

3º Assignment – Informatica II, Modulo 1

Lo scopo di questo assignment è quello di analizzare un dataset di immagini contenenti dei numeri scritti a mano, e usare delle tecniche di clustering per catalogare le immagini.

Dataset da analizzare

Il dataset che andiamo ad analizzare è *semeion.csv*. Si tratta di 1593 immagini da 16x16 pixel, che rappresentano numeri scritti a mano. Ogni riga rappresenta un'immagine, e 256 colonne, ognuna delle quali rappresenta un pixel.

Le ultime dieci colonne rappresentano il numero che è rappresentato dall'immagine, questa informazione non verrà utilizzata in fase di clustering, ma sarà utilizzata successivamente, nel momento in cui andremo a valutare le performance dei vari metodi utilizzati.

Clustering

Il clustering è un metodo di analisi dei dati non supervisionato, che permette quindi di raggruppare insiemi di dati in base alle loro caratteristiche. In quanto non supervisionato, non necessita di un insieme di dati di training, ma solo dati di input, che non sono etichettati e non sono presenti informazioni a priori su quali dati devono essere raggruppati insieme.

Il risultato di un clustering è un insieme di cluster ognuno dei quali è formato da serie di dati simili tra loro.

Principal Component Analysis - (PCA)

PCA è un metodo di analisi dei dati che permette di ridurre la dimensionalità di questi, cioè ridurre il numero di caratteristiche che descrivono i dati. Per questo metodo viene utilizzato il teorema di *Reyleigh – Riz* che ci permette di approssimare i massimi e i minimi degli autovalori e degli autovettori della matrice di covarianza.

Dato un insieme di dati, PCA funziona in questo modo:

- calcola la media di tutti i dati.
- calcola la matrice di covarianza.
- calcola gli autovalori e gli autovettori della matrice di covarianza.
- seleziona gli autovettori con gli autovalori più grandi.
- calcola la matrice di proiezione dei dati.
- calcola i dati proiettati.

Un vantaggio di questa tecnica è quella di ridurre il tempo di calcolo, in quanto riduce il numero di operazioni da eseguire. Un altro vantaggio è che se la dimensionalità viene ridotta abbastanza permette di visualizzare i dati in un grafico, e quindi vedere chiaramente se vi è una divisione o meno in cluster principali.

Nel caso dei dati in questione, la dimensionalità dei dati iniziale è 256, vogliamo provare a ridurla. La funzione *PCA* del modulo *sklearn.metrics* ci aiuta in questo procedimento, e le passiamo come parametri il numero di componenti principali che vogliamo osservare. Prima di effettuare PCA, i dati verranno standardizzati utilizzando la funzione *StandardScaler().fit_transform(df)* fornita sempre da sklearn.

Il file che si occupa di questa operazione è *studyPCA.py*.

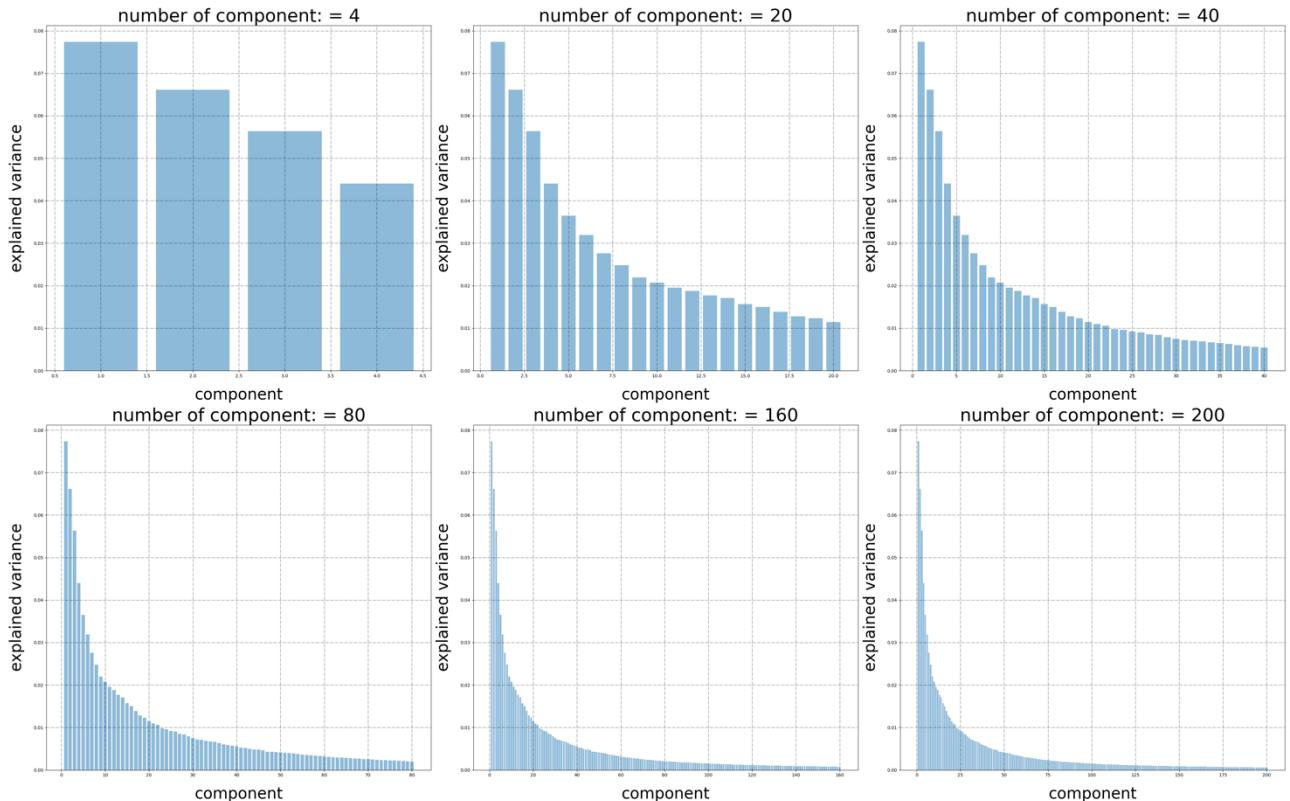


Figura 1: grafici della varianza relativa in funzione del numero di componenti principali scelti tramite PCA.

