Soft Clustering e Rough K-Means

Progetto di Multivariate Analysis and Statistical Learning

Università degli Studi di Firenze
Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Magistrale in Informatica - Data Science

Elia Mercatanti - 6425149

Gennaio 2021

1 Cluster Analysis e Soft Clustering

Il clustering è uno degli strumenti più utilizzati nell'analisi dei dati. Negli ultimi decenni, a causa dell'aumento della complessità dei dati, abbiamo assistito ad un crescente interesse verso tali tecniche, in particolare il *soft clustering* ha ricevuto molta attenzione.

La *Cluster Analysis* mira a determinare un piccolo numero k di gruppi omogenei (clusters) da un insieme di n oggetti secondo un data misura della dissomiglianza basata su p variabili osservate $X_1, \ldots, X_p$, in altre parole, la *Cluster Analysis* ha il compito di suddividere un set di dati in gruppi (*clusters*) in modo significativo ed utile.

In molte applicazioni pratiche esistono oggetti che hanno caratteristiche intermedie tra i cluster, quindi spesso non possono essere chiaramente assegnati. In questi casi, l'approccio classico (*hard*) al raggruppamento porta ad un'assegnazione non realistica e gli oggetti sono costretti ad appartenere a un solo cluster. L'approccio *soft* nasce proprio per superare questo inconveniente. L'idea principale è che ogni dato può appartenere a più di un cluster. Esistono almeno quattro tipi di approcci di clustering *soft*: *fuzzy*, possibilistico, basato su modelli e *rough*.

Il clustering *fuzzy* consiste nell'assegnare dati ai cluster secondo un certo grado, chiamato *membership degree* e compreso tra 0 (completa non appartenenza) a 1 (completa appartenenza). L'utilizzo dei *membership degrees* è molto importante dal punto di vista computazionale, anche se in alcune situazioni non è necessario dal punto di vista teorico.

L'approccio possibilistico consiste nell'allentare i vincoli somma-unità dei *membership degrees*, aggiungendo un termine di penalizzazione. Questo porta a gradi di appartenenza che possono essere interpretati come gradi di compatibilità degli oggetti con i cluster e gli oggetti lontani da tutti i prototipi vengono assegnati ai cluster con gradi di appartenenza inferiori.

Il terzo approccio si basa sulla nozione di *rough set*. È possibile definire ciascuno *rough set* utilizzando la sua *lower approximation* (LA) e *upper approximation* (UA). Sulla base delle distanze tra ogni oggetto e ogni prototipo, si può stabilire se un dato appartiene a una *lower approximation* o a due o più *upper approximations* o regioni di confine (*boundary regions*). Il più conosciuto di questi algoritmi è il *Rough K-Means* (RkM).

Infine la maggior parte dei ricercatori pensa che solo i tre approcci sopra descritti possano essere considerati *soft*. Da altri punti di vista invece anche quello *model-based* può anche essere considerato come un *soft clustering*. I

metodi di clustering *model-based* producono anch'essi una partizione flessibile degli oggetti e la probabilità a posteriori di un componente/cluster gioca un ruolo simile a quello della *membership degrees*. Partendo dal modello più utilizzato, la miscela di densità Gaussiane, sono state proposte estensioni che coinvolgono differenti distribuzioni parametriche.

Questi quattro approcci verranno descritti ed analizzati da un punto di vista teorico e da un'analisi comparativa empirica, in particolare ci concentreremo sull'approccio di tipo *rough* fornendo un'implementazione in R di una sua versione basata sull'algoritmo di clustering *K-Means*.

2 Fuzzy Clustering

Nel clustering *fuzzy* i dati vengono assegnati ai cluster in base ad un grado di appartenenza. Un grado vicino a 1 implica che l'oggetto è simile al prototipo di quel cluster, mentre un valore vicino a 0 indica la completa non appartenenza dell'oggetto al cluster, quindi è lontano dal prototipo corrispondente. Partendo dall'estensione del classico algoritmo *K-Means* (kM) al caso *fuzzy*, l'FkM, sono state introdotte un gran numero di varianti di algoritmi di clustering fuzzy.

2.1 Fuzzy K-Means

L'algoritmo di clustering *fuzzy* più conosciuto e utilizzato è l'FkM. Ha lo scopo di determinare una partizione *fuzzy* di n oggetti in k cluster risolvendo il seguente problema di minimizzazione:

$$\min_{\mathbf{U},\mathbf{H}} J_{FkM} = \sum_{i=1}^{n}\sum_{g=1}^{k} u_{ig}^{m} d^2(\mathbf{x}_i, \mathbf{h}_g), \tag{1}$$

$$\text{s.t.} \quad u_{ig} \in [0,1], \quad \sum_{g=1}^{k} u_{ig} = 1, \quad i = 1,\ldots,n, \quad g = 1,\ldots,k, \tag{2}$$

dove $d(\cdot,\cdot)$ è la distanza euclidea. $\mathbf{x}_i = [x_{i1},\ldots,x_{ip}]$ $(i = 1,\ldots,n)$ è la riga generica della matrice dei dati $\mathbf{X}$ di dimensione $(n \times p)$. In (1) $\mathbf{U}$ è la matrice dei gradi di appartenenza di dimensione $(n \times k)$, il cui elemento generico u_{ig} rappresenta il grado di appartenenza dell'oggetto i al cluster g.

Le somme di riga di $\mathbf{U}$ sono uguali a 1. $\mathbf{H}$ è la matrice dei prototipi (centroidi) di dimensione $(k \times p)$, la cui riga generica è del tipo $\mathbf{h}_g = [h_{g1}, \ldots, h_{gp}]$ $(g = 1, \ldots, k)$. Infine, $m > 1$ è il parametro di *fuzziness*, di solito fissato tra 1.5 e 2.

La soluzione di (1) è ottenuta mediante un algoritmo iterativo. In pratica, viene adottato il metodo del moltiplicatore Lagrangiano (λ), impostando le derivate parziali della funzione Lagrangiana (L) rispetto a u_{ig} e λ uguale a 0. Si noti che solo l'ultimo vincolo in (2) viene utilizzato, poiché il primo è soddisfatto automaticamente. Quindi, fissando u_{ig}, otteniamo $\mathbf{h}_g$ ponendo uguale a 0 le derivate parziali di L rispetto a $\mathbf{h}_g$. La soluzione iterativa è la seguente:

$$u_{ig} = \frac{1}{\sum_{g'=1}^{k} \left(\frac{d^2(\mathbf{x}_i, \mathbf{h}_g)}{d^2(\mathbf{x}_i, \mathbf{h}_{g'})} \right)^{\frac{1}{m-1}}} \qquad \mathbf{h}_g = \frac{\sum_{i=1}^{n} u_{ig}^m \mathbf{x}_i}{\sum_{i=1}^{n} u_{ig}^m}. \tag{3}$$

Come possiamo vedere, i gradi di appartenenza degli oggetti ai cluster sono tali da essere inversamente correlati alle relative dissomiglianze tra gli oggetti e i centroidi. Per questo motivo, i gradi di appartenenza possono essere interpretati come gradi di condivisione (delle osservazioni ai cluster).

