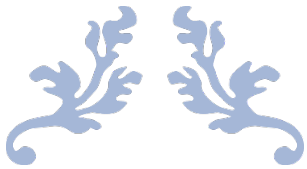

PROGETTO AOCR

Ripetizione Rapida Code

19 GIUGNO 2023

ROMANO RICCARDO
M63001489

IDEA DEL PROGETTO

L’idea del progetto è partita dal presupposto secondo il quale, partendo da un generico Data Set (noi utilizzeremo Moon), trovandoci di fronte ad un problema di Clustering, è possibile risolverlo mediante l’algoritmo del K-Means. Tuttavia quest’ultimo non presenta soluzioni ottimali, dunque si può migliorare mediante la meta-euristica del Tabu Search.

Problema di Clustering

Un problema di clustering è un tipo di problema di apprendimento non supervisionato che coinvolge la suddivisione di un insieme di dati in gruppi omogenei o "**cluster**" in base alle loro caratteristiche comuni. L'obiettivo principale è raggruppare gli elementi del dataset in modo che gli oggetti all'interno dello stesso cluster siano simili tra loro, mentre gli oggetti in cluster diversi siano distinti o dissimili.

Nel contesto del clustering, non si dispone di etichette o classificazioni predefinite per i dati di input. Quindi, il clustering è un processo di esplorazione dei dati per individuare strutture o relazioni nascoste, senza una guida o una supervisione esterna. Si basa sulla somiglianza tra gli oggetti nel dataset, che viene misurata utilizzando una qualche forma di distanza o similarità tra le caratteristiche dei dati.

I problemi di clustering possono essere affrontati in vari domini e hanno diverse applicazioni. Ad esempio, nel campo del marketing, il clustering può essere utilizzato per suddividere i clienti in gruppi omogenei in base ai loro comportamenti di acquisto, consentendo alle aziende di personalizzare le loro strategie di marketing per gruppi specifici.

Esistono diversi algoritmi di clustering, come il k-means, il clustering gerarchico, il DBSCAN (Density-Based Spatial Clustering of Applications with Noise) e molti altri, ognuno con i propri punti di forza e limitazioni. La scelta dell'algoritmo dipenderà dal tipo di dati, dalle dimensioni del dataset, dalle caratteristiche dei cluster desiderati e da altri fattori specifici del problema.

K-MEANS

Il k-means è un algoritmo di clustering ampiamente utilizzato per suddividere un insieme di dati non etichettati in gruppi omogenei. L'obiettivo del k-means è di assegnare ciascun punto dati a uno dei “k” cluster predefiniti, in modo tale che la somma delle distanze quadrate tra i punti dati e i centroidi dei rispettivi cluster sia minimizzata.

Ecco come funziona l'algoritmo k-means:

1) **Inizializzazione**: Si selezionano casualmente “k” centroidi iniziali, che rappresentano i punti nel dataset. Tipicamente, questi centroidi iniziali vengono scelti in modo casuale o utilizzando una strategia basata su euristiche.

2) **Assegnazione**: Per ciascun punto dati nel dataset, viene calcolata la distanza tra il punto e tutti i centroidi. Il punto viene quindi assegnato al cluster il cui centroide è più vicino a esso in base alla distanza. In altre parole, viene determinato il cluster più vicino per ciascun punto.
3) **Aggiornamento**: Dopo aver assegnato tutti i punti ai cluster, i centroidi vengono aggiornati calcolando la media dei punti assegnati a ciascun cluster. Questo calcolo determina il nuovo centroide di ciascun cluster.
4) **Ripetizione**: I passi 2 e 3 vengono ripetuti fino a quando non si verifica una condizione di convergenza. La condizione di convergenza può essere soddisfatta quando i centroidi non cambiano più o quando la somma delle distanze quadrate tra i punti e i centroidi non diminuisce significativamente.
5) **Risultato**: Alla fine dell'algoritmo, i punti dati saranno assegnati in modo definitivo ai rispettivi cluster. I centroidi rappresentano i punti centrali di ciascun cluster.