Dai grafici riportati nell'*immagine 1*, si può notare che superate le venti componenti, la varianza dei dati resta pressoché costante, o diminuisce relativamente poco. Possiamo quindi dire che la maggior parte dell'informazione che il dataset contiene, è contenuta nelle prime venti componenti analizzate tramite PCA, per cui per lo studio che faremo successivamente di clustering, ci limiteremo ad analizzare le prime venti componenti.

Rand Index

Usando il rand index possiamo definire quanto buono sia un clustering rispetto ad uno di riferimento. Il clustering di riferimento che useremo è una partizione dei dati costruita da noi utilizzando le ultime dieci colonne del dataset, che segnano appunto, di che immagine si tratta realmente.

Per il calcolo del Rand Index, è necessario calcolare tre cose:

- A: è il numero di coppie di dati che rappresentano la stessa immagine, e sono stati classificati nello stesso cluster.
- B: è il numero di coppie di dati che rappresentano numeri diversi e che sono stati classificati in cluster diversi.
- N: è il numero di dati nel dataset, in questo caso sono 1593.

Una volta trovati questi valori, il Rand Index verrà calcolato come:

$$R = \frac{2 * (A + B)}{N * (N - 1)}$$

Ad occuparsi di questa operazione, è il file *randind.py* che al suo interno contiene la funzione *rand_index_* che svolge tutti i calcoli del caso.

Analisi dati

Per i modelli che verranno studiati, i dati saranno riportati in grafici in cui i dati vengono rimappati in uno spazio bidimensionale, e i vari cluster contraddistinti da diversi colori.

Si nota in particolare che quando il numero di componenti principali è due, la divisione dei cluster è ben definita, mentre all'aumentare della dimensionalità si nota una sovrapposizione tra questi.

Gaussian Mixture Model

Il primo metodo che verrà implementato è Gaussian Mixture. Si tratta di un modello di clustering probabilistico, che permette di raggruppare i dati in base alle loro caratteristiche. Per il suo utilizzo si assume che i dati siano stati generati secondo un numero finito k di gaussiane con parametri sconosciuti.

La probability density function (PDF) della gaussiana è:

$$f(\mu, \Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} * e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

Dove μ è la media dei dati e Σ è la matrice di covarianza dei dati. Quest'ultima serve a vedere come il cambiamento di varianza in una variabile, influisce sulle altre. Nel modello che utilizzeremo, useremo una matrice di covarianza di tipo diagonale, vale a dire che si assume che le caratteristiche in gioco siano indipendenti tra loro.

La stima dei parametri della gaussiana viene fatta usando l'algoritmo Expectation-Maximization (EM), ovvero si calcola la probabilità che un dato punto appartenga ad una certa gaussiana, e si aggiorna il valore dei parametri della gaussiana in base a questa probabilità.

Come numero di cluster, proviamo una serie di numeri, da 6 a 15. A priori avremmo definito che le classi che devono definire il modello sono dieci, come e cifre utilizzate, ma vedremo che farà difficoltà a separare alcune cifre, ad esempio, si trova di fronte a una difficoltà quando deve scegliere tra il numero tre e il numero otto.

Di seguito vi sono riportati una serie di grafici con vari valori per quanto riguarda PCA, il numero di cluster che utilizza il modello.

In ogni riquadro vi è riportato in basso a destra il tempo utilizzato per il clustering e in basso a sinistra il Rand Index (RI).

L'ultima riga riporta 256 componenti principali, quindi come se non fosse stato effettuato nessun tipo di PCA.

