Ingegneria Informatica

Multistart k-means

Optimization Methods

Docente:
Fabio Schoen

Studente:
Paula Mihalcea

Indice

1 Introduzione

La presente relazione costituisce una breve descrizione dell'implementazione di un algoritmo di ottimizzazione globale, il **multistart**, nel linguaggio di programmazione Python, e la sua applicazione ad un problema di clustering utilizzando l'algoritmo di ottimizzazione locale **k-means**.

2 Dataset

Tutti i problemi di ottimizzazione necessitano di alcuni dati su cui lavorare, che si tratti di una funzione obiettivo oppure di una raccolta di osservazioni. Nel presente progetto è stato scelto, per semplicità, di generare un dataset di punti casuali da utilizzare come base su cui applicare l'algoritmo; tali punti saranno variamente raggruppati in un numero predeterminato di cluster, per sfruttare k-means al massimo delle sue potenzialità.

In particolare, la funzione `gen_dataset()` restituisce un *dataframe* (libreria *Pandas*) secondo i seguenti parametri specificabili dall'utente:

- `points`: n°di punti da generare in ogni cluster (default: `1000`);
- `features`: n°di caratteristiche da assegnare ad ogni punto (default: `2`);
- `k`: n°di cluster da generare (default: `2`);
- `center_range`: range delle coordinate che verranno casualmente assegnate ai centri dei cluster; determina l'ampiezza dello spazio in cui verrano generati i punti (default: `5000`);
- `scale_low`: deviazione standard minima per la generazione dei punti (default: `500`);
- `scale_high`: deviazione standard massima per la generazione dei punti (default: `1000`);
- `plot`: se posto a `True`, genera e visualizza il grafico del dataset creato, se i suoi punti hanno due caratteristiche (default: `False`).

La funzione consiste in un loop principale che comincia creando un centro e prosegue aggiungendo *points* punti intorno ad esso, secondo una distribuzione normale con i parametri specificati in precedenza (le variabili *scale_low* e *scale_high* determinano il loro grado di sparsità). Creato così un cluster, lo si aggiunge al dataset e si ricomincia il loop per generare quello successivo, per k volte. Infine, se il numero di caratteristiche delle osservazioni è pari a 2 e la variabile `plot` è posta a `True`, allora la funzione plotta il dataset in un grafico bidimensionale e lo visualizza (fig. **??**).

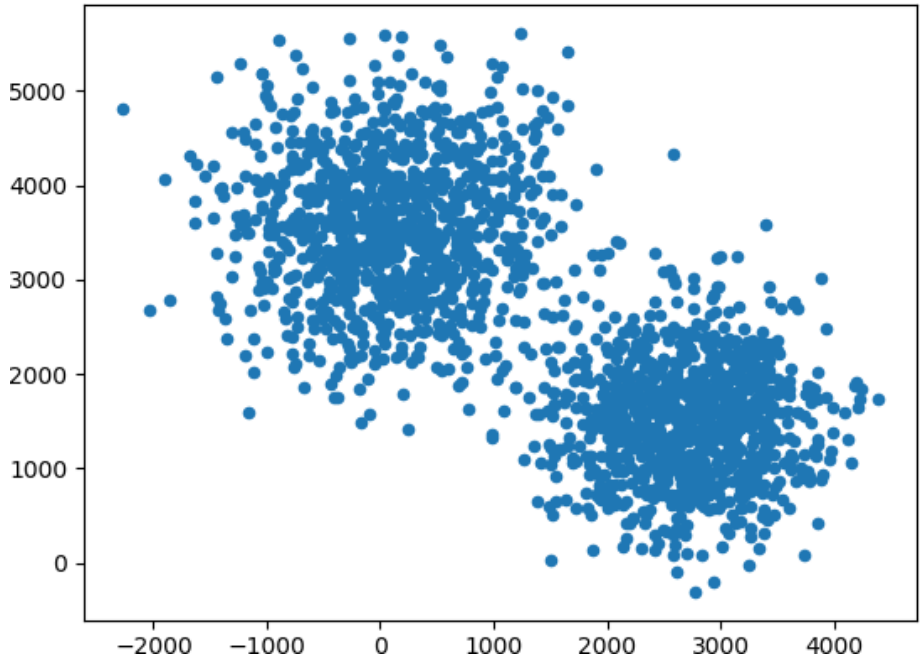

Figura 1: Esempio di dataset generato con `gen_dataset()` (parametri di default).