Per quanto riguarda i classici metodi basati sul *K-Means*, le prestazioni di tutti gli algoritmi *fuzzy* sono influenzate dai valori anomali (*outliers*). Il problema è dovuto ai vincoli di somma unitaria delle *membership degrees*. Ciò implica che anche gli oggetti anomali siano assegnati ai cluster e che i centroidi ne siano influenzati. Un timido tentativo di robustificazione è stato introdotto dall'algoritmo *Fuzzy K-Medoids*, una generalizzazione del classico *K-Medoids*. Un'altra proposta per rafforzare l'algoritmo è quella che introduce il cosiddetto *noise cluster* che rappresenta una partizione aggiuntiva (non un vero e proprio cluster) contenente tutti gli oggetti considerati *outlier*, in questo caso il prototipo degli *outliers* è un'entità universale tale che è sempre alla stessa distanza da ogni punto del set di dati.

3 Possibilistic Clustering

Come già discusso nella sezione precedente, l'FkM e le sue estensioni possono portare a risultati inaspettati in presenza di possibili valori anomali. Per ovviare a questo inconveniente, invece di utilizzare il FkMed o gli algoritmi di clustering *fuzzy* con un *noise cluster*, può essere utilizzato l'approccio pos-

sibilistico. Esso consiste nel rilassare i vincoli della somma unitaria sui gradi di appartenenza (*membership degrees*), aggiungendo un termine di penalizzazione. Ciò porta a gradi di appartenenza che possono essere interpretati come gradi di compatibilità degli oggetti con i cluster e dove gli oggetti lontani da tutti i prototipi sono assegnati ai cluster con gradi di appartenenza inferiori (vicini a 0).

3.1 Possibilistic K-Means

L'algoritmo di clustering possibilistico più noto è il metodo di clustering PkM, che può essere formalizzato come di seguito:

$$\min_{\mathbf{T},\mathbf{H}} J_{PkM} = \sum_{i=1}^{n}\sum_{g=1}^{k} t_{ig}^{\eta} d^2(\mathbf{x}_i, \mathbf{h}_g) + \sum_{g=1}^{k}\gamma_g \sum_{i=1}^{n}(1-t_{ig})^{\eta}, \tag{4}$$

$$\text{s.t. } t_{ig} \in [0,1], \quad i = 1,\ldots,n, \quad g = 1,\ldots,k, \tag{5}$$

dove t_{ig} è il grado di appartenenza dell'osservazione i al cluster g, memorizzato nella matrice $\mathbf{T}$ di dimensioni $(n \times k)$. I gradi di appartenenza sono valori di possibilità e sono generalmente indicati come gradi di tipicità (delle osservazioni rispetto ai cluster). Essi sono la variante possibilistica dei gradi di appartenenza nel FkM e servono per riconoscere la partizione *soft*. Il parametro γ_g è *cluster-specific* e regola l'importanza dei cluster mentre η (> 1) è il *fuzzifier*. Il parametro γ_g può essere definito come:

$$\gamma_g = \gamma \frac{\sum_{i=1}^{n} u_{ig}^m d^2(\mathbf{x}_i, \mathbf{h}_g)}{\sum_{i=1}^{n} u_{ig}^m}, \quad g = 1,\ldots,k, \tag{6}$$

dove, solitamente, $\gamma = 1$ e gli $\mathbf{h}_g$ e i vari u_{ig} sono ottenuti dal FkM. L'equazione (6) può essere motivata osservando che i γ_g rappresentano il peso relativo del secondo termine della funzione obbiettivo rispetto al primo. Il secondo termine evita inoltre la soluzione banale con $\mathbf{T} = 0$.

La soluzione di PkM, derivata utilizzando un algoritmo iterativo, è la seguente:

$$t_{ig} = \frac{1}{1 + \left(\frac{d^2(\mathbf{x}_i,\mathbf{h}_g)}{\gamma_g}\right)^{\frac{1}{\eta-1}}} \qquad \mathbf{h}_g = \frac{\sum_{i=1}^{n} t_{ig}^{\eta}\mathbf{x}_i}{\sum_{i=1}^{n} t_{ig}^{\eta}}. \tag{7}$$

Nel PkM, i gradi di tipicità, t_{ig}, sono inversamente correlati alle differenze tra le osservazioni e i centroidi. Infatti, una proprietà comune dei metodi di clustering basati sull'approccio possibilistico è che i gradi di appartenenza vengono calcolati considerando solo la dissomiglianza tra l'osservazione e il centroide più vicino, indipendentemente dai centroidi dei restanti cluster. Questo spiega come tale approccio gestisce i dati contaminati, i valori anomali sono ben lungi dall'essere il grosso dei dati e, quindi, risultano lontani da tutti i centroidi. Pertanto, di solito hanno gradi di tipicità vicini allo 0 per tutti i cluster.

Purtroppo il PkM può soffrire del rischio di cluster coincidenti. Questa limitazione è dovuta alla somma dei gradi di tipicità per ogni osservazione su tutti i cluster che non è più forzata ad essere uguale a uno. Inoltre, possiamo notare che la funzione di ottimizzazione in (4) può essere scomposta nella somma di k termini che possono essere minimizzati in modo indipendente uno d'altro. Un rimedio euristico al problema dei cluster coincidenti è l'uso di un punto di partenza razionale, ovvero dei centroidi iniziale che non siano scelti casualmente ma che abbiano un senso. Ad esempio, l'algoritmo iterativo di PkM può essere eseguito a partire dalla soluzione del FkM. Un ulteriore approccio da superare il problema del raggruppamento coincidente consiste nell'aggiungere un termine di repulsione tra i centroidi nella funzione obiettivo. Questo termine cerca di costringe i centroidi a essere lontani l'uno dall'altro. Inoltre, sono stati proposti anche algoritmi di clustering possibilistici basati sulla funzione obiettivo del FkM.