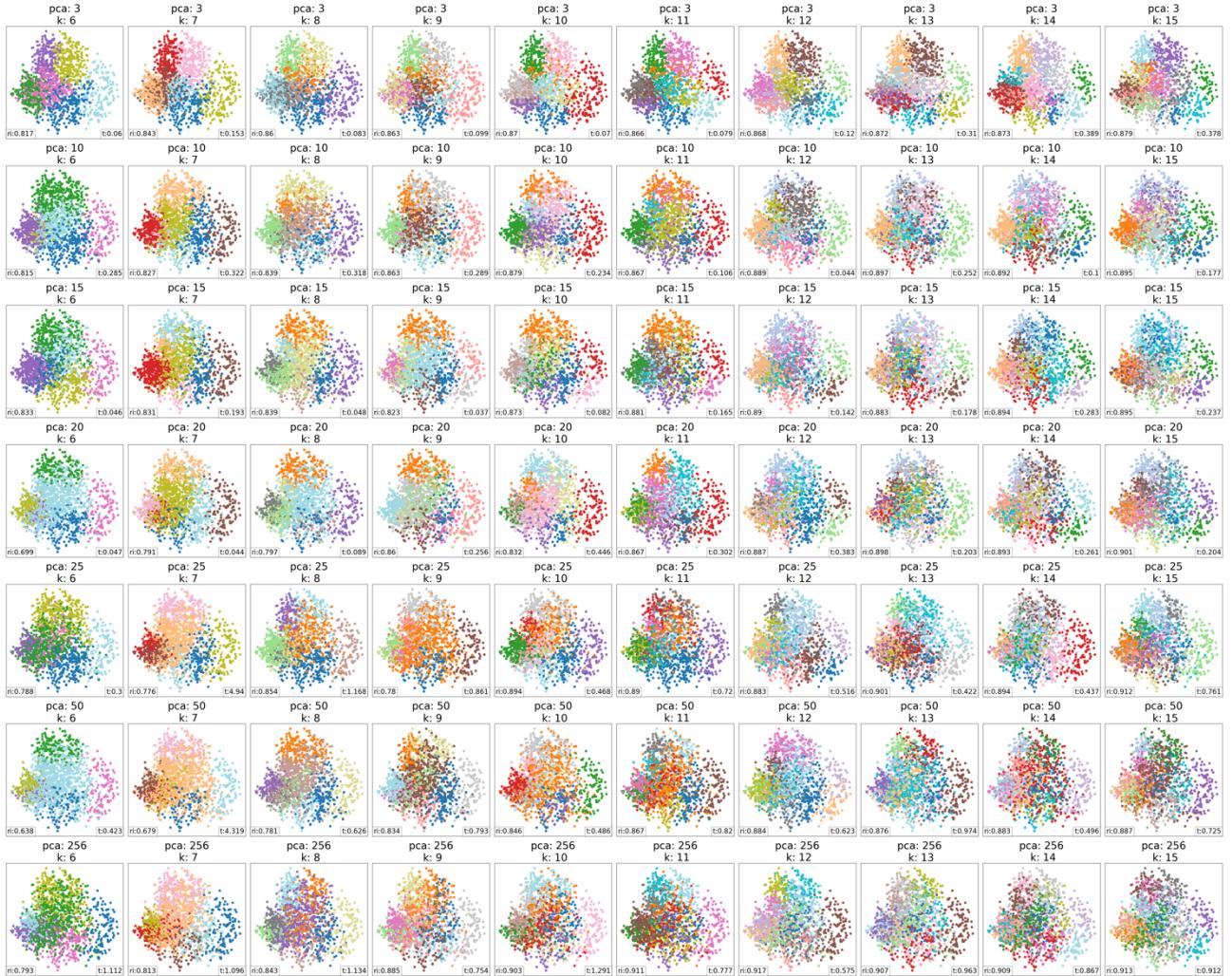


Figura 2: settanta clustering per diversi valori di PCA-componenti e di centri di cluster.

Come supposto prima si nota che diminuendo il numero di componenti tramite PCA, il tempo di training dei dati diminuisce.

Il Rand index invece, non sembra essere legato al numero di componenti principali utilizzate, anzi, in alcuni casi, ad esempio il caso in angolo in basso a destra, con tutte le componenti e suddiviso in 15 cluster, risulta essere dell'ordine del 0.913, piuttosto alto, ma con un tempo di calcolo prossimo al secondo. Risultati simili in termini di rand index si possono ottenere settando le componenti principali a venti e sempre con 15 cluster si nota un rand index di 0.901, con un tempo di calcolo pari a 0.2s, nettamente più veloce.

Inoltre, grazie all'utilizzo dei colori, possiamo vedere che non sempre la divisione dei dati viene effettuata in maniera chiara e definita.

Per i valori di cluster minori di dieci, ci aspettiamo che molti valori non siano catalogati come avrebbero dovuto.

Ad occuparsi della visualizzazione della media dei cluster nelle immagini seguenti, è il file *visualizeGaussianMean.py*.

Vediamo un esempio plotando la media delle immagini racchiuse in un cluster:

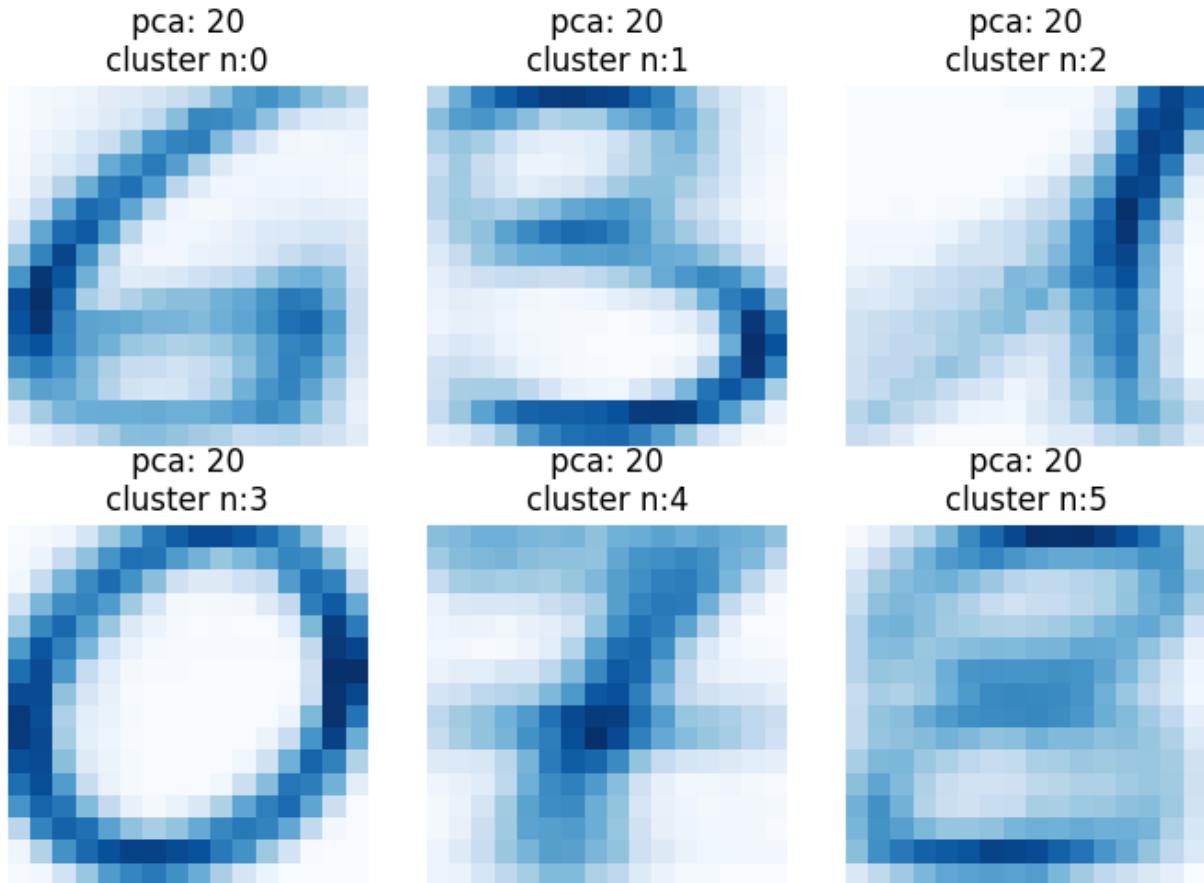


Figura 3: media per ogni cluster dell'immagine catalogata in questo.

Per alcuni numeri, ad esempio lo zero, possiamo dire che il modello ha effettuato un buon lavoro, mentre per altri, ad esempio il cinque, non vi è alcun cluster dedicato, il rand index corrispondente, infatti, è di 0.70, non bassissimo, ed infatti vediamo dei numeri ben suddivisi, ma nemmeno alto, il sesto cluster difatti non rappresenta un numero in particolare.

