AA-2022/23 **Algoritmo Mean Shift Clustering**

Adalberto Lombardi
E-mail address
adalberto.lombardi@stud.unifi.it

Abstract

Il presente documento descrive l'algoritmo di mean shift clustering ed alcune sue implementazioni una sequenziale ed una parallela. Dopo un'introduzione con la definizione degli obiettivi si passa a descriverne la struttura e le fasi di cui è composto. Nella sezione dell'implementazione vengono descritti gli algoritmi realizzati con le rispettive logiche di funzionamento. Infine nella sezione dei risultati vengono confrontati gli output ottenuti con le diverse tecniche variando la dimensione del dataset in ingresso ed in quella parallela anche il numero di thread.

1. Introduzione

Il mean shift clustering è un potente algoritmo di clusterizzazione utilizzato nell'apprendimento non supervisionato, ma a differenza di altri non fa alcuna ipotesi quindi è detto non parametrico.

L'algoritmo assegna i punti ai cluster in modo iterativo spostandoli verso la più alta densità di punti dati, ossia identificando il centroide del cluster, in questo caso non è necessario specificare il numero di gruppi in anticipo perché sarà determinato dall'algoritmo in base ai dati.

L'obiettivo è di trovare i picchi o le zone di densità nei dati, che corrispondono ai cluster.

2. Struttura

ll Mean Shift Clustering è un algoritmo di *hill climbing*, una tecnica di ottimizzazione matematica che appartiene alla famiglia della ricerca locale, ossia si muove continuamente nella direzione di maggiore elevazione, terminando l'esecuzione al raggiungimento del picco massimo o quando nessun valore più alto è presente.

Il suo funzionamento può essere sintetizzato nei seguenti passi:

- Inizializzazione del centroide per ogni punto nel set di dati;
- Per ogni centroide, calcolare la media ponderata di tutti i punti nel dataset, pesando ogni punto in base alla sua distanza dal centroide;
- Spostare il centroide alla media ponderata calcolata;
- Ripetere i passi precedenti fino a quando il centroide non si sposta più di una certa soglia predefinita chiamata "picco";
- Assegnare ogni punto al centroide più vicino.

L'algoritmo di mean-shift clustering può essere utilizzato con diversi tipi di kernel per calcolare i pesi e la distanza euclidea può essere sostituita da una metrica di distanza personalizzata se necessario. Inoltre ho diverse proprietà interessanti tra cui la robustezza ai valori anomali e la capacità di gestire cluster di forme e dimensioni arbitrarie.

3. Implementazioni

Sono state implementate, come richiesto, due versioni dell'algoritmo: una sequenziale ed una parallela. Nella versione sequenziale sono utilizzate la variabile *radius* che imposta la larghezza di banda, la lista *labels* per tenere traccia delle etichette di clustering, la lista *new_centroids* per tracciare i nuovi centroidi ed infine la lista *within_radius* che contiene i punti all'interno del raggio di banda *radius*. Il programma utilizza una funzione *euclidean_distance* per calcolare la distanza euclidea tra due punti, una funzione *mean_shift* per eseguire il clustering. Il tempo di elaborazione viene misurato con la funzione *time.time*(). La variabile *start_time* contiene il momento di avvio dell'algoritmo, mentre la variabile *end_time* quello in cui termina.

Nel codice principale vengono letti i dati in input da un file esterno e chiamata la funzione *mean_shift*. All'interno di questa vi è un ciclo *while* in cui viene calcolata la distanza euclidea tra il punto osservato ed il centroide e se il punto è all'interno del raggio di *radius* lo aggiunge alla lista *within_radius*, a questo punto vengono calcolati i nuovi valori dei centroidi. Il processo itera fintantoché il calcolo dei nuovi centroidi non subisce più variazioni. Si può notare come il valore di *radius* può influenzare significativamente i risultati del clustering. Il tempo di

elaborazione viene calcolato tramite differenza tra i valori *end_time* e *start_time*. Il tempo di elaborazione viene mostrato a video.