4 Model-Based Clustering

Nell'approccio *model-based*, i metodi di clustering presuppongono che i dati siano generati da un modello statistico casuale e provano dunque a recuperarlo dai dati. In questo contesto, i dati seguono una mistura di distribuzioni. Dato $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}_p$, dove si presume che il vettore casuale $\mathbf{x}_i$ derivi da una mistura finita di funzioni di densità di probabilità:

$$f(\mathbf{x}_i;\ \mathbf{\Phi}) = \sum_{g=1}^{k} \pi_g f(\mathbf{x}_i|\theta_g), \tag{8}$$

dove π_g, $g = 1, \ldots, k$, tale che $\pi_g > 0$ e $\sum_{g=1}^{k} \pi_g = 1$, sono le proporzioni di mistura, k è il numero di componenti, $f(\mathbf{x}_i|\theta_g)$ è la densità della

componente ($g = 1, \ldots, k$) e $\mathbf{\Phi} = (\pi_1, \ldots, \pi_k, \theta_1, \ldots, \theta_k)$ è il vettore dei parametri. Ogni densità dei componenti di mistura appartiene a una specifica classe parametrica e rappresenta un cluster. Anche se non è necessario che ogni densità dei componenti della mistura derivi dalla stessa famiglia di distribuzione parametrica, ci concentreremo solo sul caso dove la famiglia di distribuzione parametrica è la stessa per ogni componente della miscela. Questo è il caso più comune esaminato nella letteratura. La stima dei parametri viene eseguita con l'approccio della massima verosimiglianza. In dettaglio, la stima dei parametri del modello in $\mathbf{\Phi}$ viene eseguita applicando l'algoritmo *Expectation–Maximization* (EM). Una volta che l'algoritmo EM raggiunge la convergenza, la partizione *soft* dei dati può essere ottenuta mediante le probabilità a posteriori, utilizzando la regola del massimo a posteriori, cioè assegnando ogni dato al cluster con la più alta probabilità a posteriori. Quest'ultime, come i gradi di appartenenza dell'algoritmo FkM e i gradi di tipicità nel PkM, vengono utilizzati per determinare la *softness* della partizione.

4.1 Mistura Finita di Densità Gaussiane

Il modello più utilizzato per il clustering è la mistura di distribuzione Gaussiane, ovvero $f(\mathbf{x}_i|z_{ig} = 1, \theta_g) \sim N(\mu_g, \mathbf{\Sigma}_g)$. La miscela finita di densità gaussiane (FMG) è quindi data da:

$$f(\mathbf{x}_i;\ \mathbf{\Phi}) = \sum_{g=1}^{k} \pi_g \phi(\mathbf{x}_i|\mu_g, \mathbf{\Sigma}_g), \tag{9}$$

dove $\mathbf{\Phi} = \{\pi_1, \ldots, \pi_{g-1}, \mu_1, \ldots \mu_g, \mathbf{\Sigma}_1, \ldots, \mathbf{\Sigma}_g\}$ denota il set di parametri per il modello di mistura finita e $\phi(\mathbf{x}_i|\mu_g, \mathbf{\Sigma}_g)$ la sottostante funzione di densità specifica del componente con parametri μ_g, $\mathbf{\Sigma}_g$, $g = 1, \ldots, k$.

Pertanto, i cluster ellissoidali centrati sul vettore medio μ_g sono generati dal modello in (9). Il parametro $\mathbf{\Sigma}_g$ controlla le altre proprietà geometriche di ogni cluster. Le parsimoniose parametrizzazioni delle matrici di covarianza dei cluster possono essere ottenute attraverso la decomposizione $\mathbf{\Sigma}_g = \lambda_g \mathbf{D}_g \mathbf{A}_g \mathbf{D}_g^T$, dove il volume, la forma e l'orientamento sono controllati, rispettivamente, tramite lo scalare λ_g, la matrice diagonale $\mathbf{A}_g$, e la matrice ortogonale $\mathbf{D}_g$.

Le misture finite di densità gaussiane sono sensibili ai dati anomali. Per ottenere un modello robusto possono essere adottate una mistura di distri-

buzioni t. Un approccio alternativo per gestire i valori anomali si basa sul *trimming*. A differenza degli approcci menzionati in precedenza in cui l'obiettivo è quello di adattare i valori anomali, in quello di *trimming*, i valori anomali vengono scartati.

5 Rough Clustering

I metodi di clustering *rough* sono basati sulla nozione di *rough set*. Un *rough set* C_g è definito mediante la sua *lower approximation* (LA) e la sua *upper approximation* (UA), ovvero, $\underline{\mathrm{A}}(C_g)$ e $\overline{A}(C_g)$. La famiglia di LA e UA di C_g devono soddisfare le seguenti proprietà:

(**P1**) Un oggetto $\mathbf{x}_i$ può far parte al massimo di una LA.

(**P2**) $\mathbf{x}_i \in \underline{\mathrm{A}}(C_g) \Rightarrow \mathbf{x}_i \in \overline{A}(C_g)$.

(**P3**) $\mathbf{x}_i$ non fa parte di alcuna LA $\Leftrightarrow$ $\mathbf{x}_i$ appartiene a due o più UA.

Lo scopo del clustering *rough* è determinare se un oggetto appartiene alla UA o LA di un cluster. Per ogni vettore oggetto, $\mathbf{x}_i$, per stabilire l'assegnazione di $\mathbf{x}_i$, data una soglia ψ, vengono utilizzati i rapporti $d(\mathbf{x}_i, \mathbf{c}_g)/d(\mathbf{x}_i, \mathbf{c}_{g'})$ con $g = 1, \ldots, k$ e $g' = 1, \ldots, k$. Per ricavare le varie UA e LA sono utilizzate le seguenti regole:

1. Se $d(\mathbf{x}_i, \mathbf{h}_g)$ è il minimo per $1 \leq g \leq k$ e $d(\mathbf{x}_i, \mathbf{h}_g)/d(\mathbf{x}_i, \mathbf{h}_{g'}) \leq \psi$ per qualsiasi coppia (g, g') allora $\mathbf{x}_i \in \overline{A}(C_g)$ e $\mathbf{x}_i \in \overline{A}(C_{g'})$. Inoltre, $\mathbf{x}_i$ non fa parte di nessuna LA. La proprietà (P3) viene dunque soddisfatta .

2. Altrimenti, $\mathbf{x}_i \in \underline{\mathrm{A}}(C_g)$ tale che $d(\mathbf{x}_i, \mathbf{h}_g)$ è il minimo per $1 \leq g \leq k$. Inoltre, per la proprietà (P2), $\mathbf{x}_i \in \overline{A}(C_g)$. Segue che anche in questo caso la proprietà (P1) è soddisfatta.