Proviamo a vedere come si comporta con dei parametri adeguati:

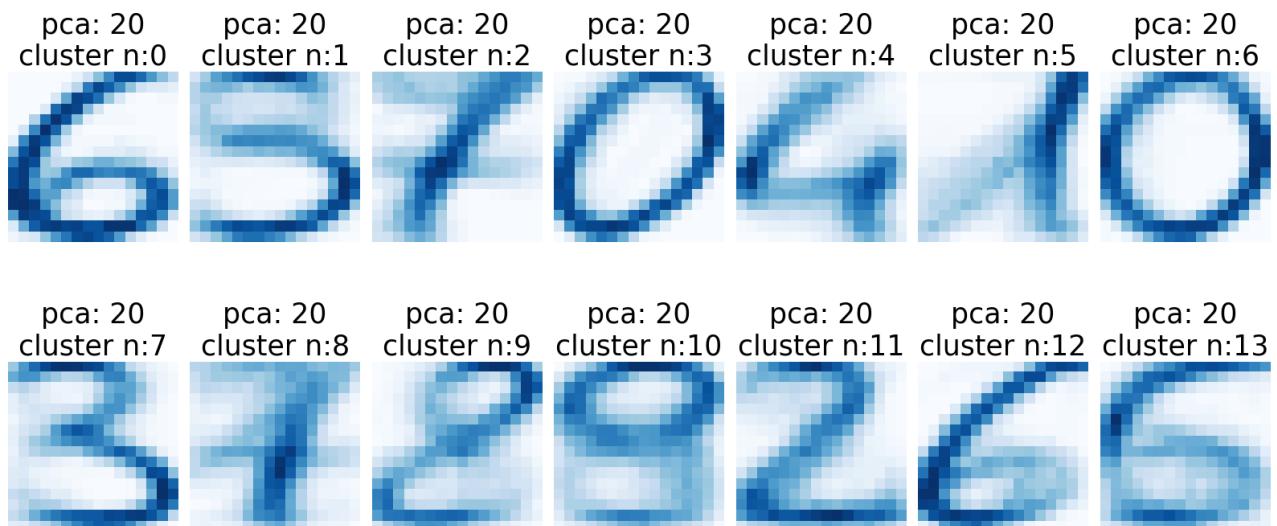


Figura 4: clustering PCA con 20 componenti e 14 cluster.

Il rand index in questo caso è pari a 0.893 il tempo di calcolo è di circa 0.260, si riescono difatti a vedere senza troppe difficoltà dei cluster dedicati ad ogni cifra, restano comunque dei leggeri dubbi per il numero cinque e il numero nove, che risentono della presenza rispettiva del numero nove e del numero otto. Come è logico che sia invece, vediamo dei cluster diversi che rappresentano lo stesso numero.

Nella seguente immagine vedremo uno dei casi migliori di clustering:

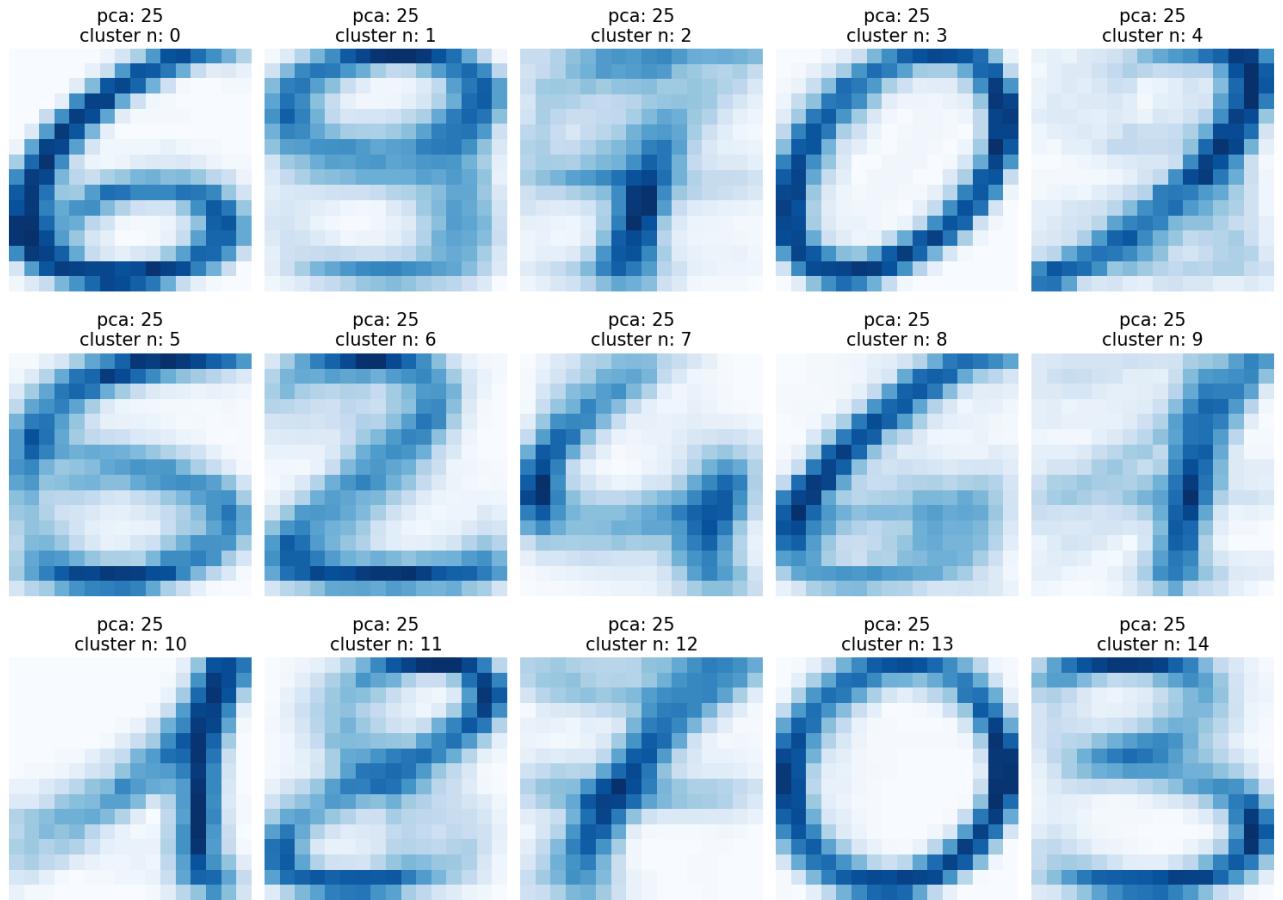


Figura 5: clustering PCA con 10 componenti e 15 cluster.

