# Gaussian Mixture Models

Paolo Lazzaroni

August 2019

# Contents

**Abstract**

I metodi di clustering sono ampiamente usati per la loro capacità di sintetizzare grandi quantità di dati in gruppi, a volte come supporto per aiutare a comprendere la natura stessa dei dati e permettere una discriminazione dei dataset in questione e, altre volte per riuscire ad individuare outliers che potrebbero essere dannosi per la successiva trattazione con altri metodi o ancora veri e propri fenomeni da eliminare, come nell'applicazione di *anomaly detection* – ad esempio dei prodotti difettosi in una catena produttiva o componenti difettosi di macchinari.

Il metodo più rinomato e semplice per fare clustering è il *K-means*, ma discrimina immediatamente i dati in cluster senza dare informazioni di tipo probabilistico che potrebbero essere utili sia per verificare la veridicità dell'associazione a un cluster, sia per la generazione di dati dai cluster ottenuti. Per questo motivo il mio interesse va a spostarsi sui *Gaussian Mixture Models* (GMM).

Si introducono i fondamenti matematici che stanno dietro la tecnica e i modelli che permettono la massimizzazione della funzione di valutazione della stessa, per poi passare a un metodo iterativo per l'identificazione di soluzioni attraverso l'algoritmo EM, *Expectation-Maximization*. Questa non è l'unica via per procedere ma è un buon metodo per mostrare, senza entrare troppo nel dettaglio, il metodo GMM.

Si presenterà infine un'applicazione della tecnica al dataset *Seeds* per valutarne le prestazioni e gli eventuali sviluppi.

# Chapter 1

# Clustering

## 1.1 Introduzione

Il clustering è una tecnica di tipo *non supervisionato* molto usata nel machine learning e data mining. Nelle tecniche non supervisionate cerchiamo di apprendere delle strutture interne ad un dataset e in questo caso, vogliamo potere raggruppare i dati e discriminarli in *cluster*, così da ridurre le loro dimensioni e renderli più significativi. Il dataset $D$, in questo caso, si presenta composto da singoli sample $\boldsymbol{x} \in \mathbb{R}^M$ e non ha alcun tipo di target come, invece, succede nelle tecniche supervisionate. Il dataset quindi, se $N$ è il numero di sample che abbiamo

$$D = \{\boldsymbol{x}_n\}_{n=1,2,\cdots,N} \tag{1.1}$$

## 1.2 K-means

Per prima cosa andiamo a mostrare la tecnica più famosa e semplice utilizzata nel clustering, il *K-Means*, visto che pone le basi per quelle più avanzate e, alle volte, è utilizzata per inizializzare altri metodi.

### 1.2.1 Idea di base

L'idea base del $K$-means è quella di sfruttare un vettore media per rappresentare i $K$ cluster $\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_K)$ dove la media $k$-esima $\boldsymbol{\mu}_k$ rappresenta il centro del $k$-esimo cluster. Introduciamo inoltre un parametro $r_{nk} = \{0, 1\}$ che avrà valore $r_{nk} = 1$ se $\boldsymbol{x}_n$ appartiene al $k$-esimo cluster e $r_{nk} = 0$ altrimenti. Possiamo allora esprimere la *funzione obiettivo* come

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 \tag{1.2}$$

che rappresenta la somma quadra delle distanze tra i sample e i relativi centri di cluster.

### 1.2.2 Algoritmo EM per K-means

Possiamo, per minimizzare la quantità $J$, considerare l'algoritmo EM. L'acronimo di questo algoritmo sta per *Expectation-Maximization*, che sono esattamente gli step che compie lo stesso nelle sue iterazioni. Nel caso particolare del $K$-means, partiamo con lo scegliere i valori $\boldsymbol{\mu}_k$ in modo randomico e aggiorniamo i valori degli $r_{nk}$ (E step)

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j ||\boldsymbol{x}_n - \boldsymbol{\mu}_j||^2 \\ 0 & \text{otherwise} \end{cases} \tag{1.3}$$

e di seguito i valori di $\boldsymbol{\mu}_k$ ottenuti minimizzando $J$ (M step)

$$\frac{dJ}{d\boldsymbol{\mu}_k} = 2\sum_{n=1}^{N} r_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k) = 0 \tag{1.4}$$

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} r_{nk}\boldsymbol{x}_n}{\sum_{n=1}^{N} r_{nk}} \tag{1.5}$$

iterativamente fino a convergenza. La convergenza è assicurata visto che ogni step riduce il valore di $J$, anche se potrebbe convergere in un minimo locale.

### 1.2.3 Considerazioni ed estensioni

Esistono molte considerazioni da fare su questo metodo per ottenere migliori risultati o estenderne l'utilizzo, visto che il $K$-means assume di ottiene cluster sferici, ben separati, con lo stesso volume ed aventi lo stesso numero di punti. Senza guardare nuove tecniche, possiamo fare diverse osservazioni:

- Per velocizzare il processo di minimizzazione, solitamente, si possono scegliere valori iniziali dei $\boldsymbol{\mu}_k$ uguali a un subset randomico di $K$ punti appartenenti a $D$.
- Abbiamo considerato una cosiddetta *batch version* del $K$-means, ossia in cui utilizziamo tutti i dati assieme e, in alcune applicazioni, questo potrebbe non essere fattibile per questioni di tempo o perchè i dati arrivano non tutti assieme ma uno dopo l'altro nel tempo. Potremmo allora essere interessati a un aggiornamento di tipo on-line come

  $$\boldsymbol{\mu}_k^{new} = \boldsymbol{\mu}_k^{old} + \eta_n(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{old}) \tag{1.6}$$

  dove $\eta_n$ è chiamato *learning rate*.
- L'algoritmo si mostra essere poco robusto agli outliers utilizzando la distanza euclidea. Si possono provare ad usare diverse distanze come, ad esempio, la *distanza Manhattan* o la *distanza Mahalanobis*.
- L'algoritmo si può applicare, in questa forma, solo a dati quantitativi. Per estendere lo stesso a dati categorici, si può sostituire la distanza euclidea con una misura di dissimilarità $\mathcal{V}(\boldsymbol{x}, \boldsymbol{x}')$. L'algorimo prende in questo caso il nome di *K-medioids*.