L'algoritmo k-means può produrre risultati soddisfacenti, ma può anche essere influenzato dalla scelta iniziale dei centroidi. A seconda dell'inizializzazione, l'algoritmo può convergere a soluzioni diverse, che potrebbero essere ottimi locali invece di un ottimo globale. Pertanto, in pratica, l'algoritmo k-means viene spesso eseguito più volte con diverse inizializzazioni per trovare la migliore soluzione.

TABU SEARCH

Tabu Search è un algoritmo di ricerca meta-euristica utilizzato per risolvere problemi di ottimizzazione combinatoria. L'obiettivo del Tabu Search è trovare una soluzione di buona qualità in uno spazio di ricerca molto vasto e complesso.

A differenza di altri algoritmi di ricerca locale, come la ricerca locale steepest descent o l'algoritmo di Hill Climbing, il Tabu Search permette di esplorare anche soluzioni che potrebbero peggiorare temporaneamente il valore della funzione obiettivo. Questo è reso possibile tramite l'uso di una lista tabù, che tiene traccia delle mosse già effettuate e impedisce di ritornare a soluzioni precedentemente visitate.

L'algoritmo Tabu Search opera in modo iterativo, seguendo un processo simile al seguente:

1. **Inizializzazione**: Si parte da una soluzione iniziale o si genera casualmente una soluzione nel dominio del problema.
2. **Valutazione**: Viene calcolato il valore della funzione obiettivo per la soluzione corrente.
3. **Generazione delle mosse**: Vengono generate una o più mosse o modifiche alla soluzione corrente per esplorare il vicinato della soluzione corrente.
4. **Valutazione delle mosse**: Per ciascuna mossa generata, viene calcolato il valore della funzione obiettivo per la nuova soluzione proposta.
5. **Selezione delle mosse**: Si selezionano una o più mosse considerando la loro qualità, ma evitando quelle che sono tabù, ovvero quelle che porterebbero a soluzioni già visitate recentemente.
6. **Aggiornamento della soluzione corrente**: Si applica la mossa selezionata per ottenere una nuova soluzione corrente.
7. **Aggiornamento della lista tabù**: Si aggiornano le informazioni nella lista tabù, registrando la mossa appena eseguita per impedire di ripeterla.
8. **Convergenza**: Si verifica se una condizione di convergenza è soddisfatta. Ad esempio, potrebbe essere un numero massimo di iterazioni raggiunte o una condizione di stop basata sul valore della funzione obiettivo.
9. **Ripetizione**: I passi dal 2 all'8 vengono ripetuti fino a quando non si raggiunge la condizione di convergenza.

10. **Risultato**: Alla fine dell'algoritmo, si ottiene la soluzione migliore trovata durante l'esecuzione, che può essere considerata come una soluzione approssimata al problema di ottimizzazione.

L'algoritmo Tabu Search è flessibile e può essere adattato a diversi problemi di ottimizzazione combinatoria. La lista tabù può essere implementata in vari modi, tenendo traccia di mosse specifiche o di attributi delle soluzioni. Inoltre, possono essere introdotte strategie di aspirazione che consentono di superare alcune restrizioni tabù in casi eccezionali. Queste caratteristiche consentono all'algoritmo Tabu Search di esplorare lo spazio delle soluzioni in modo efficace e di superare ottimi locali.

F1-Score

L'F1-score è una misura comune dell'accuratezza di un modello di classificazione. È una metrica che combina la precisione e il richiamo (recall) in un unico valore, fornendo una valutazione complessiva delle prestazioni del modello.

La precisione misura la frazione di istanze positive che sono correttamente identificate dal modello rispetto a tutte le istanze classificate come positive. In altre parole, la precisione indica quanto un modello è preciso nel predire correttamente le istanze positive.