Il rand index in questo caso è piuttosto alto: 0.902 con un tempo di calcolo molto buono per queste performance: 0.7s. In particolare, in questo caso si riescono ad individuare, senza difficoltà, cluster dedicati ad ogni cifra

Performance poco migliori si possono ottenere a patto di accettare un tempo di calcolo che può, in alcuni casi, farsi prossimo al secondo.

Mean Shift Model

Mean Shift Clustering è un algoritmo non supervisionato che permette di raggruppare insiemi di dati in base alle loro caratteristiche. Il nome si può intendere come spostamento alla media in modo iterativo, ad ogni iterazione ogni punto viene spostato verso la regione più consona e la posizione finale sarà il cluster di appartenenza, è come se i punti shiftassero verso il loro simile.

Con il termine *media* intendiamo la media pesata dei dati con kernel uniforme, dove i pesi dei dati sono dati in funzione della distanza euclidea tra i punti, definita d come la distanza del punto interessato da tutti gli altri, e R come il raggio del cerchio di interesse attorno al punto:

$$M_w = \frac{\sum w_i x_i}{\sum x_i} \quad w_i = \begin{cases} 1 & \text{se } d \leq R \\ 0 & \text{se } d > R \end{cases}$$

Il parametro che andremo a testare per l'utilizzo di questo metodo, oltre al numero di componenti principale tramite PCA, è proprio R , anche detto *bandwidth*.

Ad occuparsi della generazione della figura seguente è il file *MeanShiftModel.py*

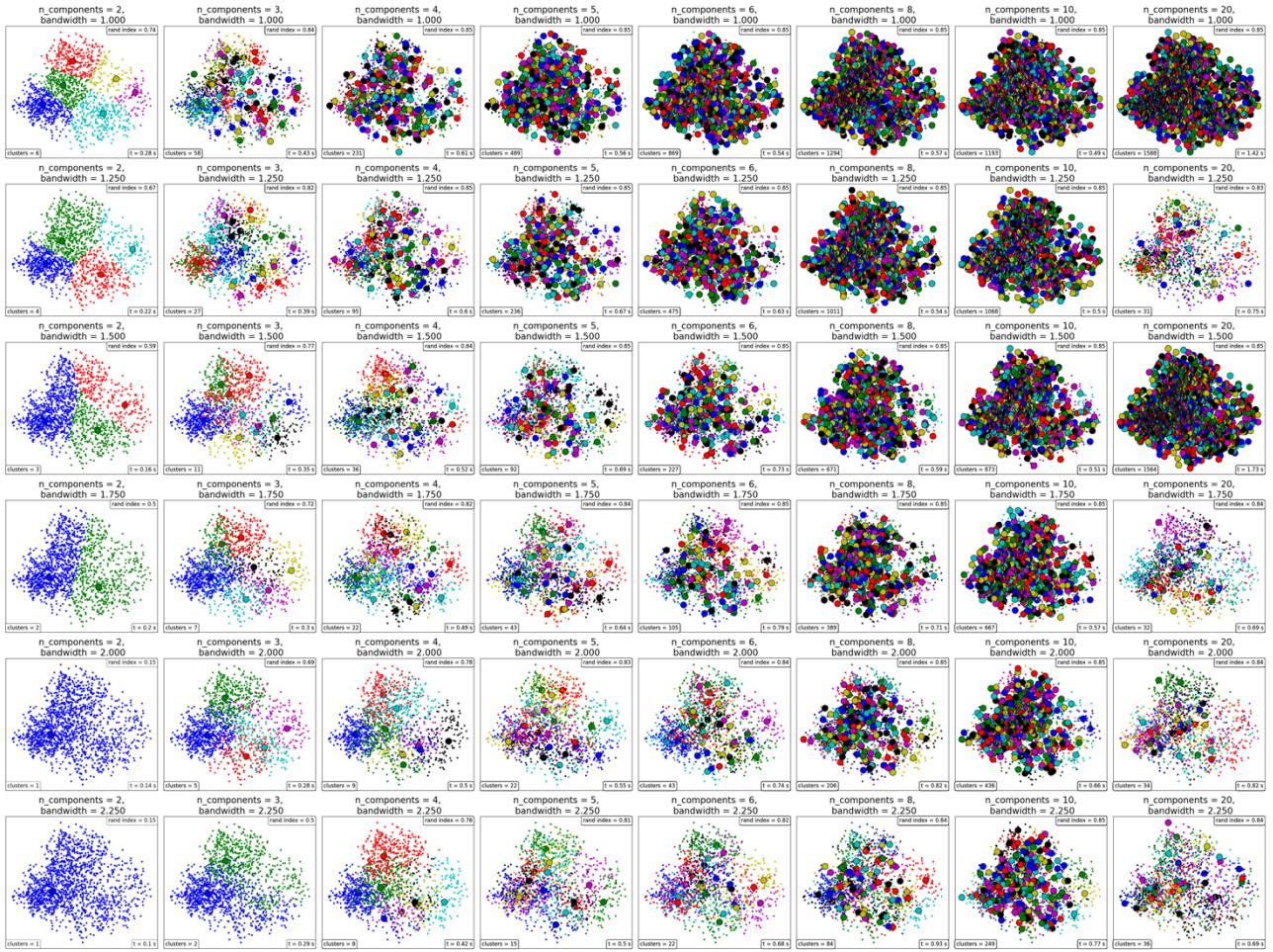


Figura 6: quarantotto modelli di Mean shift con vari parametri di PCA e Bandwidth.