La complessità di $K$-means è $O(KN)$.

## 1.3 EM per gaussian mixture models

Il $K$-means adopera un cosiddetto assegnamento "forte" ai cluster trovati con l'algoritmo, ossia ogni punto è associato direttamente a un singolo cluster e non ha alcuna relazione con gli altri. Per alcuni problemi questa assunzione potrebbe essere scomoda o non dare abbastanza informazioni sulla natura dei dati presi in considerazione.

L'algoritmo che andiamo ad analizzare ora, ossia EM applicato ai *Gaussian Mixture Models* (GMM), opera un assegnamento "lasco" ai cluster, basandosi sulle probabilità a posteriori e restituendo un valore probabilistico di appartenenza a un cluster al posto di fare un assegnamento diretto senza ulteriori motivazioni. Questa via, oltre che permetterci di sviluppare modelli più complessi, restituisce anche più informazioni sui cluster, non solo la media, e sui dati stessi in relazione ad ogni cluster.

### 1.3.1 Gaussian mixture models

La *distribuzione gaussiana* ha delle ottime proprietà analitiche e, per questo, è molto sfruttata nel mondo statistico. Ricordiamo che la forma della densità di probabilità $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ di una gaussiana multivariata, con $\boldsymbol{x}$ variabile casuale continua, media $\boldsymbol{\mu}$ e matrice di covarianza $\boldsymbol{\Sigma}$, ha la forma

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\Big\{ -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\Big\} \tag{1.7}$$

dove l'apice $^T$ indica la matrice o, in questo caso, il vettore trasposto.

L'unico problema di questa distribuzione, quando andiamo a scontrarci con i dati del mondo reale, è che risulta essere molto limitata e semplificata rispetto al vero modello delle popolazioni. Da qui l'idea di unire l'utile al dilettevole con una *mistura* di distribuzioni gaussiane. Un modello simile si può ottenere considerando la sovrapposizione di $K$ gaussiane, chiamate *componenti* della mistura, ognuna con la sua media $\boldsymbol{\mu}_k$ e varianza $\boldsymbol{\Sigma}_k$, pesate da dei *coefficienti di mistura* $\pi_k$ ottenendo la forma

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1.8}$$

Integrando l'equazione ottenuta per ambo i lati rispetto a $\boldsymbol{x}$ e ricordando che vale per la probabilità che $p(\boldsymbol{x}) \geq 0$ troviamo che

$$\sum_{k=1}^{K} \pi_k = 1 \tag{1.9}$$

e che

$$0 \leq \pi_k \leq 1 \tag{1.10}$$

per $k = 1, 2, \cdots, K$.

### 1.3.2 Variabili latenti discrete

Per giustificare la successiva applicazione dell'algoritmo EM, facciamo uso di una diversa formulazione per la mistura di gaussiane secondo variabili discrete *latenti*. Introduciamo una variabile binaria discreta $\boldsymbol{z} \in \mathbb{R}^K$ in cui un particolare elemento $z_k$ ha valore 1 e tutti gli altri 0 così da avere $z_j \in \{0, 1\}$ con $j = 1, 2, \cdots, K$ e $\sum_j z_j = 1$.

Vogliamo definire la distribuzione congiunta $p(\boldsymbol{x}, \boldsymbol{z})$ e per questo ci servono la marginale $p(\boldsymbol{z})$ e la condizionata $p(\boldsymbol{x}|\boldsymbol{z})$. La distribuzione marginale di $\boldsymbol{z}$ è espressa rispetto ai coefficienti di mistura $\pi_k$ e, date le precedenti definizioni, possiamo scrivere

$$p(z_k = 1) = \pi_k \tag{1.11}$$

o equivalentemente

$$p(\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k^{z_k} \tag{1.12}$$

vista la rappresentazione *one hot* di $\boldsymbol{z}$.

La condizionata è invece rappresentabile come

$$p(\boldsymbol{x}|z_k = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1.13}$$

o ancora

$$p(\boldsymbol{x}|\boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k} \tag{1.14}$$

Da queste espressioni possiamo ricavare la congiunta come il prodotto $p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})$ e la marginale di $\boldsymbol{x}$ tramite il teorema della probabilità totale

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1.15}$$

che è esattamente l'espressione della mistura di gaussiane considerata in (1.8).

Prima di passare all'algoritmo, abbiamo bisogno di un'altra quantità che gioca un ruolo fondamentale nel modello, la probabilità condizionata di $\boldsymbol{z}$ dato $\boldsymbol{x}$. Il suo valore è dato dal teorema di Bayes

$$\begin{aligned} \gamma(z_k) \equiv p(z_k = 1|\boldsymbol{x}) &= \frac{p(z_k = 1)p(\boldsymbol{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\boldsymbol{x}|z_j = 1)} = \\ &= \frac{\pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \end{aligned} \tag{1.16}$$

e queste quantità possono essere viste come la *responsabilità* che il componente $k$ si prende per spiegare l'osservazione $\boldsymbol{x}$.