5.1 Rough K-Means

Nel Rough K-Means (RkM), un cluster è descritto da due approssimazioni *hard*, una LA e una UA (o regione di confine). Quindi, un dato ha due gradi di appartenenza bivalenti al cluster g, uno per la sua LA e uno per la UA:

$$\mu_{ig}^{LA} \in \{0, 1\} \qquad \mu_{ig}^{UA} \in \{0, 1\}.$$

Gli oggetti nella LA appartengono sicuramente al cluster corrispondente, mentre gli oggetti nell'UA possono appartenere al cluster o meno, c'è incertezza su di essi. L'RkM può essere considerato come un metodo *soft* a causa della regione di confine per gestire l'incertezza nel processo di clustering. Nel clustering RkM, i dati vengono assegnati ai LA o ai UA utilizzando le regole [1] o [2] precedenti. Le varie LA e UA sono usate per definire la partizione *soft* dei dati, quindi, svolgono lo stesso ruolo di quelli degli u_{ig} in FkM e dei t_{ig} in PkM. I centroidi sono calcolati come segue. Se le cardinalità di $\underline{\mathrm{A}}(\mathbf{c}_g)$ e $\overline{A}(\mathbf{c}_g) - \underline{\mathrm{A}}(\mathbf{c}_g)$ (indicato con $|\underline{\mathrm{A}}(\mathbf{c}_g)|$ e $|\overline{A}(\mathbf{c}_g) - \underline{\mathrm{A}}(\mathbf{c}_g)|$, rispettivamente) non sono uguali a 0, avremo che:

$$\mathbf{h}_g = w_l \times \frac{\sum_{\mathbf{x}_i \in \underline{\mathrm{A}}(\mathbf{c}_g)} \mathbf{x}_i}{|\underline{\mathrm{A}}(\mathbf{c}_g)|} + w_u \times \frac{\sum_{\mathbf{x}_i \in \overline{A}(\mathbf{c}_g) - \underline{\mathrm{A}}(\mathbf{c}_g)} \mathbf{x}_i}{|\overline{A}(\mathbf{c}_g) - \underline{\mathrm{A}}(\mathbf{c}_g)|}, \tag{10}$$

con $w_l > 0$ e $w_u > 0$ tali che $w_l + w_u = 1$ (solitamente $w_l > w_u$). Se $|\underline{\mathrm{A}}(\mathbf{c}_g)| \neq 0$ e $|\overline{A}(\mathbf{c}_g) - \underline{\mathrm{A}}(\mathbf{c}_g)| = 0$ allora:

$$\mathbf{h}_g = \frac{\sum_{\mathbf{x}_i \in \underline{\mathrm{A}}(\mathbf{c}_g)} \mathbf{x}_i}{|\underline{\mathrm{A}}(\mathbf{c}_g)|} \tag{11}$$

Infine, se $|\underline{\mathrm{A}}(\mathbf{c}_g)| = 0$ e $|\overline{A}(\mathbf{c}_g) - \underline{\mathrm{A}}(\mathbf{c}_g)| \neq 0$ allora:

$$\mathbf{h}_g = \frac{\sum_{\mathbf{x}_i \in \overline{A}(\mathbf{c}_g) - \underline{\mathrm{A}}(\mathbf{c}_g)} \mathbf{x}_i}{|\overline{A}(\mathbf{c}_g) - \underline{\mathrm{A}}(\mathbf{c}_g)|}. \tag{12}$$

L'algoritmo proposto è stato poi leggermente perfezionato in rivisitazioni successive migliorandone la funzione obiettivo, la stabilità numerica e quella dei cluster. In letteratura viene proposto di combinare *fuzzy* e *rough set*, introducendo la LA e UA *fuzzy* per ogni cluster.

5.2 Implementazione del Rough K-Means in R

In questa sezione andremo a mostrare un'implementazione dell'algoritmo Rough K-Means nel linguaggio R che verrà poi utilizzata in seguito anche per mostrare alcuni risultati di sperimentazione e confronti. Di seguito viene riportato il codice principale dell'algoritmo.

Listing 1: Rough K-Means

```
1 # Rough K-Means implementation
2 rough_kmeans = function(dataset, means_matrix = 2, num_clusters = 2,
```

```
                        max_iterations = 100, threshold = 1.5,
                        weight_lower = 0.7) {

  # Setting of variables and matrices
  num_obs = nrow(dataset)
  num_features = ncol(dataset)
  threshold = threshold^2
  dataset = as.matrix(dataset)

  # Initialize Upper Approximation (UA) matrix and means
  old_upper_approx_matrix = matrix(999, nrow = num_obs, ncol = num_clusters)
  means_matrix = initialize_means(dataset, num_clusters, means_matrix)
  upper_approx_matrix = assign_upper_approx(dataset, means_matrix, threshold)

  # Initialize the iteration
  iterations = 0

  # Repeat until classification unchanged or max_iterations reached
  while ( !identical(old_upper_approx_matrix, upper_approx_matrix)
          && iterations < max_iterations ) {

    # Lower Approximation (LA) Matrix and UA-LA matrix (boundary)
    lower_approx_matrix = assign_lower_approx(upper_approx_matrix)
    boundary_matrix = upper_approx_matrix - lower_approx_matrix

    # Calculate sums of observations in LA and boundary in every cluster
    means_matrix_lower = crossprod(lower_approx_matrix, dataset)
    means_matrix_boundary = crossprod(boundary_matrix, dataset)

    # Update means matrix
    for (i in 1:num_clusters) {

      # Dividers means calculation, cardinalities of LA and UA-LA (boundary)
      divider_lower_approx = sum(lower_approx_matrix[, i])
      divider_boundary = sum(boundary_matrix[, i])

      if (divider_lower_approx != 0 && divider_boundary != 0) {
        means_matrix_lower[i,] = means_matrix_lower[i,] /
                                 divider_lower_approx
        means_matrix_boundary[i,] = means_matrix_boundary[i,] /
                                    divider_boundary
        means_matrix[i,] = weight_lower*means_matrix_lower[i,] +
                           (1-weight_lower)*means_matrix_boundary[i,]
      } else if (divider_boundary == 0) {
        means_matrix[i,] = means_matrix_lower[i,] / divider_lower_approx
      } else { # if(divider_lower_approx[,i]) == 0)
        means_matrix[i,] = means_matrix_boundary[i,] / divider_boundary
      }
    }

    # Saving upper approximations of previous iteration
    old_upper_approx_matrix = upper_approx_matrix
    upper_approx_matrix = assign_upper_approx(dataset, means_matrix,
                                              threshold)

    iterations = iterations + 1
  }
```

```
60
61    return ( list(upper_approx=upper_approx_matrix, cluster_means=means_matrix,
62                  num_iterations=iterations) )
63  }
```

L'algoritmo prende come input un dataset, una matrice dei centroidi iniziali oppure un valore per indicare come generarli all'interno della funzione, il numero di cluster da ricercare (*num_clusters*) il massimo numero di iterazione da eseguire, il valore della soglia ψ (default 1.5) e il valore dato al peso per i dati appartenenti alle *Lower Approximations* LA nel calcolo dei centroidi (*weight_lower*, default 0.7).

