <- i
         break
      }
   }
   sceltaK <- ris[i,1]
   sceltaK
}
```

A.16 Scenari di simulazione

Le istruzioni utilizzate per la simulazione degli scenari sono:

```
# PRIMO SCENARIO ###
x <- NULL
for(i in 1:50) #50 dataset da 200 osservazioni in 10 dimensioni
{
   xx <- NULL
   for(j in 1:200)
   {
      y <- runif(10)
      xx <- rbind(xx,y)
   }
   x <- rbind(x,xx)
}
x <- as.matrix(x)

# SECONDO SCENARIO ###
cov <- matrix(c(1,0,0,1),2,2)
x <- NULL
for(i in 1:50) #50 dataset da 100 osservazioni
{
   x1 <- mvrnorm(25,c(0,0),cov)
   x2 <- mvrnorm(25,c(0,5),cov)
   x3 <- mvrnorm(50,c(5,-3),cov)
   y <- rbind(x1,x2,x3)
```

```
   x <- rbind(x,y)
}
x <- as.matrix(x)

# TERZO SCENARIO ###
x <- NULL
sig1 <- matrix(c(4,1.7,1.7,1),2,2)
sig2 <- matrix(c(0.25,0,0,0.25),2,2)
sig3 <- matrix(c(4,-1.7,-1.7,1),2,2)
for(i in 1:50) # 50 dataset da 150 osservazioni
{
   x1 <- mvrnorm(50,c(0,0),sig1)
   x2 <- mvrnorm(50,c(0,3),sig2)
   x3 <- mvrnorm(50,c(4,3),sig3)
   y <- rbind(x1,x2,x3)
   x <- rbind(x,y)
}
x <- as.matrix(x)

# QUARTO SCENARIO ###
x123 <- seq(-0.5,0.5,length=100)
y123 <- seq(9.5,10.5,length=100)
x <- NULL
for(j in 1:50) #50 dataset da 200 osservazioni
{
   x1 <- NULL
   x2 <- NULL
   x3 <- NULL
   y1 <- NULL
   y2 <- NULL
   y3 <- NULL
   for(i in 1:100)
   {
      x1[i] <- x123[i]+rnorm(1,0,0.1)
      x2[i] <- x123[i]+rnorm(1,0,0.1)
      x3[i] <- x123[i]+rnorm(1,0,0.1)
      y1[i] <- y123[i]+rnorm(1,0,0.1)
      y2[i] <- y123[i]+rnorm(1,0,0.1)
      y3[i] <- y123[i]+rnorm(1,0,0.1)
   }
   clu1 <- cbind(x1,x2,x3)
   clu2 <- cbind(y1,y2,y3)
   xy <- rbind(clu1,clu2)
   x <- rbind(x,xy)
}
x <- as.matrix(x)

# QUINTO SCENARIO ###
cov <- matrix(c(1,0.5,0.5,0.5,1,0.5,0.5,0.5,1),3,3)
cov
covnoise <- matrix(c(1,0,0,0,0,0,0,0,0,0,
                     0,1,0,0,0,0,0,0,0,0,
```

```
                    0,0,1,0,0,0,0,0,0,0,
                    0,0,0,1,0,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,
                    0,0,0,0,0,1,0,0,0,0,
                    0,0,0,0,0,0,1,0,0,0,
                    0,0,0,0,0,0,0,1,0,0,
                    0,0,0,0,0,0,0,0,1,0,
                    0,0,0,0,0,0,0,0,0,1),10,10)

x <- NULL
for(i in 1:50) #50 dataset da 150 osservazioni
{
   x1 <- mvrnorm(50,c(0,0,0),cov)
   x2 <- mvrnorm(50,c(2,-2,2),cov)
   x3 <- mvrnorm(50,c(-2,2,-2),cov)
   x4 <- mvrnorm(150,c(rep(0,10)),covnoise)
   y <- rbind(x1,x2,x3)
   ynoise <- cbind(y,x4)
   x <- rbind(x,ynoise)
}
x <- as.matrix(x)