Dobbiamo considerare le $\pi_k$ come le probabilità a priori di $z_k = 1$ e le $\gamma(z_k)$ come le corrispondenti probabilità a posteriori data l'osservazione di $\boldsymbol{x}$.

### 1.3.3 Considerazioni sulla verosimiglianza

Supponiamo di avere delle osservazioni $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$ e vogliamo modellare questo dataset con una mistura di gaussiane. Avremo la matrice delle osservazioni $\boldsymbol{X} \in \mathbb{R}^{N \text{x} M}$, in cui l'$n$-esima riga è $\boldsymbol{x}_n^T$, e la matrice delle variabili latenti $\boldsymbol{Z} \in \mathbb{R}^{N \text{x} K}$, in cui l'$n$-esima riga è $\boldsymbol{z}_n^T$. Assumiamo anche che i dati siano stati presi in modo indipendente dalla distribuzione (i.i.d.). Il logaritmo della funzione di verosimiglianza è dato dall'espressione

$$\ln p(\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \right\} \tag{1.17}$$

Ci sono due importanti considerazioni da fare su questa funzione che ci spingono all'uso di tecniche diverse rispetto alla massimizzazione analitica della log verosimiglianza.

Innanzitutto possono essere presenti delle singolarità. Si consideri il caso semplificato $\mathbf{\Sigma}_k = \sigma_k^2 \boldsymbol{I}$, con $I$ matice identità delle appropriate dimensioni, che si può dimostrare valere anche per una matrice di covarianza generica. Supponiamo che la componente $j$-esima abbia la media uguale a un generico punto $n$-esimo del dataset $\boldsymbol{\mu}_j = \boldsymbol{x}_n$. Il contributo di questo punto sarà

$$\mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_j = \boldsymbol{x}_n, \mathbf{\Sigma}_j = \sigma_j^2 \boldsymbol{I}) = \frac{1}{(2\pi)^{1/2}} \frac{1}{\sigma_j} \tag{1.18}$$

e vediamo facilmente che questo termine, se $\sigma_j \to 0$, andrà all'infinito.

Un altro problema nel massimizzare la funzione di verosimiglianza sta nella complessità del problema. Una generica mistura di gaussiane con $K$ componenti avrà $K!$ soluzioni equivalenti, corrispondenti ai $K!$ modi di assegnare $K$ set di parametri a $K$ componenti. Inoltre è molto complesso massimizzare questa espressione a causa della somma su $k$ che appare all'interno del logaritmo, che non agisce più direttamente sulla gaussiana. Ponendo le derivate della log verosimiglianza a zero non troveremo più una soluzione chiusa.

### 1.3.4 Algoritmo EM per GMMs

Per massimizzare la log verosimiglianza, calcoliamo le derivate rispetto a $\boldsymbol{\mu}_k$ ottenendo

$$\begin{aligned} 0 &= -\sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \mathbf{\Sigma}_j)} \mathbf{\Sigma}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \\ &= -\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{\Sigma}_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \end{aligned} \tag{1.19}$$

e, supponendo che la matrice di covarianza $\mathbf{\Sigma}_k$ non sia singolare, ricaviamo

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n \tag{1.20}$$

dove possiamo interpretare $N_k$ come l'effettivo numero di punti assegnati al cluster $k$. Notiamo che la media $\boldsymbol{\mu}_k$ è ottenuta da una media pesata di tutti i punti $\boldsymbol{x}_n$ del dataset rispetto alle probabilità a posteriori $\gamma(z_{nk})$.

Valutando le derivate della log verosimiglianza rispetto a $\mathbf{\Sigma}_k$ e usando simili passaggi, troviamo l'espressione

$$\mathbf{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \tag{1.21}$$

che è identica a quella della gaussiana standard ma pesata ancora secondo le probabilità a posteriori.

Infine massimizziamo la log verosimiglianza rispetto ai coefficienti di mistura $\pi_k$. I coefficienti, però, sono sottoposti a delle limitazioni che abbiamo trovato nell'espressione (1.9). Dobbiamo allora modificare l'espressione, introducendo un moltiplicatore di Lagrange e massimizzare la seguente quantità

$$\ln p(\boldsymbol{X} | \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{\Sigma}) + \lambda \left( \sum_{k=1}^{K} \pi_k - 1 \right) \tag{1.22}$$

ponendo la derivata uguale a zero

$$0 = \sum_{n=1}^{N} \frac{\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda \tag{1.23}$$

e con qualche passaggio

$$\begin{aligned} \lambda &= -\sum_{n=1}^{N} \frac{\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \\ \pi_k \lambda &= -\sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = -\sum_{n=1}^{N} \gamma(z_{nk}) = -N_k \\ \sum_{k=1}^{K} \pi_k \lambda &= -\sum_{k=1}^{K} N_k = -N \end{aligned} \tag{1.24}$$

e visto che la costrizione dice che $\sum_{k=1}^{K} \pi_k = 1$, allora si avrà $\lambda = -N$ che ci porta ad eliminare $\lambda$ e ottenere, tramite la (1.24)

$$\pi_k = \frac{N_k}{N} \tag{1.25}$$

che possiamo interpretare come la responsabilità media che ha il $k$-esimo componente nello spiegare i punti del dataset.

Grazie a queste considerazioni abbiamo uno schema iterativo che possiamo utilizzare per l'algoritmo EM. Si noti che queste espressioni non danno una soluzione chiusa, visto che le $\gamma(z_{nk})$ dipendono a loro volta dai parametri $\pi_k$, $\boldsymbol{\mu}_k$ e $\boldsymbol{\Sigma}_k$ in modo molto complesso come da (1.16).