3 Multistart

L'algoritmo k-means necessita, prima di entrare nel loop principale, di una fase di inizializzazione in cui, secondo qualche criterio, vengano scelti i centroidi iniziali. Ai fini di utilizzare un approccio di ottimizzazione globale, la versione implementata in questo progetto prevede una scelta casuale[1] di k punti entro il range delle coordinate dei centri (`center_range`) sommato alla deviazione standard massima (`scale_high`), secondo l'algoritmo **multistart**.

In particolare, la funzione `gen_rand_centroids(k, features, center_range, scale)` ritorna una matrice di centroidi con queste caratteristiche.

4 K-means

4.1 Inizializzazione

L'algoritmo **k-means**[2] viene dunque inizializzato tramite **multistart**. Prima di procedere con il loop principale, tuttavia, viene aggiunta al dataset una colonna finale

[1] Utilizzando una distribuzione uniforme.

[2] Implementato ad hoc per il progetto, senza l'utilizzo di librerie che lo contenessero già.

contenente il cluster di appartenenza di ogni punto (l'etichetta), calcolato grazie alla funzione `label(dataset, centroids, k, features)`. Essa calcola la **distanza euclidea** di ogni punto da ognuno dei k cluster presenti nel dataset[3], e individua per ognuno quella più breve, determinando in questo modo il cluster a cui assegnarlo.

4.2 Loop principale

A questo punto, avendo tutti i dati necessari, è possibile avviare il loop principale dell'algoritmo k-means. Questo consiste in un ciclo infinito che calcola, per ogni cluster, la posizione media dei centroidi in base ai punti presenti in esso, ed aggiorna poi nel dataset con i dati così ottenuti la colonna delle etichette, invocando nuovamente la funzione `label()`. Le iterazioni si fermano dunque quando i centroidi non vengono più spostati (fig. **??**).

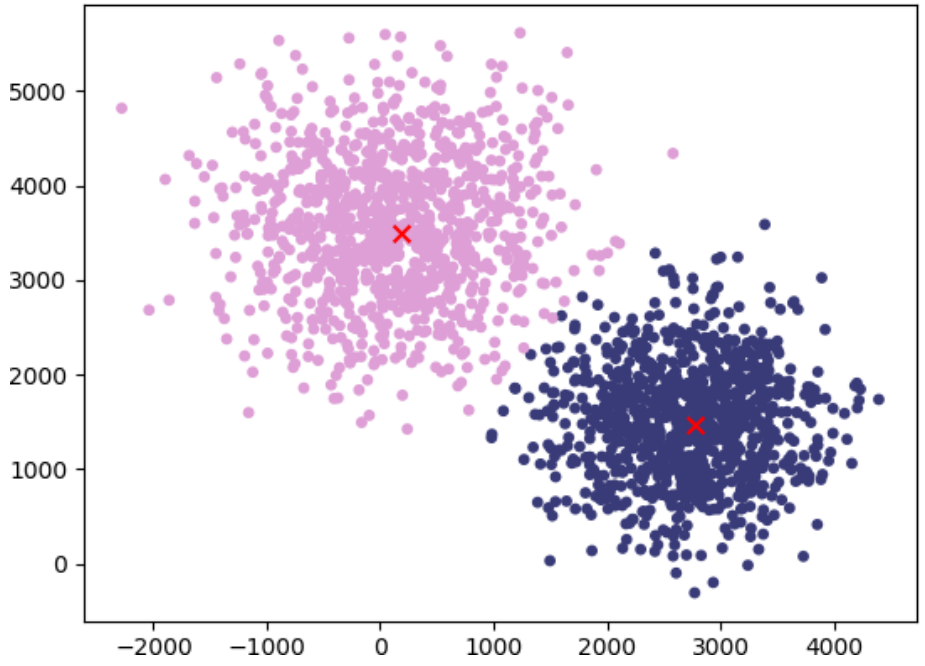

Figura 2: Il dataset di fig. **??** dopo le iterazioni di k-means.

5 Conclusioni

L'algoritmo implementato dimostra un'ottima capacità di clustering dei dati, in particolar modo se la deviazione standard fornita nei parametri iniziali è bassa.

[3] Attraverso la funzione ausiliaria `centroid_distance(dataset, centroids, k)`.

K-means, tuttavia, non è stato ottimizzato per la velocità di esecuzione (essendo stato implementato ad hoc e parzialmente esulante dallo scopo del progetto), per cui risulta assai lento per dataset contenenti un numero totale di osservazioni superiore a 15.000, anche su macchine di fascia alta.

6 Esempi

Seguono alcuni esempi di grafici creati a partire da dataset a due features con 100 punti per cluster e deviazione compresa tra 500 e 1000.