Per prima cosa l'algoritmo si occupa di andare ad inizializzare alcune variabili utili ai calcoli successivi come il numero di osservazioni presenti dentro il dataset (*num_obs*), il numero di *features* di esso (*num_features*), il contatore del numero di iterazioni (*iterations*), la matrice dei centroidi (*means_matrix*) e la matrice contenente le *Upper Approximations* UA dei vari cluster (*upper_approx_matrix*). La matrice dei centroidi può essere passata in input dall'utente oppure può essere generata all'interno dell'algoritmo, la funzione *initialize_means* mostrata in 2 fa proprio questo. In particolare, nel caso in cui non venga passata in input una matrice con già i centroidi l'utente può scegliere se generare i prototipi iniziali in modo casuale (inserendo il valore 1) oppure generandoli prendendo come centroidi gli oggetti del dataset che sono tra loro più distanti (inserendo il valore 2). Di default viene scelto la seconda strategia.

Listing 2: initialize_means

```
1  # Delivers an initial means matrix
2  initialize_means = function(data_matrix, num_clusters, means_matrix) {
3
4    if(is.matrix(means_matrix)) { # means pre-defined # no action required
5
6    }else if (means_matrix == 1) {  # random means
7
8      num_features = ncol(data_matrix)
9      means_matrix = matrix(0, nrow=num_clusters, ncol=num_features)
10     for (i in 1:num_features) {
11       means_matrix[,i] = c(runif(num_clusters, min(data_matrix[,i]),
12                                  max(data_matrix[,i])))
13     }
14
15   }else if (means_matrix == 2) { # maximum distance means
16
17     means_objects = seq(length=num_clusters, from=0, by=0)
18     objects_dist_matrix = as.matrix(dist(data_matrix))
19
20     pos_vector = which(objects_dist_matrix == max(objects_dist_matrix),
```

```
                        arr.ind = TRUE)
    means_objects[1] = pos_vector[1,1]
    means_objects[2] = pos_vector[1,2]

    for(i in seq(length=(num_clusters-2), from=3, by=1) ) {
      means_objects[i] = which.max(
        colSums(objects_dist_matrix[means_objects, -means_objects]))
    }

    means_matrix = data_matrix[means_objects, ]

  }

  return(as.matrix(means_matrix))
}
```

Per quanto riguarda l'inizializzazione della matrice degli UA, prima di tutto quest'ultima ha una dimensione pari a $num\_obs \times num\_clusters$, dove un elemento rappresenta la presenza (con uno 0 o 1) di una osservazione in una determinata UA di ogni cluster. Quello che viene fatto è sostanzialmente applicare la regola 1 e dunque vengono calcolate per ogni osservazione le distanze da ogni centroide e prendendo quella minima viene controllato se il rapporto $d(\mathbf{x}_i, \mathbf{h}_g)/d(\mathbf{x}_i, \mathbf{h}_{g'})$ scende sotto la soglia ψ, se questo è il caso allora tale osservazione viene associata alla UA del cluster g. Nello specifico questa inizializzazione viene eseguita dalla funzione *assign_upper_approx* riportata in 3. Tale funzione oltre ad ad essere utilizzata per inizializzare la matrice degli UA per la prima iterazione dell'algoritmo viene utilizzata anche per poi aggiornare la stessa matrice per le iterazioni successive.

Listing 3: assign_upper_approx

```
# Assign object to upper approximation
assign_upper_approx = function(dataset, means_matrix, threshold) {

  num_obs  = nrow(dataset)
  num_clusters = nrow(means_matrix)

  distances_to_clusters = seq(length=num_clusters, from=0, by=0)

  upper_approx_matrix = matrix(0, nrow = num_obs, ncol = num_clusters )

  for (i in 1:num_obs) {

    # distances_to_clusters from object i to all clusters j
    for (j in 1:num_clusters){
      distances_to_clusters[j] = sum( (dataset[i,] - means_matrix[j,] )^2 )
    }

    min_distance = max(min(distances_to_clusters), 1e-99)

    # Includes the closest objects.
```

```
    closest_clusters_idx = which((distances_to_clusters / min_distance)
                                 <= threshold)

    upper_approx_matrix[i, closest_clusters_idx] = 1

  }

  return(upper_approx_matrix)
}
```

Successivamente l'algoritmo entra nella suo ciclo principale dove continua ad iterare tutte le varie operazioni fino a che la matrice degli UA non cambia da una iterazione all'altra oppure quando viene raggiunto il numero massimo di iterazioni. Ad ogni iterazione viene prima di tutto aggiornata anche la matrice delle LA grazie alla funzione *assign_lower_approx*, riportata in 4, che aggiorna la matrice *lower_approx_matrix* utilizzando quella delle UA andando semplicemente a controllare quali osservazioni sono state assegnate ad una sola UA di un cluster e prendere proprio quest'ultime per la proprietà (P3) per assegnarle alla LU dello stesso cluster.

Listing 4: assign_lower_approx

```
# Assign object to lower approximation out of an upper approximation.
assign_lower_approx = function(upper_approx_matrix) {

  # Initialization of lower_approx_matrix
  lower_approx_matrix = 0 * upper_approx_matrix

  object_idx = which( rowSums(upper_approx_matrix) == 1 )

  lower_approx_matrix[object_idx, ] = upper_approx_matrix[object_idx, ]

  return(lower_approx_matrix)
}
```

Dopodiché vengono calcolate ed aggiornate le matrici dei centroidi delle LA e della *boundary region*, ovvero delle osservazioni che ricadono nella partizione $\overline{A}(\mathbf{c}_g) - \underline{A}(\mathbf{c}_g)$, per il calcolo dei centroidi veri e propri dei cluster del Rough K-Means seguendo l'equazione 10 e successive. Ed è proprio questo che viene fatto successivamente, ovvero inizialmente vengono calcolate le somme di tutte le osservazione presenti nelle LA e nella *boundary region* attraverso un prodotto vettoriale tra matrici col dataset e poi con un ciclo *for* vengono applicate le equazioni 10 e successive a seconda delle cardinalità di $\underline{A}(\mathbf{c}_g)$ e $\overline{A}(\mathbf{c}_g) - \underline{A}(\mathbf{c}_g)$ sfruttando le somme calcolate prima, come spiegato nella sezione precedente. Infine come ultimo passaggio viene salvata la matrice *upper_approx_matrix* dell'iterazione corrente per utilizzarla nell'iterazione successiva nel controllo di convergenza e come ultima

cosa viene aggiornata la matrice delle UA con i centroidi aggiornati calcolati precedentemente, dopodiché l'algoritmo aumenta il contatore delle iterazioni e passa all'iterazione successiva. In output l'algoritmo restituisce matrice delle UA (*upper_ approx_ matrix*), la matrice dei centroidi (*means_ matrix*) e il numero di iterazioni eseguite (num_iterations).