Gli step dell'algoritmo sono riportati nell'algoritmo 1 in pagina successiva.

**Algorithm 1** Algoritmo EM per GMMs

1. Inizializzare le medie $\boldsymbol{\mu}_k$, le matrici di covarianza $\boldsymbol{\Sigma}_k$ e i coefficienti di mistura $\pi_k$ e valutare il valore iniziale della log verosimiglianza.

2. Valutare le responsabilità con i parametri correnti (E step)

$$\gamma(z_k) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{1.26}$$

3. Stimare i nuovi parametri usando le responsabilità trovate

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n \tag{1.27}$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{\text{new}})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{\text{new}})^T \tag{1.28}$$

$$\pi_k^{\text{new}} = \frac{N_k}{N} \tag{1.29}$$

dove

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}). \tag{1.30}$$

4. Valutare la log verosimiglianza

$$\ln p(\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \right\} \tag{1.31}$$

e controllare la convergenza dei parametri o della log verosimiglianza. Se la convergenza non è stata raggiunta, ritornare allo step 2.

# Chapter 2

# Caso di studio

## 2.1 Introduzione

Il problema posto dal caso di studio che andiamo a considerare è quello di una classificazione di 3 tipi di grano (Kama, Rosa e Canadian) secondo delle misurazioni fatte sulle proprietà geometriche e strutturali dei chicchi tramite tecniche di "soft X-rays". Si tratta di un problema prettamente di classificazione, ma anche tecniche di clustering possono dare informazioni ed essere utili alle fasi successive.

Per aiutarci nell'analisi del dataset e nello sviluppo del modello GMM, si è scelto Python come linguaggio di programmazione data la sua versatilità e la quantità (e qualità) di librerie dedicate all'algebra, allo studio di dataset tramite machine learning e alla visualizzazione dei dati.

## 2.2 Il dataset

Il dataset "Seeds" contiene un totale di $N = 210$ elementi di dimensione $M = 7$ ognuno, più la label del tipo di grano identificato dal vettore. Abbiamo quindi $\boldsymbol{X} \in \mathbb{R}^{210\text{x}7}$. Visto che stiamo applicando tecniche di clustering, metodo non supervisionato, non ci interessa la label, se non per le considerazioni finali, ma vogliamo ottenere delle strutture che saranno di supporto nell'analisi del dataset.

I sample hanno i seguenti attributi:

1. Area $A$
2. Perimetro $P$
3. Compattezza, calcolata come $C = \frac{4\pi A}{P^2}$
4. Lunghezza del chicco
5. Larghezza del chicco
6. Coefficiente di asimmetria
7. Lunghezza dell'incavo del chicco

ognuno dei quali è un numero appartenente ai reali, come già specificato sopra.

## 2.3 Visualizzazione del dataset

Visto che il dataset è composto da elementi in $\mathbb{R}^7$ non possiamo visualizzarlo direttamente, ma dobbiamo prima applicare allo stesso *tecniche di visualizzazione*. Vogliamo visualizzare i dati in uno spazio $\mathbb{R}^2$ e, per farlo, sono state scelte la tecniche più tipiche: il *MultiDimensional Scaling* (MDS), che assicura che punti vicini nello spazio originario siano vicini anche nello spazio di dimensione ridotta, e il *Principal Component Analysis* (PCA), che proietta i dati in nuove direzioni, ordinate secondo la loro varianza spiegata, e da cui noi preleviamo tante direzioni quante dimensioni vogliamo.

[1]:
```
import numpy as np
from scipy import linalg

import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import matplotlib.colors as pltcolors

from sklearn.preprocessing import StandardScaler
from sklearn.manifold import MDS
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture

D = np.genfromtxt('seeds_dataset.tsv', delimiter='\t')
X = np.delete(D,7,1) #rimozione della label
print('N =', X.shape[0], ', M =', X.shape[1])
```

```
N = 210 , M = 7
```

Innanzitutto un po' di *preprocessing* sui dati. Normalizziamoli per ottenere migliori risultati nelle fasi successive e avere informazioni sulla media e la varianza dei dati di partenza

[2]:
```
seed = 42
np.random.seed(seed) #per la riproduzione dei risultati
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)
print('Mean:', scaler.mean_, '\nVariance:',scaler.var_)
with np.printoptions(suppress=True): #solo per visualizzare 0 invece di numeri␣
↪infinitesimi
    print('Normalised mean:',X_norm.mean(axis=0),'\nNormalised variance:
↪',X_norm.var(axis=0))
```

```
Mean: [14.84752381 14.55928571  0.87099857  5.62853333  3.25860476  3.70020095
  5.40807143]
Variance: [8.42603482e+00 1.69740663e+00 5.55690522e-04 1.95370458e-01
 1.41988830e-01 2.24991888e+00 2.40402828e-01]
Normalised mean: [-0. -0.  0. -0. -0.  0. -0.]
Normalised variance: [1. 1. 1. 1. 1. 1. 1.]
```

Per quanto riguarda MDS, non usiamo i dati normalizzati, dato che ci servono le distanze vere tra gli stessi per ottenere una corretta rappresentazione in uno spazio di minori dimensioni. I dati verranno solo centrati dalla procedura stessa

[3]:
```
embedding = MDS(n_components=2, random_state=seed)
X_transformed = embedding.fit_transform(X)
print('N =', X_transformed.shape[0], ', M =', X_transformed.shape[1])
```

```
N = 210 , M = 2
```

Come vediamo il dataset è stato ridotto a vettori bidimensionali.

