AA-2022/23 **Algoritmo K-means clustering**

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

Abstract

Il presente documento descrive l'algoritmo K-means 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. Quindi si affronta il tema della scelta del numero di classi da impostare e si descrive la "tecnica del gomito" ed infine vengono indicati i casi d'uso. 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 il numero di cluster e la dimensione del dataset in ingresso.

1. Introduzione

Il k-means clustering è un algoritmo non supervisionato utile per la segmentazione dei dati. L'obiettivo è quello di raggruppare le osservazioni simili per scoprire pattern nascosti dividendo i dati in cluster.

2. Struttura

L'algoritmo prevede la creazione di un punto di riferimento denominato centroide per un numero desiderato di classi e quindi assegnando i punti dati ai cluster di classi in base al punto di riferimento più vicino. Prima di tutto si definiscono i cluster che sono solo gruppi di elementi ed i raggruppamenti consistono semplicemente nell'associare elementi in quei gruppi, pertanto gli algoritmi di clustering mirano a fare due cose:

- Assicurati che tutti i punti dati in un cluster siano il più simili possibile tra loro;
- Assicurati che tutti i punti dati nei diversi cluster siano il più possibile dissimili tra loro.

In generale gli algoritmi di clustering raggruppano gli elementi in base a una metrica di somiglianza, questo viene spesso identificato trovando il "centroide" dei diversi gruppi possibili nel dataset. Esistono diversi algoritmi di clustering, ma l'obiettivo è lo stesso determinare i gruppi intrinseci ad un insieme di dati.

Il K-Means Clustering è uno dei tipi di algoritmi più comunemente usati e funziona in base alla quantizzazione vettoriale ossia viene identificato un punto nello spazio selezionato come origine, quindi i vettori vengono disegnati dall'origine a tutti i punti dati in input. In generale, il clustering K-means può essere suddiviso nelle seguenti fasi:

1. Posizionare tutte le istanze in sottoinsiemi, dove il numero di sottoinsiemi è uguale a K;
2. Trovare il punto/centroide medio delle partizioni del cluster appena create;
3. Sulla base di questi centroidi, assegna ogni punto a un cluster specifico;
4. Calcolare le distanze da ogni punto ai centroidi e assegnare punti ai cluster in cui la distanza dal centroide è la minima;
5. Dopo che i punti sono stati assegnati ai cluster, trovare il nuovo centroide dei cluster.

Questi passaggi vengono ripetuti fino al termine del processo di formazione.

In alternativa, dopo che i centroidi sono stati posizionati, si può concepire il clustering di K-means come uno scambio avanti e indietro tra due diverse fasi: l'etichettatura dei punti dati e l'aggiornamento dei centroidi. Nella fase di etichettatura del punto dati, ad ogni punto dati viene assegnata un'etichetta che lo colloca nel cluster appartenente al centroide più vicino, questo viene in genere determinato utilizzando la distanza euclidea al quadrato, sebbene sia possibile utilizzare altre metriche di distanza a seconda del tipo di dati inseriti nell'algoritmo di clustering. Mentre nella fase di aggiornamento del centroide questo viene calcolato trovando la distanza media tra tutti i punti dati attualmente contenuti in un cluster.

Una decisione iniziale molto importante riguarda la scelta del numero di classi K da considerare, visto che tale dato non è noto a priori, vi sono diverse tecniche che

permettono una sua determinazione. Una tecnica per selezionare il numero dei gruppi da definire è chiamata “tecnica del gomito”, questa consiste nell’esecuzione di un algoritmo di clustering K-means per un intervallo di diversi valori K e nell’utilizzo di una metrica di precisione, tipicamente la somma dell’errore quadratico, per determinare quali valori di K danno i risultati migliori. La somma dell’errore quadratico viene calcolata con la distanza media tra il centroide di un cluster e i punti dati in quel cluster.

Il clustering K-means può essere utilizzato in modo sicuro in qualsiasi situazione in cui i punti dati possono essere segmentati in gruppi/classi distinti. Di seguito sono riportati alcuni esempi di casi d’uso comuni come ad esempio nella classificazione dei documenti, raggruppando i documenti in base a funzionalità come argomenti, tag, utilizzo delle parole, metadati e altre funzionalità del documento; potrebbe anche essere utilizzato per classificare gli utenti come bot o meno in base a modelli di attività come post e commenti. Inoltre può essere utilizzato anche per mettere le persone in gruppi in base ai livelli di preoccupazione durante il monitoraggio della loro salute, in base a caratteristiche come comorbidità, età, anamnesi del paziente; altresì può essere utilizzato per la creazione di sistemi di “raccomandazione” come ad esempio nello streaming video dove gli utenti possono essere raggruppati in base a modelli di visualizzazione e contenuti simili consigliati. Infine potrebbe essere utilizzato per attività di rilevamento di anomalie, evidenziando potenziali casi di frode o articoli difettosi.