6 Verifiche Sperimentali

Le misure di valutazione, chiamate anche indici, che vengono applicate per giudicare vari aspetti della valutazione del clustering, sono generalmente classificate in due tipologie:

- Misure Non Supervisionate (Indici Interni), comprendono gli indici che valutano la qualità del clustering senza riferirsi ad informazioni esterne.
- Misure Supervisionate (Indici Esterni), valutano quanto corrisponde la struttura o un pattern trovato da un algoritmo di clustering rispetto ad una struttura fornita esternamente, come ad esempio le etichette di classe dei reali cluster contenuti nel dataset, utilizzano dunque delle informazioni che non sono presenti nel set di dati.

Come indice interno, dato che andremo a testare algoritmi di clustering *soft* o *fuzzy* è stato deciso di utilizzare il *Fuzzy Silhouette Index*. Esso si basa sul coefficiente di *silhouette* standard che combina le misure di coesione e separazione dei cluster.

Per calcolare il coefficiente di *silhouette* standard per l'i-esimo oggetto di un set di dati usiamo la seguente formula:

$$s_i(k) = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{13}$$

dove a_i rappresenta la distanza media tra l'oggetto coinvolto e tutti gli oggetti appartenente allo stesso cluster mentre b_i rappresenta la distanza media più bassa di i rispetto a qualsiasi altro cluster dove i non è presente.

Il valore del coefficiente di *silhouette* può variare da -1 a 1. Un valore negativo non è desiderabile perché avremo che la distanza media dagli oggetti del proprio cluster risulti maggiore della minima distanza media dagli oggetti in un differente cluster. Vogliamo dunque che il coefficiente di *silhouette* sia positivo, ovvero $a_i < b_i$, e vogliamo che a_i sia il più vicino possibile a 0,

dato che in questo caso otterremo il valore massimo 1 del coefficiente. Per calcolare la *silhouette* di un cluster basterà calcolare la media del coefficiente degli oggetti appartenenti a tale cluster, mentre se vogliamo una misura media globale basterà eseguire la media del coefficiente di *silhouette* di tutti gli oggetti del set di dati. Il *Fuzzy Silhouette* invece di conseguenza viene calcolato con al seguente formula:

$$FS(k) = \frac{\sum_{i=1}^{n}(u_{ig} - u_{ig'})^{\alpha} s_i(k)}{\sum_{i=1}^{n}(u_{ig} - u_{ig'})^{\alpha}} \tag{14}$$

dove $s_i(k)$ il coefficiente di *silhouette* per l'oggetto i, u_{ig} e $u_{ig'}$ sono il primo e il secondo più grande dell'i-esima riga della matrice dei gradi di appartenenza U e α è un coefficiente di ponderazione (in genere $\alpha = 1$). In questo caso, come esempio, l'indice è stato applicato ad un *Fuzzy K-Means* ma si può facilmente estendere anche agli altri algoritmi utilizzando le matrice di appartenenza ai cluster corrispondenti.

Come indici esterni invece utilizzeremo due metriche la *Purity* e l'*Adjusted Rand Index* (ARI). Per quanto riguarda la *Purity* se supponiamo che $C = \{C_1, C_2, \ldots, C_K\}$ sia il set dei cluster restituiti dall'algoritmo di clustering che vogliamo valutare e $C' = \{C'_1, C'_2, \ldots, C'_K\}$ sia il set delle classi dei dati, ovvero, il set ricavato dalle etichette di classe per il set di dati in esame, dove C_i rappresenta l'insieme degli oggetti appartenenti al cluster i, mentre C'_j rappresenta l'insieme degli oggetti appartenenti alla classe j. Per calcolare la *purity*, ogni cluster è assegnato alla classe più frequente presente in esso e successivamente l'accuratezza di tale assegnamento viene misurata contando il numero di oggetti correttamente assegnati e dividendolo per il numero totale degli oggetti del set di dati, che chiameremo N. Formalmente dunque la *Purity* viene calcolata nel modo seguente.

$$Purity = \frac{1}{N} \sum_{i=1}^{K} \max_{1 \leq j \leq K} |C_i \cap C'_j| \tag{15}$$

Un clustering di bassa qualità avrà una *Purity* con un valore vicino a 0, mentre uno di alta qualità avrà una valore vicino a 1.

L'*Adjusted Rand Index* (ARI) invece è la versione corretta per la casualità dell'indice Rand. Tale correzione stabilisce una linea di base utilizzando la somiglianza attesa di tutti i confronti a coppie tra clustering specificati da un modello casuale. Tradizionalmente, l'indice Rand viene corretto utilizzando il *Permutation Model* per il clustering, ovvero, il numero e la dimensione dei

cluster all'interno di un clustering sono fissi e tutti i clustering casuali vengono generati mescolando gli elementi tra i cluster fissi. Sebbene l'indice Rand possa produrre solo un valore compreso tra 0 e 1, l'indice Rand rettificato può produrre valori negativi se l'indice è inferiore all'indice atteso. L'ARI originale utilizzando il modello di permutazione è calcolato in modo seguente:

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2}[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}} \tag{16}$$

dove n_{ij}, a_i e b_j sono valori presi dalla tabella di contingenza.