Il valore di stress finale dell'MDS, ossia la somma degli errori quadratici tra la distanza effettiva dei punti nello spazio di partenza e i punti ottenuti dalla trasformazione nello spazio ridotto, ottenuto come

$stress(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N) = \sqrt{\sum_{i \neq j = 1,2,\cdots,N} (d_{ij} - ||\boldsymbol{x}_i - \boldsymbol{x}_j||)^2}$

è pari a

[4]:
```
print('Stress:',embedding.stress_)
```

```
Stress: 53.10153547772634
```

Visualizzando

[5]:
```
fig = plt.scatter(X_transformed[:,0],X_transformed[:,1], label='Data MDS')
plt.legend(loc='best')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.title('MDS')
plt.show()
```

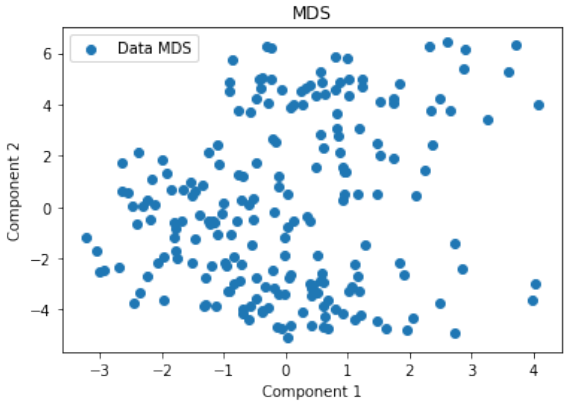


Figure 2.1: Visualizzazione del dataset dopo la riduzione a 2D tramite MDS.

sembrano effettivamente esserci alcune agglomerazioni di dati. A priori non sappiamo quanti cluster ci siano effettivamente, quindi dovremo provare diversi valori per $K$ nell'implementazione di un GMM.

Altro modo per visualizzare i dati in uno spazio ridotto è sfruttare PCA. Sfrutteremo, in questo caso, i dati normalizzati. Per la visualizzazione preleveremo le prime due componenti ottenute dal PCA e vediamo che la percentuale di varianza spiegata delle stesse è

[6]:
```
pca = PCA(n_components=2, random_state=seed)
X_transformedPCA = pca.fit_transform(X_norm)
print('Explained variance ratio for\nComponent 1:',
      pca.explained_variance_ratio_[0],
      '\nComponent 2:',
      pca.explained_variance_ratio_[1])
print('Total explained variance:',np.sum(pca.explained_variance_ratio_))
```

```
Explained variance ratio for
Component 1: 0.7187430265675441
Component 2: 0.1710818352815798
Total explained variance: 0.8898248618491239
```

circa 0.89%. Riteniamo quindi una buona percentuale della varianza totale del dataset, anche se una componente in più non farebbe male. Per semplicità ci limitiamo a visualizzare i dati in uno spazio 2D.

Queste *riduzioni di dimensionalità* dei dati potrebbero tornare utili come fase di *preprocessing* per l'eventuale successiva classificazione, se fosse quello lo scopo dell'analisi, per semplificare i dati stessi in input al metodo di classificazione scelto e velocizzare quindi la fase di *training*.

Visualizzando i dati ottenuti tramite PCA

[7]:
```
fig = plt.scatter(X_transformedPCA[:,0],X_transformedPCA[:,1], label='Data PCA')
plt.legend(loc='best')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.title('PCA')
plt.show()
```

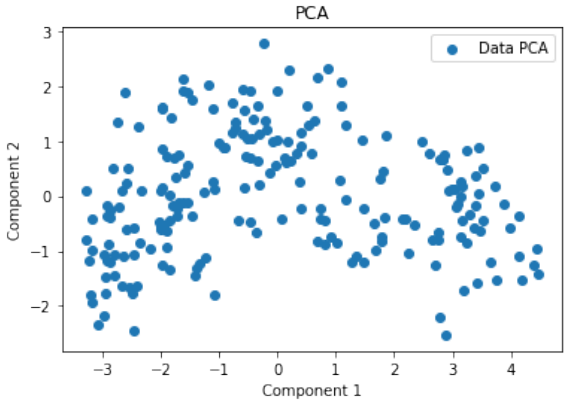


Figure 2.2: Visualizzazione del dataset dopo la riduzione a 2D tramite PCA.

possiamo anche qui notare diverse agglomerazioni di dati il che, ancora una volta, giustificano l'applicazione di un metodo di clustering.

Dopo questa prima analisi visiva, soprattutto aiutata da PCA, e basandoci sulle sezioni dello spazio 2D in cui più dati sono agglomerati, sembrerebbe che i cluster siano 2, 3 o 4. Teniamo questi numeri per $K$ per evitare di fare *overfitting* sul dataset che stiamo valutando. Potremmo infatti, potenzialmente, avere $K = N$ ma non avrebbe più senso fare clustering a quel punto.

## 2.4 Applicazione di GMM sul dataset

Applicheremo GMM prima sul dataset originario e poi su quello ottenuto con la PCA per la visualizzazione e vedremo di comparare i risultati ottenuti. Per inizializzare i centri del GMM utilizzeremo $K$-means, come è di default nell'implementazione usata da scikit-learn.

