Architetture e Programmazione dei Sistemi di Elaborazione
Progetto a.a. 2019/20

"K-D Tree and NIPALS Algorithm"
in linguaggio assembly x86-32+SSE e x86-64+AVX

Fabrizio Angiulli Fabio Fassetti

1 Decrizione del problema

Il *k-d-Tree* (K-Dimensional Tree) è un albero binario di ricerca multi-dimensionale. Dato un dataset $\mathcal{D}$, ossia un insieme di punti a k dimensioni, costruire un *k-d-Tree* equivale a costruire un indice del dataset che consente di rispondere efficientemente alla seguente ricerca:

- dato un punto Q e un raggio r, quali sono i punti di $\mathcal{D}$ che distano da Q meno di r?

È stato dimostrato che quando k assume valori molto elevati le tecniche di indicizzazione risultano meno efficaci. Una soluzione è rappresentata dal ridurre le dimensioni del dataset di partenza, considerando solo le dimensioni *principali* attraverso una tecnica di *PCA*, *Principal Component Analysis*.
Dato un dataset $\mathcal{D}$ di dimensione $n \times k$, il problema consiste quindi nell'eventuale trasformazione di $\mathcal{D}$ in un dataset h-dimensionale con $h \leq k$ e nel costruire un *k-d-Tree* sul dataset eventualmente trasformato.

Nozioni preliminari

Una regione k dimensionale $\mathcal{H}$ è identificata da un vettore di k coppie

$$\mathcal{H} = [(d_0^{\min}, d_0^{\max}), \ldots, (d_{k-1}^{\min}, d_{k-1}^{\max})]$$

che specificano per ogni dimensione $j \in [0, k)$ i limiti della regione. Nel seguito, $\mathcal{H}_j^{\min}$ identifica il valore $d_j^{\min}$, mentre $\mathcal{H}_j^{\max}$ identifica il valore $d_j^{\max}$.

Sia $\mathcal{D}$ un dataset, ossia un insieme di punti a k-dimensioni, viene definita *regione indotta* da $\mathcal{D}$ la regione:

$$\mathcal{H}(\mathcal{D}) = [(\min D_0, \max D_0), \ldots, (\min D_{k-1}, \max D_{k-1})],$$

dove $\mathcal{D}_j$ indica la j-esima colonna del dataset $\mathcal{D}$, $\min D_j$ indica il minimo della j-esima colonna del dataset $\mathcal{D}$ e $\max D_j$ indica il massimo della j-esima colonna del dataset $\mathcal{D}$.

Function BuildTree($\mathcal{D}$, l)

Input: un dataset $\mathcal{D}$, il livello corrente l
Output: il nodo radice del *k-d-Tree* associato a $\mathcal{D}$

```
1  begin
2      if D = ∅ then
3          return null
4      sia c = l%k la dimensione di taglio
5      sia P il punto mediano lungo la dimensione c
6      D_1 = {P_i ∈ D : P_i[c] < P[c]}
7      D_2 = {P_i ∈ D \ {P} : P_i[c] ≥ P[c]}
8      sia n un nuovo nodo tale che:
9      - point(n) = P
10     - leftchild(n) = BuildTree(D_1, l + 1)
11     - rightchild(n) = BuildTree(D_2, l + 1)
12     return n
```

Struttura e vincoli strutturali

Un nodo `node` dell'albero contiene almeno:

- punto del dataset;
- riferimento al figlio sinistro;
- riferimento al figlio destro.

In seguito si farà riferimento alle funzioni

- *point*(n) per indicare il punto del dataset associato al nodo n,
- *leftchild*(n) per indicare il figlio sinistro di n,
- *rightchild*(n) per indicare il figlio destro di n.

Sia $\mathcal{D}$ un dataset k-dimensionale e sia $\mathcal{A}$ il *k-d-Tree* costruito per indicizzare $\mathcal{D}$. Dal punto di vista strutturale, $\mathcal{A}$ gode delle proprietà di seguito riportate.

Sia n un nodo dell'albero, sia p il punto associato a n e sia l il livello di n, allora:

- al livello l è associata la coordinata $c = l\%k$ come dimensione di taglio,
- per ogni nodo n' del sottoalbero sinistro di n, il punto p' associato a n' è tale che

$$p'[c] < p[c],$$

- per ogni nodo n' del sottoalbero destro di n, il punto p' associato a n' è tale che

$$p'[c] \geq p[c],$$

- ad ogni nodo n è implicitamente associata una regione rettangolare $\mathcal{H}(n)$,

- alla radice dell'albero è associata l'intera regione indotta da $\mathcal{D}$.