Dalla figura 5, si nota chiaramente che all'aumentare delle componenti principali, il numero di cluster che il modello individua, aumenta, e come nel caso del modello precedente, siamo interessati a individuare un modello che restituisca un numero di cluster che sia almeno in un intorno di dieci, come le cifre. Questa è una criticità del metodo, in quanto se i parametri usati non sono adeguati, il modello tende a over-fittare i dati.

L'*over fitting* è il fenomeno per cui un modello viene sottoposto a dei dati, memorizzando molto bene anche il rumore presente in questi, il che rende molto poco agile l'algoritmo a rappresentare i dati in maniera consona alla realtà.

Il rand index per i vari modelli che sono stati costruiti varia da 0.7 a 0.85, escludendo i dati che ovviamente non riescono a rappresentare la realtà dei dati, individuando ad esempio, solamente due cluster. Una scelta consona potrebbe essere quello con cinque componenti principali e un bandwidth pari a 2.25, che restituisce quindici cluster con un rand index pari a 0.81, ed il tutto con un tempo di calcolo sull'ordine del mezzo secondo.

Spectral Clustering – Normalized Cut

Spectral Clustering fa parte di una serie di algoritmi che si presta a classificare nodi su un grafo pesato. Un grafo pesato rappresenta una matrice di similarità fra gli oggetti associati ai nodi del grafo.

Un valore alto associato al peso dell'arco che connette due nodi, convince l'algoritmo di clustering a collegare i due nodi ed è determinato da una grande somiglianza fra gli stessi. Un taglio (Cut) fa la divisione del grafo in sottografi, ovvero quando un gruppo di dati simili tra loro vengono divisi da un altro gruppo a loro volta simili ma diversi dai primi.

Il grado di dissimilarità tra due gruppi può essere calcolato come somma totale dei pesi dei nodi che sono stati rimossi:

$$Cut(C_1, C_2) = \sum_{i \in C_1} \sum_{j \in C_2} w_{ij}$$

Dove, data $d(i, j)$ la distanza tra i pixel i, j :

$$w_{ij} = e^{\frac{-d(i,j)^2}{\sigma_m^2}}$$

La migliore bipartizione del grafo è quella che minimizza il valore calcolato con la formula scritta qui sopra. Si cerca quindi di dividere il grafo iniziale in k -sottografi, in modo da minimizzare il taglio massimo dei sottogruppi, questo viene effettuato trovando in modo ricorsivo i tagli che dividono in due i gruppi esistenti.

Si è notato però che questo processo, portava ad una serie di problematiche per quanto riguarda il taglio di piccole partizioni di dati. La soluzione proposta da Jianbo Shi & Jitendra Malik, è quella del Normalized Cut, ovvero al posto di calcolare il peso totale della connessione di due partizioni, si calcola il costo di taglio come frazione delle connessioni totali del bordo con tutti i nodi del grafo:

$$Ncut(A, B) = \frac{cut(A, A)}{assoc(A, V)} + \frac{cut(B, B)}{assoc(B, V)}$$

Dove $assoc(A, V)$ sono le connessioni totali tra i nodi nella partizione A e tutto il resto dei nodi sul grafo. Il Normalized Cut si basa sul trovare il minimo di $Ncut(A, B)$. Nello spazio delle soluzioni continuo Normalized Cut, si propone di trovare gli autovettori con gli autovalori più piccoli possibili.

L'algoritmo si può quindi riassumere nei seguenti step:

- Presi dei dati, si crea un grafo pesato $G = (V, E)$, e riportare le informazioni in W e D .
- Risolvere $(D - W)x = \lambda Dx$ per gli autovettori con autovalori più piccoli.
- Usare gli autovettori per trovare il punto di divisione del grafo in modo che $Ncut$ sia minimizzato.
- Decidere se la partizione trovata debba essere effettivamente suddivisa controllando che $Ncut$ sia sotto ad uno specifico valore.
- Ripetere il processo in modo ricorsivo fino al raggiungimento del numero di cluster desiderato.

La classe presente su *sklearn: SpectralClustering*, si presta ad effettuare l'algoritmo *Normalized Cut* per dei dati sotto forma, non più di grafi, ma di matrice.

Oltre a testare l'algoritmo per i vari componenti di PCA e di cluster, testiamo tre valori per il parametro *assign_label*: *kmeans*, *dicreteize* e *cluster_gr*.