### 2.4.1 Senza preprocessing

Partiamo con GMM sul dataset iniziale senza preprocessing (i valori della likelihood non sono stampati per mantenere ordine, ma sono stati valutati e si arriva a convergenza in massimo 30 step, nel caso $K = 2$)

[8]:
```
print('NO PREPROCESSING\n')
min_k = 2
max_k = 4
gmm_np = {}
gmm_np_pred = {}
gmm_np_count = {}
for k in range(min_k, max_k+1):
    idx = k - min_k + 1
    gmm_np[k] = GaussianMixture(n_components=k, covariance_type='full',
```

```
                        max_iter=100, init_params='kmeans', random_state=seed).
↪fit(X)
    gmm_np_pred[k] = gmm_np[k].predict(X)
    unique, count = np.unique(gmm_np_pred[k], return_counts=True)
    gmm_np_count[k] = list(zip(unique, count))
    print('k =', k)
    print('AIC:', gmm_np[k].aic(X), ', BIC:', gmm_np[k].bic(X))
    print('Points per cluster: ', gmm_np_count[k])
    print('Weights:', gmm_np[k].weights_, '\nMeans:', gmm_np[k].means_)
  ␣
↪print('=========================================================================')
```

```
NO PREPROCESSING

k = 2
AIC: -1988.8331477876472 , BIC: -1751.188513106707
Points per cluster:  [(0, 102), (1, 108)]
Weights: [0.48265305 0.51734695]
Means: [[17.40618519 15.70345836  0.88454536  5.99069092  3.58928249  3.29082613
   5.77073561]
 [12.46044928 13.49184264  0.85836024  5.29066246  2.95010267  4.08212261
   5.06972793]]
=========================================================================
k = 3
AIC: -2259.187097621821 , BIC: -1901.046591835052
Points per cluster:  [(0, 67), (1, 67), (2, 76)]
Weights: [0.31763136 0.3203806  0.36198804]
Means: [[18.51039381 16.20696129  0.88460422  6.17017178  3.70118352  3.60902317
   6.03105877]
 [14.55627613 14.41021924  0.88009292  5.55125457  3.26967863  2.74788125
   5.14819356]
 [11.89125956 13.24544275  0.85101108  5.22166148  2.86045694  4.62306483
   5.09142977]]
=========================================================================
k = 4
AIC: -2347.0556905356843 , BIC: -1868.4193136430863
Points per cluster:  [(0, 53), (1, 27), (2, 70), (3, 60)]
Weights: [0.25562666 0.1253162  0.33526982 0.28378732]
Means: [[18.96799697 16.39606991  0.88625382  6.23953925  3.75066838  3.51617876
   6.09896161]
 [13.33399908 13.83730872  0.87501339  5.36719331  3.11833176  2.82562929
   4.93322424]
 [11.82166682 13.22505197  0.84867139  5.2245273   2.84411408  4.73137865
   5.10952861]
 [15.37906595 14.79986278  0.88186185  5.67085999  3.36699621  3.033914
   5.34812688]]
```

==========

Secondo AIC il migliore modello è quello con $K = 4$, mentre secondo BIC quello con $K = 3$. Il modello con $K = 2$ sembra non spiegare il dataset come gli altri due.

Guardando alla mistura ottenuta con i valori $K = 3$, i pesi delle 3 gaussiane sono simili, non esiste quindi una gaussiana che pesa molto di più delle altre. Nella mistura con $K = 4$ la prima gaussiana pesa più di due volte rispetto alla quarta, ma anche qui non esistono grossi scompensi.

Dopo queste considerazioni si potrebbe dire che sono entrambi buoni modelli e la scelta tra i due potrebbe essere dettata dalla preferenza tra il criterio AIC e quello BIC oppure dal grado di complessità del modello di uscita (BIC predilige modelli meno complessi rispetto ad AIC, per questo sceglie $K = 3$).

Anche se non stiamo facendo classificazione, visto che abbiamo la label nel dataset di partenza, vediamo quanti elementi effettivamente della stessa specie sono stati raggruppati nello stesso cluster per GMM con $K = 3$. Visualizzando i vettori osserviamo che

[9]:
```
#print(D[:,7],'\n',gmm_np_pred[3]) #commentato per leggibilità
#dal comando si vede la corrispondenza di cluster->label: 0->2, 1->1,2->3
#mappiamo quindi i risultati
res_np = np.copy(gmm_np_pred[3])
replacements_np = {0: 2, 1: 1, 2: 3}
new_np = np.zeros(res_np.shape)
for idx, e in replacements_np.items():
    new_np[res_np == idx] = e

#calcolo del numero di elementi uguali
eq_np = np.where(new_np == D[:,7], 1., 0.)
ratio_correct_np = eq_np.sum()/eq_np.shape[0]
print('Ratio of cluster elements of same seed kind:',ratio_correct_np)
```

```
Ratio of cluster elements of same seed kind: 0.9285714285714286
```

effettivamente i cluster formati sono significativi per identificare lo stesso tipo di seme, con l'92.9% di elementi contenuti nei cluster che sono effettivamente dello stesso tipo di seme. Questa considerazione non ha senso nel clustering, ma mostra in modo inequivocabile che i cluster sono significativi per riassumere i dati.

Per fare le stesse valutazioni su GMM con $K \geq 4$ dovremmo decidere quali cluster unire sotto una singola label, ma eviteremo di farlo in questa trattazione.