3. Implementazioni

Sono state implementate, come richiesto, due versioni dell'algoritmo: una sequenziale ed una parallela. In questi algoritmi, il numero di cluster è impostato preventivamente tramite la variabile k mentre il vettore data contiene i punti da clusterizzare, i vettori centroids e clusters sono utilizzati per contenere rispettivamente i centroidi e l'assegnazione di ogni punto ad un cluster; la funzione kMeansClustering() viene chiamata per eseguire l'algoritmo K-means clustering sui dati. Il tempo di elaborazione è misurato utilizzando la funzione clock della libreria ctime, la variabile start contiene il numero di cicli di clock al momento dell'avvio dell'algoritmo, mentre la variabile end contiene il numero di cicli di clock al termine dell'algoritmo.

Inizialmente i centroidi sono assegnati ai primi k punti del vettore data mentre la funzione distance calcola la distanza euclidea tra due punti. Infine il ciclo while esegue l'algoritmo K-means finché i centroidi non convergono o raggiungono un numero massimo di iterazioni, l'assegnazione dei cluster ai punti avviene nel primo ciclo, mentre l'aggiornamento dei centroidi avviene nel secondo ciclo. Il tempo di elaborazione viene calcolato tramite differenza tra i valori end e start e dividendo il risultato per il valore di CLOCKS_PER_SEC, che indica il numero di cicli di clock al secondo.

L'esecuzione parallela inizia con la creazione dei centroidi in modo casuale e l'assegnazione di ogni punto al cluster più vicino, entrambi eseguiti all'interno di un pragma omp parallel for, questa istruzione è utilizzata per parallelizzare i cicli for che eseguono le operazioni su vettori. La successiva ripetizione fino alla convergenza del ciclo principale, dove vengono calcolati i nuovi centroidi e assegnati i punti ai cluster più vicini, viene anch'essa eseguita in parallelo. Il calcolo dei nuovi centroidi utilizza un vettore di contatori per mantenere il numero di punti assegnati a ciascun cluster e, per questo, richiede una sezione critica in cui ogni thread deve eseguire l'incremento del contatore in modo atomico. Inoltre, per aggiornare i centroidi in modo corretto, l'operazione deve essere atomica. La motivazione principale delle scelte implementative, per migliorare lo speedup, è di parallelizzare i cicli for e utilizzare gli statement atomici e criticali per gestire i conflitti tra i thread. L'algoritmo K-means può richiedere molto tempo per eseguire su grandi dataset, e parallelizzarlo può migliorare significativamente le prestazioni.

L'esecuzione parallela inizia subito dopo l'inizializzazione dei centroidi e l'assegnazione dei punti ai cluster più vicini e termina quando il ciclo principale termina e il programma esce dalla funzione kMeansClustering.

4. Risultati

Gli algoritmi sono stati testati variando il numero di cluster impostati rispettivamente con 2, 5 e 10 gruppi ed il dataset in ingresso partendo da 10.000, quindi 100.000 ed infine 1.000.000 osservazioni. Nelle prove effettuate ciò che è variato in maniera significativa all'aumentare della dimensione del dataset in ingresso è stato il tempo di esecuzione dell’algoritmo kMeansClustering(), poiché, già con una matrice di dati in input di 10.000 osservazioni, impostando 10 classi nel caso di struttura sequenziale il tempo di elaborazione è stato di 0,460282 secondi mentre nel caso di implementazione parallela utilizzando la libreria OpenMP il tempo di esecuzione è sceso a 0,105214 secondi, questa differenza si è accentuata utilizzando un dataset con 100.000 punti in cui l’elaborazione sequenziale è stata effettuata in 13,8539 secondi mentre quella parallela di appena 5,36158. Tale differenza si è accentuata con un campione di 1.000.000 osservazioni. Di seguito una tabella con le prove effettuate:

classi \ osservazioni	10.000		100.000		1.000.000	
	Sequenziale	Parallelo	Sequenziale	Parallelo	Sequenziale	Parallelo
2	0,11478	0,02606	1,64101	0,72723	33,38240	26,79750
5	0,14901	0,05453	4,10492	3,55278	72,75070	61,51660
10	0,46028	0,10521	13,85390	5,36158	138,81700	107,15100

Tabella 1: Tabella dei risultati

In conclusione, il programma parallelo utilizzando la libreria OpenMP per implementare il k-means clustering risulta avere prestazioni migliori rispetto al programma sequenziale quando si esegue su una piattaforma hardware multicore, ciò perché OpenMP consente di eseguire il lavoro su più thread, sfruttando i core della CPU in parallelo, consentendo una maggiore efficienza computazionale. Tuttavia, la differenza di prestazioni dipenderà da vari fattori, come l'efficienza dell'implementazione parallela, il numero di core disponibili sulla piattaforma hardware e le dimensioni del dataset. Se il numero di classi preimpostate è relativamente piccolo rispetto alle dimensioni del dataset, non sembra esserci una differenza così marcata di prestazioni tra le due implementazioni. Invece con l'aumentare della dimensione del dataset l'implementazione parallela risulta avere uno speedup marcato rispetto alla versione sequenziale.