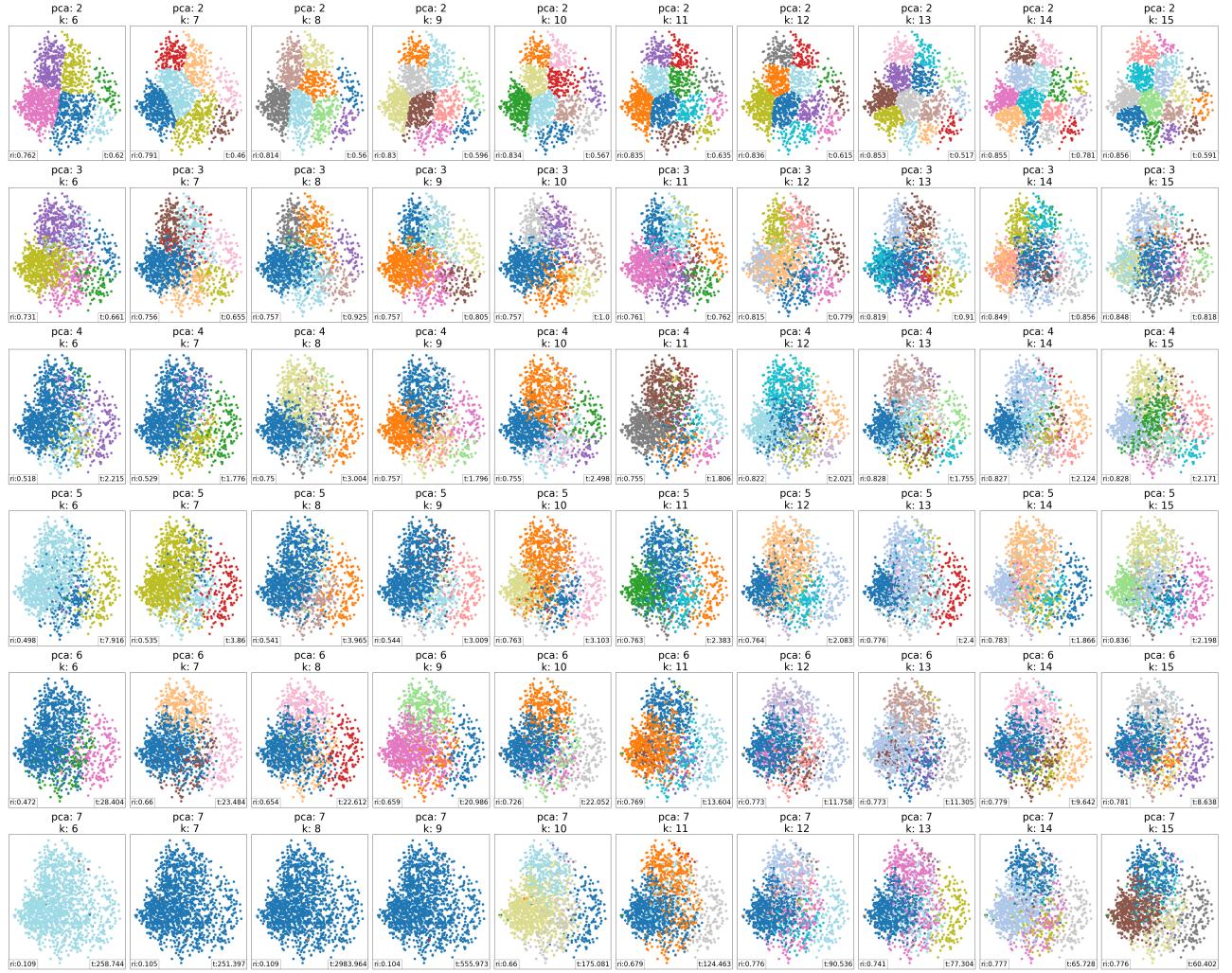


Figura 7: settanta clustering per valori differenti di pca e numero di cluster assign label = ‘discretize’.

Dalla figura 7, si nota che il tempo di calcolo aumenta all'aumentare del numero di PCA, già con sei componenti principali, il tempo di calcolo supera in media i dieci secondi, aumentando a sette si arriva tranquillamente sopra i due minuti, e per più componenti, diventa un tempo troppo grande per poter essere svolto da un computer normale da ufficio.

Con 2 e 3 componenti principali però si ottengono delle ottime performance con dei tempi di calcolo accettabili, si può osservare infatti, il caso di 2 componenti principali suddiviso in 13 cluster, il cui rand index è pari a 0.853 con un tempo di calcolo di mezzo secondo, o con 3 componenti principali e suddiviso in 14 cluster il rand index risulta pari a 0.849 con tempo di calcolo 0.856s.

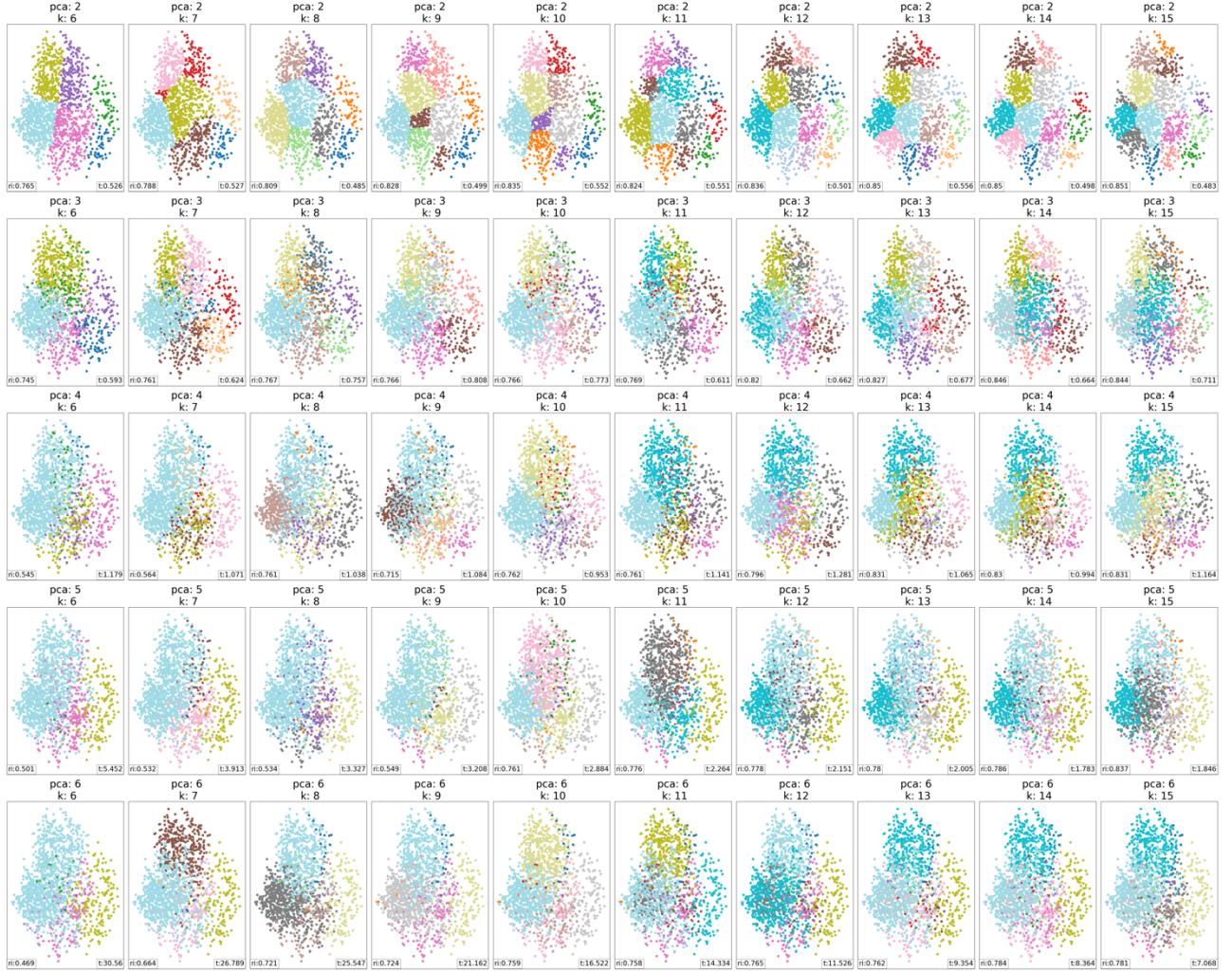


Figura 8: settanta clustering per valori differenti di pca e numero di cluster assign label = 'cluster_qr'

Dalla figura 8 si nota che ci sono delle configurazioni dei parametri che portano a valori di rand index interessanti, in particolare utilizzando un numero di componenti principali basso. Usando 2 componenti principali, e con un numero di cluster utilizzati alto, come tredici, quattordici o quindici, si ottiene un valore di rand index di circa 0.85, con tempi di calcolo nell'intorno del mezzo secondo.