### 2.4.2 Con preprocessing

Ora sfruttiamo i dati preprocessati con PCA e vediamo se la situazione cambia (i valori della likelihood non sono stampati per mantenere ordine, ma sono stati valutati e si arriva a convergenza in massimo 8 step, nel caso $K = 4$)

[10]:
```
print('WITH PREPROCESSING\n')
min_k = 2
max_k = 4
```

```
gmm = {}
gmm_pred = {}
gmm_count = {}
for k in range(min_k, max_k+1):
    idx = k - min_k + 1
    gmm[k] = GaussianMixture(n_components=k, covariance_type='full',
                             max_iter=100, init_params='kmeans', random_state=seed).
↪fit(X_transformedPCA)
    gmm_pred[k] = gmm[k].predict(X_transformedPCA)
    unique, count = np.unique(gmm_pred[k], return_counts=True)
    gmm_count[k] = list(zip(unique, count))
    print('k =', k)
    print('AIC:', gmm[k].aic(X_transformedPCA), ', BIC:', gmm[k].
↪bic(X_transformedPCA))
    print('Points per cluster: ', gmm_count[k])
    print('Weights:', gmm[k].weights_, '\nMeans:', gmm[k].means_)
  ␣
↪print('================================================================================')
```

```
WITH PREPROCESSING

k = 2
AIC: 1467.1068015923533 , BIC: 1503.9249844302456
Points per cluster:  [(0, 75), (1, 135)]
Weights: [0.36827641 0.63172359]
Means: [[ 2.51464923 -0.32586688]
 [-1.46596707  0.18997087]]
================================================================================
k = 3
AIC: 1450.0400427000686 , BIC: 1506.9408707222656
Points per cluster:  [(0, 67), (1, 73), (2, 70)]
Weights: [0.31611865 0.33520855 0.3486728 ]
Means: [[ 2.79449827 -0.40445368]
 [-2.31736537 -0.62684562]
 [-0.30570874  0.96933101]]
================================================================================
k = 4
AIC: 1434.5947392844857 , BIC: 1511.5782124909874
Points per cluster:  [(0, 47), (1, 66), (2, 72), (3, 25)]
Weights: [0.22887704 0.32186758 0.33803518 0.1112202 ]
Means: [[ 3.25554294 -0.35453288]
 [-2.31689057 -0.70909697]
 [-0.41250559  1.12929811]
 [ 1.2592526  -0.65062577]]
================================================================================
```

La situazione è cambiata. Il criterio AIC sceglie ancora $K = 4$, ma il criterio BIC sceglie $K = 2$, anche se i valori di questo criterio sono molto simili per tutti i valori di $K$ considerati.

Guardando ai valori dei coefficienti di mistura vediamo che per $K = 2$ la prima gaussiana pesa 1.6 volte di più rispetto alla seconda, sono circa equiprobabili con $\pi_k \approx \frac{1}{3}$ per $K = 3$ e per $K = 4$ abbiamo valori non troppo significativi a prima vista, come prima, senza nessuna gaussiana molto più grande di altre, ma la seconda che pesa il doppio della terza.

Consideriamo allora tutte e 3 le casistiche stavolta riportando i risultati ottenuti in grafici

[11]:
```
#per plottare i risultati ottenuti
# (ispirato a https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm.
 ↪html)
colors = ['red', 'blue', 'green', 'gold']

def plot_results(X, Y_pred, means, covariances, ax, title):
    for i, (mean, covar, color) in enumerate(zip(means, covariances, colors)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # Plot data
        ax.scatter(X[Y_pred == i, 0], X[Y_pred == i, 1], color=color, label=i,␣
 ↪marker='o')
        # Plot centers
        ax.scatter(mean[0], mean[1], s=300, color=color, marker='+')
        # Plot ellipses to show the Gaussian component
        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi  # convert to degrees
        color_rgba = pltcolors.to_rgba(color)
        for cov_factor in range(1, 4):
            ell = Ellipse(xy=mean,
                          width=np.sqrt(v[0]) * cov_factor,
                          height=np.sqrt(v[1]) * cov_factor,
                          angle=180. + angle, color=color, linewidth=.4)
            ell.set_facecolor((color_rgba[0], color_rgba[1], color_rgba[2], 1.0␣
 ↪/ (cov_factor * 4.5)))
            ax.add_artist(ell)
    ax.set_xlabel('Component 1')
    ax.set_ylabel('Component 2')
    ax.legend(loc='best')
    ax.set_title(title)

# Original label for data
fig, axs = plt.subplots(2, 2, figsize=(15, 10), facecolor='w', edgecolor='k')
ax = axs[0, 0]
labels = ['Kama', 'Rosa', 'Canadian']
el_per_label = 70
for i in range(3):
```

```
    ax.scatter(X_transformedPCA[i*el_per_label:(i+1)*el_per_label,0],
               X_transformedPCA[i*el_per_label:(i+1)*el_per_label,1],
               color=colors[i],
               label=labels[i])
ax.set_xlabel('Component 1')
ax.set_ylabel('Component 2')
ax.legend(loc='best')
ax.set_title('Original labels')

# GMM prep k=2
ax = axs[0, 1]
plot_results(X_transformedPCA, gmm_pred[2], gmm[2].means_, gmm[2].covariances_,
 ↪ax, '$K=2$')

# GMM prep k=3
ax = axs[1, 0]
plot_results(X_transformedPCA, gmm_pred[3], gmm[3].means_, gmm[3].covariances_,
 ↪ax, '$K=3$')

# GMM prep k=4
ax = axs[1, 1]
plot_results(X_transformedPCA, gmm_pred[4], gmm[4].means_, gmm[4].covariances_,
 ↪ax, '$K=4$')
plt.show()
```