# SESTO SCENARIO ###
cov1 <- matrix(c(1,0,0,1),2,2)
cov2 <- matrix(c(0.1,0,0,0.1),2,2)
x <- NULL
for(i in 1:50) #50 dataset da 115 osservazioni
{
   y <- NULL
   x1 <- mvrnorm(100,c(0,0),cov1)
   x2 <- mvrnorm(15,c(5,0),cov2)
   y <- rbind(x1,x2)
   x <- rbind(x,y)
}
x <- as.matrix(x)

# SETTIMO SCENARIO ###
cov <- matrix(c(1,0,0,1),2,2)
x <- NULL
for(i in 1:50) #50 dataset da 100 osservazioni
{
   y <- NULL
   x1 <- mvrnorm(20,c(0,0),cov)
   x2 <- mvrnorm(20,c(2.5,2.5),cov)
   x3 <- mvrnorm(20,c(5,5),cov)
   x4 <- mvrnorm(20,c(-2.5,2.5),cov)
   x5 <- mvrnorm(20,c(-5,-5),cov)
   y <- rbind(x1,x2,x3,x4,x5)
   x <- rbind(x,y)
}
x <- as.matrix(x)
```

Bibliografia

[1] Azzalini A, Scarpa B., *Analisi dei dati e data mining*, Milano, Springer, 2004

[2] Bighignoli M., Diana G. (relatore), *Cluster Analysis: determinazione del numero di gruppi*, Tesi di Laurea in Scienze Statistiche ed Economiche, 2005

[3] Calinski R. B., Harabasz J., "A dendrite method for cluster analysis", *Communications in Statistics*, 1974, Vol.3, 1-27

[4] Fowlkes E. B., Mallows C. L., "A method for comparing two hierarchical clusterings", *Journal of the American Statistical Association*, 1983a, Vol.78, 553-569

[5] Fridlyand J., Dudoit S., "A prediction-based resampling method for estimating the number of clusters in a data set", *Genome Biology*, 2002, Vol.3, 1-21

[6] Gordon A. D., *Classification*, 2nd Edition, London, Chapman & Hall/CRC, 1999

[7] Hartigan J. A., *Clustering Algorithms*, New York, Wiley, 1975

[8] Jain A. K., Dubes R. C., *Algorithms for Clustering Data*, Englewood Cliffs, Prentice Hall, 1988

[9] Kaufman L., Rousseeuw P. J., *Finding Groups in Data: An Introduction to Cluster Analysis*, New York, Wiley-Interscience, 1990

[10] Krzanowski W. J., Lai Y. T., "A criterion for determining the number of groups in a data set using sum of squares clustering, *Biometrics*, 1988, Vol.44, 23-34

[11] MacQueen J., "Some methods for classifications and analysis of multivariate observations", *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, Vol.1, 281-297

[12] Marriott F. H. C., "Practical problems in a method of cluster analysis", *Biometrics*, 1971, Vol.27, 501-514

[13] Milligan G. W., Cooper M. C., "An examination of procedures for determining the number of clusters in a data set", *Psychometrica*, 1985, Vol.50, 159-179

[14] Pollard K. S., Van der Laan M. J., "A method to Identify Significant Clusters in Gene Expression Data", *SCI2002 Proceedings*, 2002a, Vol.2, 318-325

[15] Rand W. M., "Objective criteria for the evaluation of clustering methods", *Journal of the American Statistical Association*, 1971, Vol.66, 846-850

[16] Sugar C.A., James G. M., "Finding the Number of Clusters in a Data set: An Information-Theoretic Approach", *Journal of the American Statistical Association*, 2003, Vol.98, 750-763

[17] Tibshirani R., Walther G., "Cluster validation by prediction strength", *Journal of Computational and Graphical Statistics*, 2005, Vol.14, 511-528

[18] Tibshirani R., Walther G., Hastie T., "Estimating the number of data clusters via the gap statistic", *Journal of the Royal Statistical Society*, 2001, Vol.B 63, 411

[19] Yan M., Ye K., "Determining the Number of Clusters Using the Weighted Gap Statistic", *Biometrics*, 2007, Vol.63, 1031-1037

Finito di stampare il 21 ottobre 2008 utilizzando LaTeX 2ε