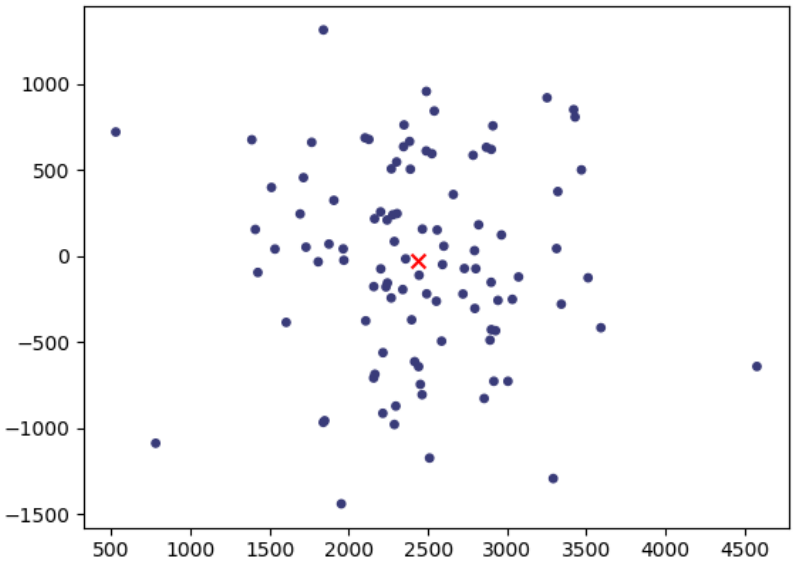

Figura 3: $k = 1$

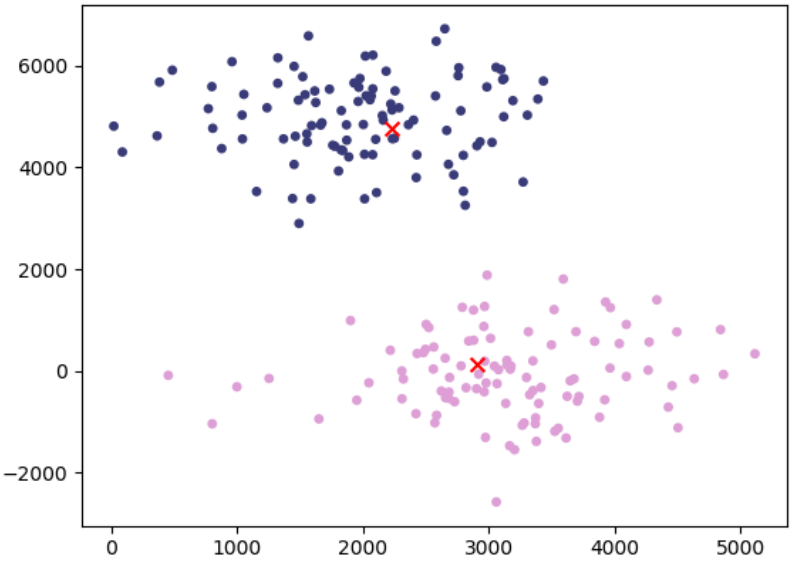

Figura 4: $k = 2$

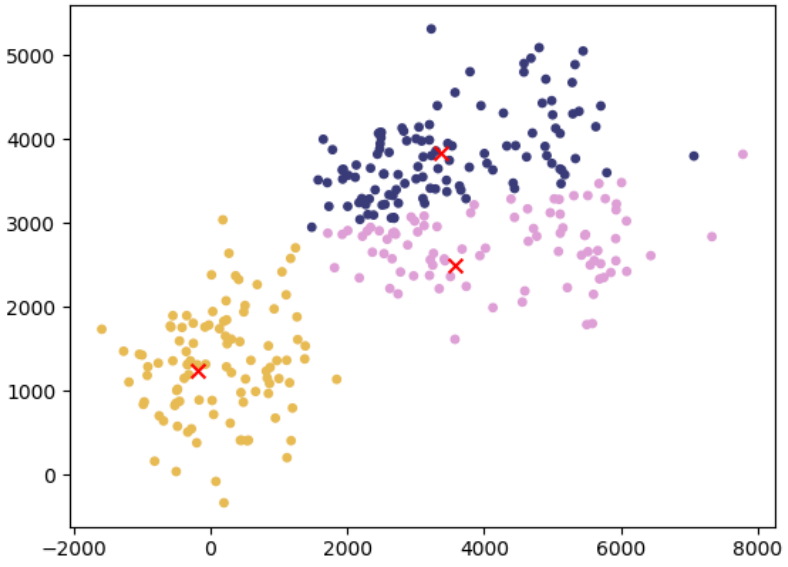

Figura 5: $k = 3$