Nella versione parallela è stata creata una funzione *mean_shift_async* che esegue il mean-shift clustering su un singolo blocco di dati in modo asincrono. Quindi con la funzione *run_mean_shift_parallel* si suddividono i dati in *n_jobs* blocchi, per cui viene calcolata la funzione *MeanShift* importata dal modulo *sklearn.cluster*, quindi viene creato una task per ogni blocco, avviando così l'esecuzione parallela. Infine, con la funzione *asyncio.gather* è possibile unire i risultati dei singoli blocchi.

4. Risultati

Gli algoritmi sono stati testati variando il dataset in input partendo da 1.000 osservazione, quindi 10.000 ed infine 20.000 ed inoltre, all'interno dell'esecuzione parallela sono state fatte differenti prove variando il numero di processi partendo da 2 fino ad un massimo di 12. Nelle prove effettuate ciò che è variato in maniera significativa all'aumentare della dimensione dei dati in ingresso è stato il tempo di esecuzione dell'algoritmo *mean_shift*(), poiché, già con una matrice di dati con 1.000 osservazioni vi è una differenza di 12,6327 secondi tra l'algoritmo sequenziale e quello parallelo che sfrutta la libreria AsyncIO, con 12 thread, mentre nel caso di minimo parallelismo, ossia 2 processi, la differenza non appare così significativa dato che vi è uno scarto di appena 3,2023 secondi. La differenza di performances, tra le due strategie implementative, si è resa ancor più evidente utilizzando un dataset con 10.000 e 20.000 osservazioni in cui l'elaborazione sequenziale ha impiegato un tempo elevato per completare l'elaborazione rispettivamente oltre 37 minuti nel primo caso e circa 152 minuti nel secondo caso, contro i risultati "peggiori" della strategia parallela, ossia quelli con solo 2 processi, che hanno impiegato rispettivamente circa 5 minuti nel caso con 10.000 osservazioni e 15 minuti nel caso con 20.000 osservazioni. Infine lo speedup più evidente vi è con il maggior grado di parallelizzazione, ossia 12 thread in cui si ha un tempo di elaborazione rispettivamente di circa 2 minuti nel primo caso e 6 minuti nel secondo.

Di seguito una tabella con le prove effettuate:

	n. thread	1.000	10.000	20.000
Sequenziale		16.93860	2259.1824	9114.0300
Parallelo	2	13.7363	292.8279	915.0976
	4	8.2244	190.4728	577.9066
	6	6.1631	203.8658	548.5062
	8	5.1105	162.5257	478.0969
	10	4.2964	144.2527	439.6670
	12	4.3059	140.9795	399.7839

Il programma con AsyncIO è più efficiente in termini di risorse computazionali, poiché sfrutta la parallelizzazione per eseguire il clustering in modo asincrono su più worker, consentendo una maggiore efficienza nell'elaborazione di insiemi di dati di grandi dimensioni. Inoltre, l'utilizzo di AsyncIO permette di eseguire altre attività durante l'attesa delle operazioni di I/O, migliorando l'utilizzo delle risorse del sistema, tuttavia ciò potrebbe richiedere più tempo per la preparazione e la gestione dei task asincroni. Inoltre è importante notare che la velocità di esecuzione parallela dipende dalla disponibilità di risorse del sistema (ad esempio, il numero di core della CPU) e dalla dimensione dei dati da elaborare. In generale, l'utilizzo di AsyncIO può migliorare le prestazioni dell'algoritmo mean-shift clustering, ma la scelta del numero di processi paralleli da utilizzare dipende dal singolo caso e può richiedere un po' di sperimentazione.
Il programma parallelo dovrebbe essere preferito per eseguire il clustering su insiemi di dati di grandi dimensioni e in situazioni in cui è necessaria una maggiore efficienza computazionale che di fatto né migliorano lo speedup. Il programma sequenziale, d'altra parte, può essere una scelta migliore per insiemi di dati più piccoli o per situazioni in cui la semplicità e la facilità di utilizzo sono prioritari.