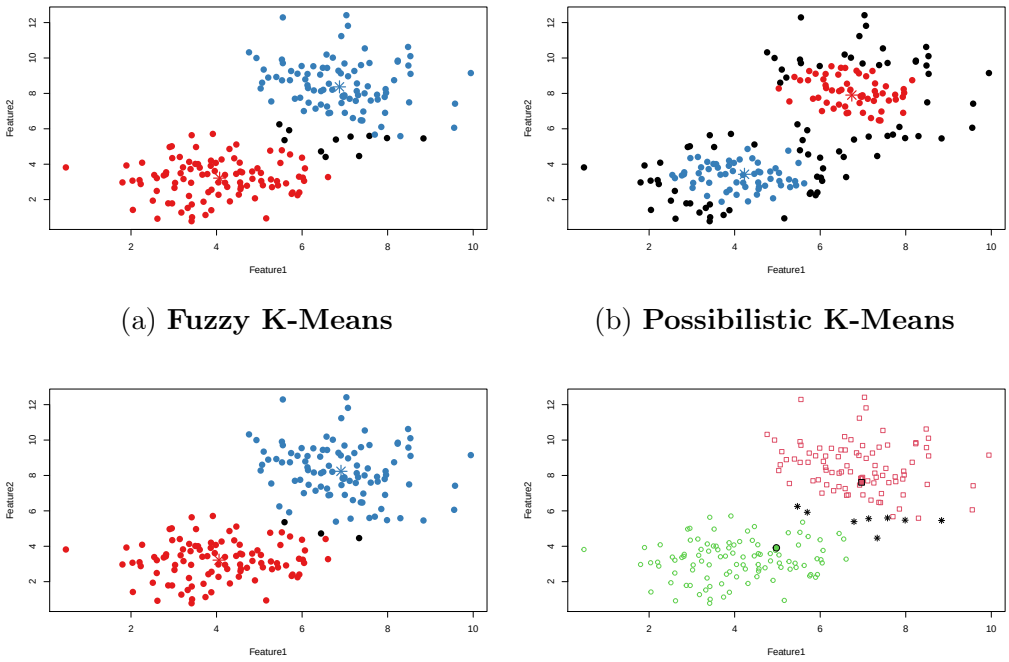
Per il Rough K-means data la sua rappresentazione delle partizioni *soft* non sarebbe direttamente valutabile con gli indice presentati precedentemente così come è stato descritto, ma è stato comunque tentato di interpretare le UA e LA come *membership degrees* assegnando casualmente ai cluster gli oggetti incerti nelle *boundary regions*.

6.1 Test e Risultati su Quattro Dataset

Per confrontare le prestazioni dei quattro algoritmi, vengono analizzati 4 semplici dataset. Tre di questi sono dataset sintetici bidimensionali con 2 cluster presenti in essi: *DemoDataC2D2a*, contenente 200 oggetti e usato in genere per scopi dimostrativi e preso dal pacchetto R *SoftClustering*, *G2*, dataset di cluster Gaussiani di 2048 oggetti e *Synth* contenente 1000 oggetti e generato casualmente grazie al pacchetto R *mlbench*. Inoltre i vari metodi sono stati testati anche con il classico dataset *Iris*, formato da tre cluster, per sfruttarne i label di classe e dunque utilizzare anche gli indici esterni.

Gli algoritmi presentati sono implementati in vari pacchetti R. In particolare, la funzione *FKM* del pacchetto R *fclust* è stata utilizzata per FkM, la funzione *pcm* del pacchetto R *ppclust* per PkM, la funzione *Mclust* del pacchetto *mclust* per FMG e la funzione *rough_kmeans* implementata da zero vista nelle sezioni precedenti.

Nella Figura 1 vengono riportati i risultati dei clustering ottenuti sul dataset *DemoDataC2D2a* usando i colori blu rosso e verde per differenziare gli oggetti assegnati a cluster diversi e utilizzando il colore nero per quelli oggetti che hanno gradi di appartenenza inferiori a 0.7 nel caso del FkM, oggetti con valori di tipicità inferiori a 0.5 per il PkM, oggetti con probabilità a posteriori inferiori a 0.7 per l'FMG e oggetti in entrambi le UA dei due cluster per il RkM, in altre parole con il colore nero sono rappresentate le partizioni *soft*



(a) **Fuzzy K-Means** (b) **Possibilistic K-Means**

(c) **Finite Mixture Gaussian Model** (d) **Rough K-Means**

Figura 1: Clustering del Dataset *DemoDataC2D2a*

del clustering. Come possiamo vedere, per il FkM la maggior parte degli oggetti ha dei gradi di appartenenza superiori allo 0.7 e mostra un clustering abbastanza simile a quello del RkM dove solo una piccola parte degli oggetti appartiene ad entrambe le UA dei cluster. Il PkM è quello che ottiene il clustering più diverso per via di come funziona l'algoritmo e per come sono interpretate le appartenenze ai cluster con le tipicità, la maggior parte degli oggetti infatti ha dei gradi di tipicità inferiori allo 0.43. Il FMG invece presenta delle probabilità a posteriori che sono in generale superiori ai gradi di appartenenza e di tipicità dato che solo 3 oggetti presentato probabilità a posteriori inferiori allo 0.7.

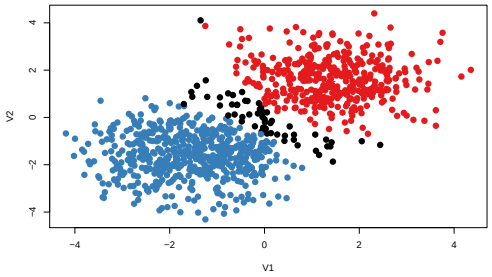
I prototipi risultanti sono invece riportati in Tabella 1. Come si può notare, quelli ottenuti mediante FkM e FMG sono molto simili, mentre quelli ottenuti con RkM sono i più diversi dagli altri. Questo è dovuto al modo in cui sono costruiti, descritto nelle sezioni precedenti.

Un discorso molto simile può essere affrontato anche per il clustering del

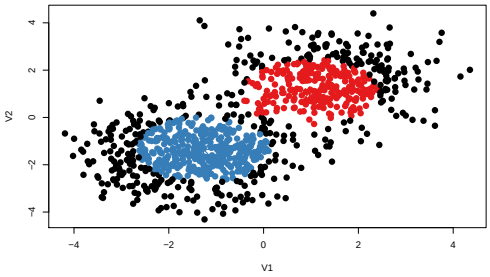
	FkM		PkM		FMG		RkM	
	Clus 1	Clus 2	Clus 1	Clus 2	Clus 1	Clus 2	Clus 1	Clus 2
Feature 1	4.072	6.874	4.230	6.739	4.058	6.909	4.975	6.967
Feature 2	3.213	8.369	3.430	7.905	3.217	8.234	3.911	7.610

Tabella 1: Prototipi del Dataset *DemoDataC2D2a*

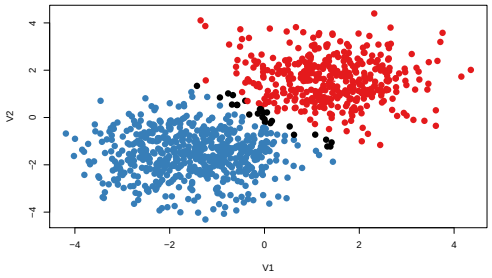
dataset *Synth*, mostrato nella Figura 2, dove anche qui vengono confermate le stesse osservazioni fatte prima.