Il richiamo, o recall, invece, misura la frazione di istanze positive correttamente identificate dal modello rispetto a tutte le istanze effettivamente positive. Il richiamo indica quanto il modello è in grado di recuperare correttamente le istanze positive.

L'F1-score è definito come la media armonica di precisione e richiamo ed è calcolato utilizzando la seguente formula:

$$F1\text{-}Score = 2 \times \frac{(precisione \times richiamo)}{(precisione + richiamo)}$$

La media armonica tiene conto dei valori bassi sia di precisione che di richiamo, attribuendo un peso maggiore alle prestazioni peggiori. Di conseguenza, l'F1-score tende ad essere più basso se sia la precisione che il richiamo sono bassi.

L'F1-score varia tra 0 e 1, dove 1 indica la massima precisione e richiamo, mentre 0 indica la peggiore performance. Un valore più alto di F1-score indica una migliore capacità del modello di bilanciare precisione e richiamo.

L'F1-score è particolarmente utile quando il dataset è sbilanciato, cioè quando una classe è rappresentata da un numero significativamente maggiore o minore rispetto all'altra. In questi casi, l'F1-score può fornire una valutazione più accurata delle prestazioni del modello rispetto alla semplice precisione o richiamo da soli.

In generale, l'F1-score è una metrica comune per valutare i modelli di classificazione, soprattutto quando le classi sono sbilanciate o quando sia la precisione che il richiamo sono importanti per il problema specifico.

CODICE

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_moons

from sklearn.cluster import KMeans

from sklearn.metrics import f1_score
```

In queste righe di codice vengono importate le librerie necessarie. numpy è utilizzata per operazioni matematiche e di algebra lineare, matplotlib.pyplot per la visualizzazione dei dati e dei grafici, make_moons per generare un dataset sintetico a forma di luna, KMeans per l'algoritmo di clustering K-means e f1_score per calcolare la misura F1-score utilizzata per valutare la qualità del clustering.

```
def tabu_search_clustering(data, k, num_iterations, tabu_size, labels_kmeans):

    # Inizializzazione con l'algoritmo k-means

    best_solution = labels_kmeans

    best_cost = f1_score(ground_truth_labels, best_solution, average='weighted')



    # Inizializzazione della lista TABU

    tabu_list = []



    for _ in range(num_iterations):

        # Generazione di una nuova soluzione vicina

        new_solution = best_solution.copy()

        i, j = np.random.choice(range(len(data)), size=2, replace=False)

        new_solution[i] = new_solution[j]

        new_solution[j] = best_solution[i]



        # Calcolo dello score della nuova soluzione

        new_cost = f1_score(ground_truth_labels, new_solution, average='weighted')
```

```
# Aggiornamento della soluzione migliore se la nuova soluzione è migliore

        if new_cost > best_cost and not any(np.array_equal(new_solution, tabu_solution) for
tabu_solution in tabu_list):

            best_solution = new_solution

            best_cost = new_cost



        # Aggiunta della nuova soluzione alla lista TABU

        tabu_list.append(new_solution)



        # Rimozione delle soluzioni più vecchie dalla lista TABU

        if len(tabu_list) > tabu_size:

            tabu_list.pop(0)



    return best_solution, best_cost
```

Questa parte del codice definisce una funzione tabu_search_clustering che implementa l'algoritmo di ricerca Tabu per il clustering. Prende in input il dataset data, il numero di cluster k, il numero di iterazioni num_iterations, la dimensione della lista TABU tabu_size e le etichette calcolate dall'algoritmo K-means labels_kmeans.

L'algoritmo inizia inizializzando la soluzione migliore con le etichette K-means e calcola il relativo costo utilizzando la misura F1-score. Successivamente, inizializza una lista TABU vuota.

All'interno di un ciclo for che esegue num_iterations iterazioni, viene generata una nuova soluzione vicina scambiando casualmente le etichette di due punti del dataset. Viene calcolato il costo della nuova soluzione utilizzando la misura F1-score.

Se la nuova soluzione ha un costo migliore della soluzione attuale e non è presente nella lista TABU, allora la soluzione migliore viene aggiornata. La nuova soluzione viene anche aggiunta alla lista TABU. Se la lista TABU supera la dimensione tabu_size, la soluzione più vecchia viene rimossa.

Alla fine delle iterazioni, la funzione restituisce la migliore soluzione trovata e il suo costo.