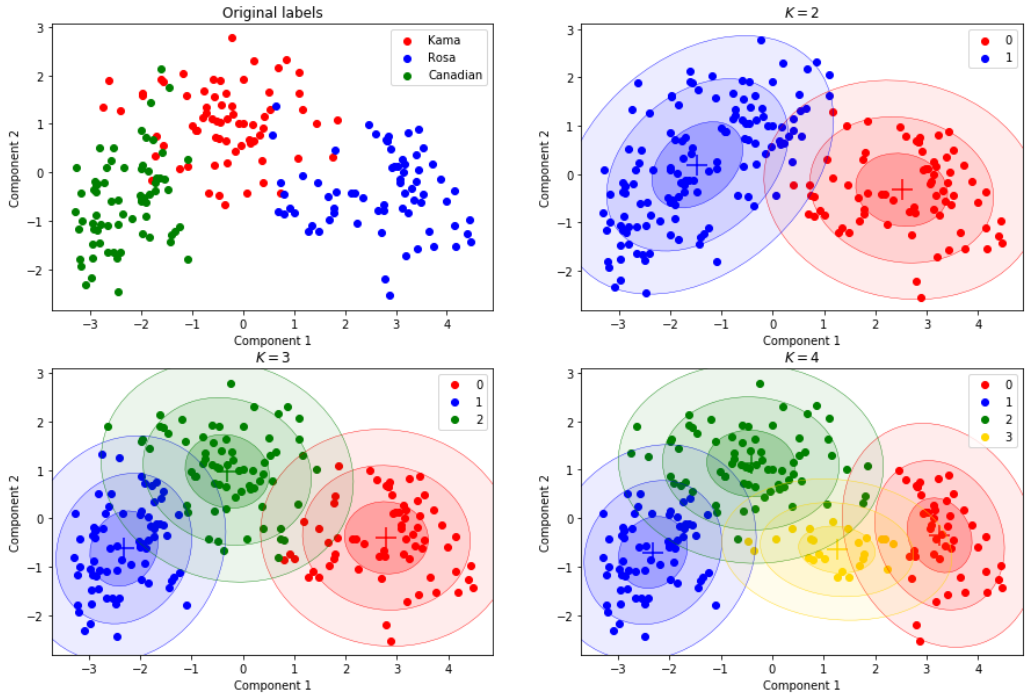


Figure 2.3: Confronto visivo tra i tre modelli ottenuti e il dataset originale, dopo il preprocessing con PCA.

L'analisi visiva è sicuramente più intuitiva rispetto ai numeri. Vediamo rappresentati quattro grafici: il primo, solo di riferimento, mostra le label originali del dataset, mentre gli altri mostrano come l'algoritmo ha strutturato i nostri dati con $K = 2, 3, 4$.

Possiamo notare, come principale differenza tra i modelli, che il cluster $K = 3$ è l'unico ad avere un numero di dati che, già ad occhio, è simile in tutti e 3 cluster creati, e le gaussiane sviluppate sono molto circolari, più che ellittiche, come invece risultano negli altri modelli (in particolare in $K = 2$).

I cluster ottenuti sono buoni, in particolare quelli di $K = 2, 3$ erano cluster che si immaginava di ottenere già dal primo plot dei dati trasformati con PCA. Tra quelli di $K = 4$, il cluster 3 poteva essere inaspettato in prima analisi. Avendo il grafico delle label originali, vediamo che quel cluster indica in realtà un misto tra Kama e Rosa in zone lontane da dove sono principalmente concentrati i dati. Sarebbe interessante capire se questi dati possono essere considerati outliers per Kama e Rosa, ma non faremo questa analisi.

Notiamo, di nuovo precisando che non è lo scopo della nostra analisi, che i cluster ottenuti con $K = 3$ si avvicinano molto alle label originali e calcolando lo stesso valore percentuale dell'analisi senza PCA otteniamo

[12]:
```
#print(D[:,7],'\n',gmm_pred[3]) #commentato per leggibilità
#dal comando si vede la corrispondenza di cluster->label: 0->2, 1->3, 2->1
```

```
#mappiamo quindi i risultati
res = np.copy(gmm_pred[3])
replacements = {0: 2, 1: 3, 2: 1}
new = np.zeros(res.shape)
for idx, e in replacements.items():
    new[res == idx] = e

#calcolo del numero di elementi uguali
eq = np.where(new == D[:,7], 1., 0.)
ratio_correct = eq.sum()/eq.shape[0]
print('Ratio of cluster elements of same seed kind:',ratio_correct)
```

```
Ratio of cluster elements of same seed kind: 0.9142857142857143
```

La percentuale di cluster con semi dello stesso tipo è ancora molto alta e raggiunge il 91.4%, nonostante la semplificazione del problema tramite PCA.

Tengo a ripetere che noi non siamo a conoscenza delle label originali in un approccio unsupervised, ma la similiarità con le label classificate vuole essere un esempio di come, se applichiamo unsupervised learning su dati che non conosciamo, possiamo ottenere strutture tra i dati che ci permettono di avere delle ottime basi per fare successive analisi. Tengo a precisare anche che, con i cluster ottenuti, siamo in grado di generare nuovi dati, grazie alle gaussiane, che possiamo usare come dati artificiali per allenare algoritmi che richiedono grandi quantitativi di dati come, ad esempio, le reti neurali.

## 2.5 Considerazioni finali

La semplicità e l'espressività del modello finale sono la forza delle GMM. Utilizzare un algoritmo simile al posto dell'usuale $K$-means ha poco prezzo aggiuntivo, può dare ottime indicazioni su dove muoversi nelle fasi successive e ci permette di avere un modello in grado di generare dati plausibili e, si sa, i dati non bastano mai in questo ambito.

# References


[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] Oscar Contreras Carrasco. "Gaussian Mixture Models Explained". `https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95`.

[3] M. Charytanowicz, J. Niewczas, P. Kulczycki, P.A. Kowalski, S. Lukasik, and S. Zak. "A Complete Gradient Clustering Algorithm for Features Analysis of X-ray Images". *Information Technologies in Biomedicine, Ewa Pietka, Jacek Kawa (eds.)*, 2010.

[4] Daniel Foley. "Gaussian Mixture Modelling (GMM)". `https://towardsdatascience.com/gaussian-mixture-modelling-gmm-833c88587c7f`.