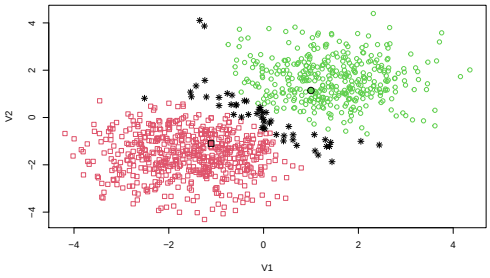
(a) **Fuzzy K-Means**



(b) **Possibilistic K-Means**



(c) **Finite Mixture Gaussian Model**

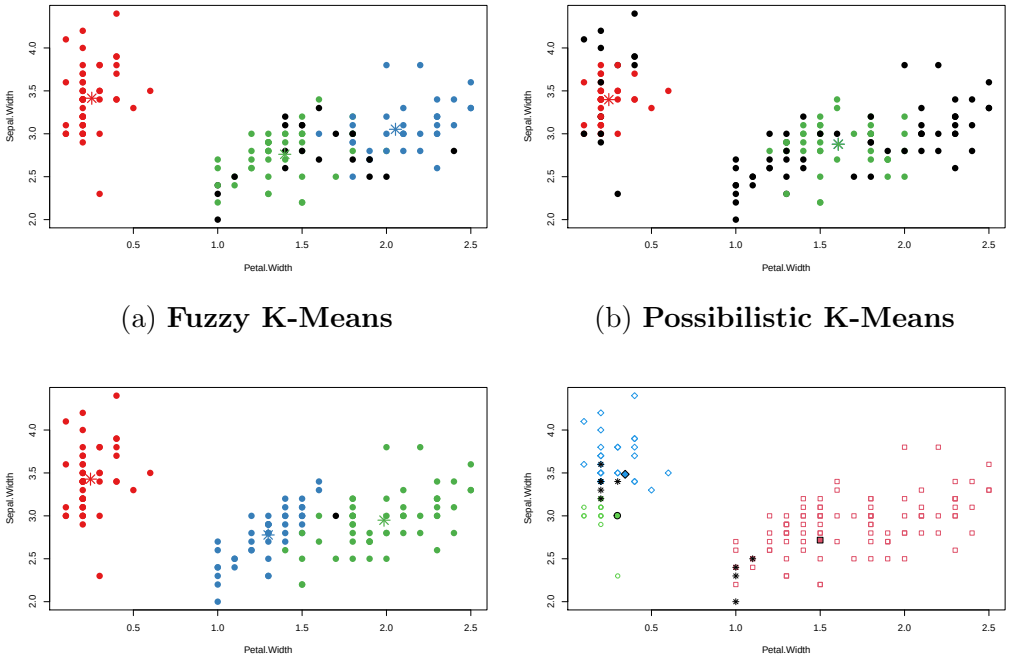


(d) **Rough K-Means**

Figura 2: Clustering del Dataset *Synth*

Riportiamo anche i risultati del clustering ottenuti sul dataset *Iris* nella Figura 3. Anche in questo caso possiamo vedere come per il FkM abbiamo che la maggior parte degli oggetti hanno gradi di appartenenza superiori allo 0.7, con qualche oggetto più incerto tra il secondo il terzo cluster. Per il PkM invece notiamo come nel dataset dell'*Iris* l'algoritmo incappa nel suo punto debole dei cluster coincidenti, infatti il secondo e il terzo cluster presentano

lo stesso centroide e questo si ripercuoterà in seguito sulla valutazione degli indici esterni. L'FMG invece riesce ad ottenere quasi per ogni oggetto delle probabilità a posteriori superiori allo 0.7, distinguendo bene i tre cluster del dataset. Mentre l'RkM riesce a trovare tre cluster ma con centroidi abbastanza diversi rispetto agli altri metodi e con solo una piccola minoranza di oggetti che sono stati associati a tutte e tre le UA dei cluster.



(a) **Fuzzy K-Means** (b) **Possibilistic K-Means**

(c) **Finite Mixture Gaussian Model** (d) **Rough K-Means**

Figura 3: Clustering del Dataset *Iris*

Successivamente riportiamo i risultati ottenuti utilizzando l'indice interno *Fuzzy Silhouette* e gli indici esterni *Purity* e l'*Adjusted Rand Index* (ARI) sul dataset *Iris* rispettivamente nelle Tabelle 2 e 3.

Possiamo notare come l'algoritmo PkM ottiene i valori più alti per il *Fuzzy Silhouette Index* riducendo dunque a generare dei cluster coesi e ben separati fra loro ma comportandosi un po peggio nel caso del dataset *Iris*. Mentre gli altri metodi ottengono comunque dei valori più che soddisfacenti a parte l'FMG sempre nel caso del dataset *Iris*. Per quanto riguarda gli indice esterni invece vediamo come i metodi che si comportano meglio sono l'FkM e l'FMG

dato che il PkM come già detto presenta il problema dei cluster coincidenti e nel caso del RkM per via del particolare tipo di partizionamento *soft* che esegue, ma si è voluto comunque provare a interpretare le varie UA come una matrice dei gradi di appartenenza per poterci applicare sopra gli indici precedenti, sia interni che esterni.

Infine vengono riportati i tempi di esecuzione in secondi nella Tabella 4 dove possiamo vedere che in generale l'algoritmo più pesante è il PkM dato che al suo interno riutilizza alcuni valori del FkM come spiegato precedentemente, mentre gli altri metodi su dataset di dimensioni simili a quelli testati hanno tutti tempi di esecuzioni accettabili, il più veloce risulta essere senza sorpresa l'algoritmo più semplice, ovvero, il FkM.

- Fuzzy Silhouette Index -	**DemoDataC2D2a**	**G2**	**Synth**	**Iris**
Fuzzy K-Means	0.8476	0.8525	0.8223	0.8091
Possibilistic K-Means	**0.8564**	**0.8633**	**0.8230**	**0.9538**
Model-Based Clustering	0.8280	0.8346	0.8038	0.6579
Rough K-Means	0.8370	0.8388	0.8118	0.7532

Tabella 2: Risultati per il *Fuzzy Silhouette Index*

- External Indices -	**Purity**	**ARI**
Fuzzy K-Means	0.9667	0.9039
Possibilistic K-Means	0.6667	0.5681
Model-Based Clustering	0.9667	0.9039
Rough K-Means	0.5200	0.4413

Tabella 3: Indici Esterni sul Dataset *Iris*

- Execution Times -	**DemoDataC2D2a**	**G2**	**Synth**	**Iris**
Fuzzy K-Means	**0.0070**	**0.3590**	**0.1940**	**0.0050**
Possibilistic K-Means	0.7640	5.2400	4.7044	1.3802
Model-Based Clustering	0.0240	1.765	0.3060	0.0680
Rough K-Means	0.1170	0.5850	0.3500	0.0140

Tabella 4: Indici Esterni sul Dataset *Iris*