```
X, ground_truth_labels = make_moons(n_samples=2000, noise=0.03, random_state=0)
```

Qui viene generato un dataset sintetico a forma di luna chiamato "X" utilizzando la funzione make_moons dalla libreria scikit-learn. Viene specificato il numero di campioni n_samples, il livello di rumore noise e lo stato casuale random_state. Vengono inoltre assegnate le etichette di verità fondamentale del dataset a ground_truth_labels.

```
kmeans = KMeans(n_clusters=2)

kmeans.fit(X)

labels_kmeans = kmeans.predict(X)

centroids_kmeans = kmeans.cluster_centers_
```

In queste righe di codice, viene creato un oggetto kmeans utilizzando l'algoritmo K-means dalla libreria scikit-learn. Viene specificato il numero di cluster n_clusters (in questo caso 2) e viene addestrato sul dataset "X". Successivamente, vengono ottenute le etichette predette per il dataset utilizzando il metodo predict e i centroidi dei cluster utilizzando l'attributo cluster_centers.

```
num_iterations = 50000

tabu_size = 100
```

Qui vengono definiti il numero di iterazioni num_iterations e la dimensione della lista TABU tabu_size.

```
best_solution, best_cost = tabu_search_clustering(X, 2, num_iterations, tabu_size, labels_kmeans)
```

Viene chiamata la funzione tabu_search_clustering passando il dataset "X", il numero di cluster (2), il numero di iterazioni e la dimensione della lista TABU. La funzione restituisce la migliore soluzione trovata best_solution e il suo costo best_cost.

```
f1_kmeans = f1_score(ground_truth_labels, kmeans.predict(X))
```

Viene calcolato l'F1-score tra le etichette di verità fondamentale ground_truth_labels e le etichette predette dall'algoritmo K-means per il dataset "X".

```
f1_tabu_search = f1_score(ground_truth_labels, best_solution)
```

Viene calcolato l'F1-score tra le etichette di verità fondamentale ground_truth_labels e la migliore soluzione trovata dall'algoritmo di ricerca Tabu.

```
fig, axs = plt.subplots(1, 3, figsize=(15, 8))
```

Qui vengono creati tre assi per i grafici, che verranno disposti in una griglia 1x3, con una dimensione di figura di 15x8 pollici.

```
axs[0].scatter(X[:, 0], X[:, 1], c=kmeans.predict(X), cmap='viridis')

axs[0].scatter(centroids_kmeans[:, 0], centroids_kmeans[:, 1], marker='x', color='red', s=100,
linewidths=2)

axs[0].set_xlabel('X')

axs[0].set_ylabel('Y')

axs[0].set_title('Clustering K-means sul dataset "Moon" (F1-score: {:.2f})'.format(f1_kmeans))
```

In questa parte del codice viene creato il primo grafico. Vengono tracciati i punti del dataset "X" utilizzando le coordinate X e Y come coordinate dei punti. I colori dei punti sono assegnati in base alle etichette predette dall'algoritmo K-means. I centroidi dei cluster vengono rappresentati come 'x' di colore rosso. Vengono impostati i nomi degli assi, il titolo del grafico che include l'F1-score calcolato per K-means. [UGUALI PER METODO TABU-SEARCH E GROUND TRUTH]

```
plt.tight_layout()

plt.show()
```

Infine, vengono impostati i layout dei grafici in modo che non si sovrappongano e vengono visualizzati i grafici.