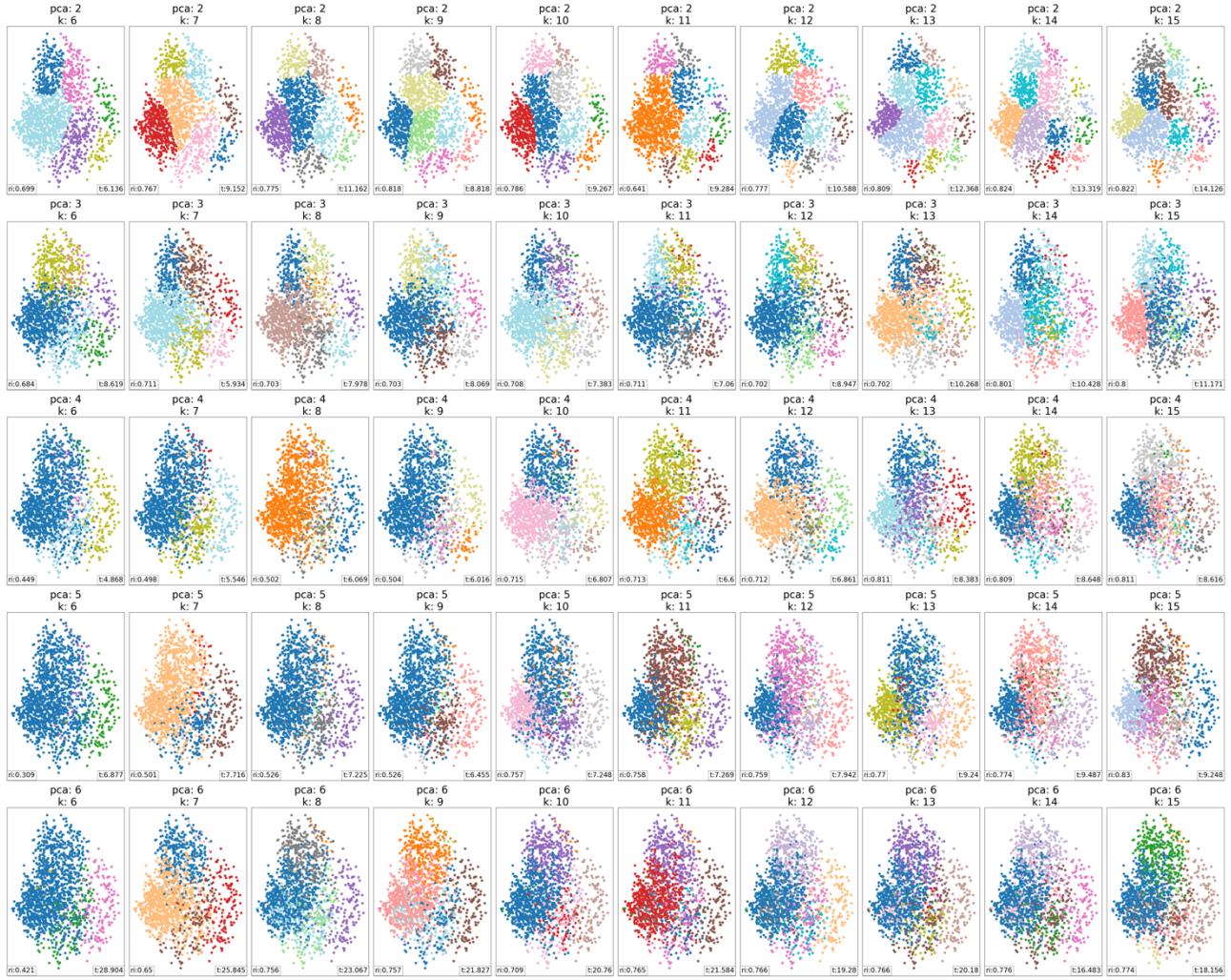


Figura 9: settanta clustering per valori differenti di pca e numero di cluster assign label = 'kmeans'

Dalla figura 9, si nota come in tutti i settaggi, nonostante vi siano dei valori di rand index accettabili, ad eccezione dei casi con un numero di cluster nettamente basso, i tempi di calcolo siano nettamente superiori ai casi precedenti.

Conclusioni

Il clustering tramite GaussianMixture necessita per una buona categorizzazione dei dati, più centri di clustering rispetto a quelli che nella realtà servirebbero, ma, nonostante ciò, possiamo dire sia un buon metodo di categorizzazione.

Il Mean Shift è particolarmente sensibile all'over fitting dei dati, ma con i parametri giusti, e con il giusto numero di componenti principali, può risultare un buon classificatore.

Inoltre, da questo classificatore, si nota una particolare relazione tra numero di cluster e RandIndex calcolato, ovvero all'aumentare del numero di cluster aumenta anche il rand index, questo è dovuto al fatto che viene creato un cluster quasi per ogni digit presente nel dataset. Di conseguenza il RandIndex preso senza guardare il contesto può non essere significativo per l'analisi di una clusterizzazione.

Il Normalized Cut è molto sensibile al numero di componenti principali che vengono utilizzati, ma con un buon settaggio dei parametri, si può ottenere un ottimo classificatore.

In particolare, con il parametro *assign_label* settato su *discretize* e *cluster_qr* si possono ottenere degli ottimi risultati, con dei tempi di calcolo accettabili, cosa che non succede per il parametro *kmeans*.