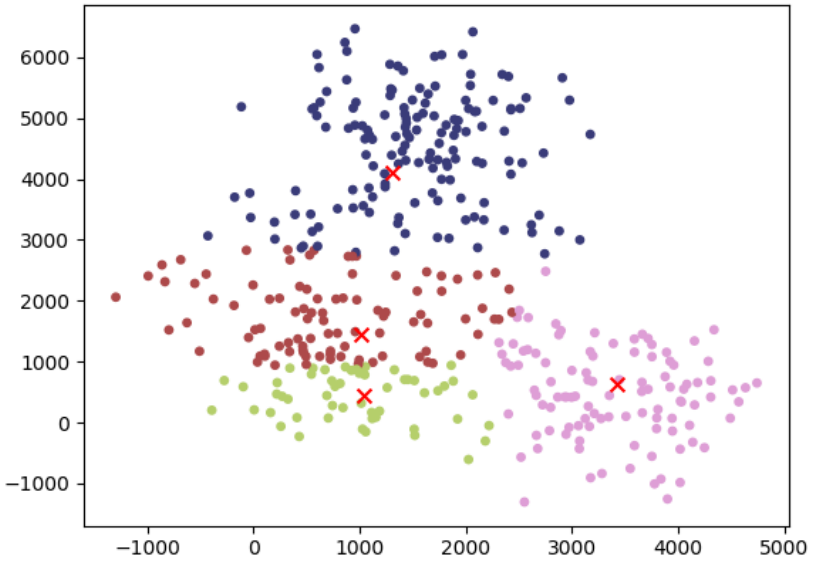

Figura 6: $k = 4$

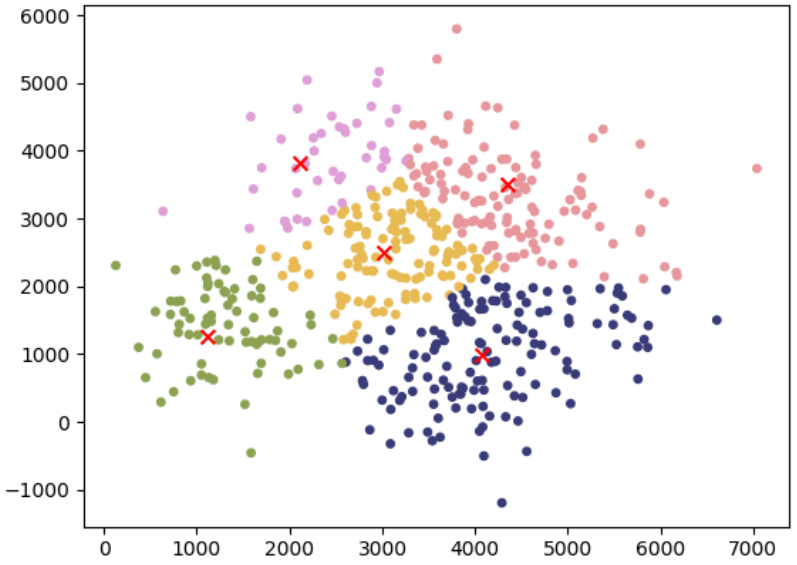

Figura 7: $k = 5$

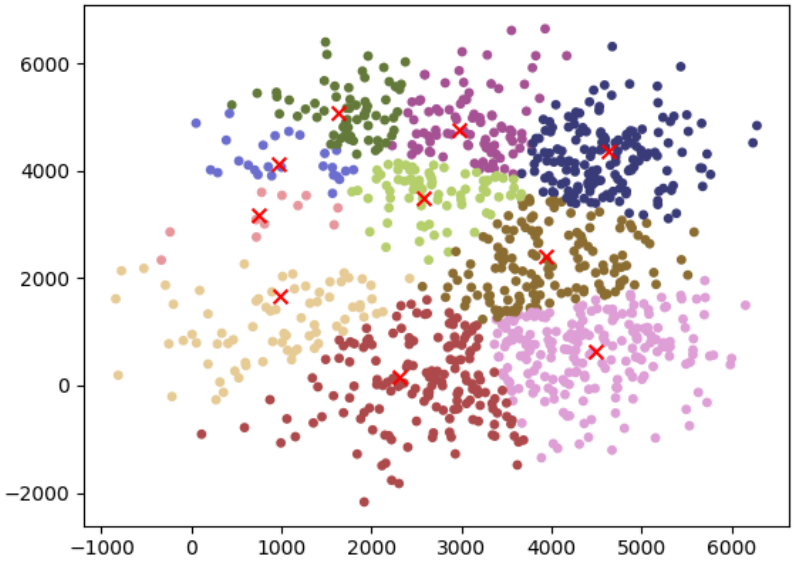

Figura 8: $k = 10$