La regione $\mathcal{H}(n)$ associata a un nodo n è definita come segue:

- se n è la radice allora

$$\forall j \in [0,k), \mathcal{H}(n)_j^{\min} = \min \mathcal{D}_j \text{ e } \mathcal{H}(n)_j^{\max} = \max \mathcal{D}_j$$

  ossia $\mathcal{H}(n)$ è l'intera regione indotta da $\mathcal{D}$

- altrimenti, sia p il padre di n, sia c la dimensione di taglio associata con il livello di p e sia p_c il valore che il punto associato al nodo p assume sulla coordinata c, allora

$$\forall j \neq c, \mathcal{H}(n)_j^{\min} = \mathcal{H}(p)_j^{\min} \text{ e } \mathcal{H}(n)_j^{\max} = \mathcal{H}(p)_j^{\max}$$

  se n è il figlio sinistro di p allora

$$\mathcal{H}(n)_c^{\min} = \mathcal{H}(p)_c^{\min} \text{ e } \mathcal{H}(n)_c^{\max} = p_c$$

  se n è il figlio destro di p allora

$$\mathcal{H}(n)_c^{\min} = p_c \text{ e } \mathcal{H}(n)_c^{\max} = \mathcal{H}(p)_c^{\max}.$$

Algoritmo di costruzione

Dato un dataset $\mathcal{D}$, l'algoritmo di costruzione del *k-d-Tree* associato a $\mathcal{D}$ è specificato dalla funzione BuildTree. L'algoritmo costruisce l'albero ricorsivamente. Ad ogni iterazione viene calcolata la dimensione di taglio c sulla base del livello corrente, viene individuato nel dataset il punto mediano P rispetto alla dimensione di taglio e il valore v che tale punto assume su c costituisce il valore di taglio. Pertanto, il punto P viene associato ad un nuovo nodo n, tutti i punti del dataset con valore sulla dimensione c inferiore a v andranno a comporre il sottoalbero di sinistra di n, tutti i punti del dataset con valore sulla dimensione c superiore o uguale a v, ad eccezione di P, andranno a comporre il sottoalbero di destra di n.

Algoritmo di Ricerca

Range query

Dato un dataset $\mathcal{D}$, una *range query* è la ricerca di tutti i punti P che distano meno di r da un punto di query Q. L'algoritmo effettua una ricerca ricorsiva. Dato il nodo corrente n e il punto P associato a n, viene valutata l'intersezione tra la regione associata a n e la sfera di raggio r centrata in Q. Se tale intersezione è vuota, nessun punto soddisfa la query, altrimenti viene valutata la distanza euclidea tra P e Q, eventualmente, si inserisce P nell'insieme risultato e viene effettuata una ricerca dei punti che soddisfano la query nei sottoalberi di n.

Per valutare l'intersezione tra la regione associata a n e la sfera di raggio r centrata in Q si può far riferimento alla funzione Distance che calcola la distanza di un punto da una regione. Se tale distanza è minore di r, l'intersezione è nulla.

Function RangeQuery(n,Q,r)

Input: un nodo n, un punto Q, un raggio r
Output: l'insieme dei punti la cui distanza da q è minore di r

```
1  begin
2      if Distance(Q, H(n)) > r then
3          return ∅
4      P = point(n);
5      L = ∅;
6      if EuclideanDistance(Q, P) ≤ r then
7          L = {Q};
8      if leftchild(n) is not null then
9          L = L ∪ RangeQuery(leftchild(n), Q, r);
10     if rightchild(n) is not null then
11         L = L ∪ RangeQuery(rightchild(n), Q, r);
12     return L
```

PCA

Nella presente descrizione, vengono omessi i dettagli teorici alla base della *PCA* in quanto esulano dal contenuto del corso.
La *Principal component analysis PCA* è uno dei più importanti metodi usati per ridurre dataset ad alta dimensione in dataset di dimensione minore ed è usata in innumerevoli contesti.
Dato un dataset $\mathcal{D}$ di dimensione $n \times k$, è il numero desiderato di dimensioni h con $h < k$, la tecnica cerca le migliori due matrici, U di dimensione $n \times h$ e V di dimensione $d \times h$ tali che

$$\mathcal{D} \approx U \cdot V^T,$$

dove U prende il nome di matrice degli *score* e V prende il nome di matrice dei *load*. La matrice U rappresenta la versione a dimensionalità ridotta del dataset di input. Si passa, quindi, da un dataset di dimensione $n \times k$ a un dataset ridotto di dimensione $n \times h$.

Uno degli algoritmi più diffusi per il calcolo delle matrici U e V è l'algoritmo *NIPALS* la cui descrizione è riportata in figura. È un algoritmo iterativo e ad ogni iterazione viene calcolata una nuova componente, ossia una nuova colonna di U e una nuova colonna di V. Si noti che l'algoritmo prevede che il dataset di input venga centrato rispetto alla media.

Centrare rispetto alla media vuol dire trasformare i punti del dataset nel seguente modo:

$$\forall i \in [0, n), \forall j \in [0, k) \quad \mathcal{D}^i_j = \mathcal{D}^i_j - \overline{\mathcal{D}_j}$$

dove $\mathcal{D}^i_j$ indica la j-esima colonna dell'i-esima riga di $\mathcal{D}$ e $\overline{\mathcal{D}_j}$ indica il valore medio della colonna j-esima.

Range query con PCA

Nel caso in cui il dataset di input venga indicizzato a valle della PCA il punto di query Q dovrà essere a sua volta proiettato utilizzando la trasformazione ottenuta dall'esecuzione

Function Distance(Q,$\mathcal{H}$)

Input: regione k-dimensionale $\mathcal{H}$ e un punto $Q = [q_0, \ldots, q_{k-1}]$
Output: distanza di Q da $\mathcal{H}$

```
1 begin
    // cerco il punto P della regione H più prossimo a Q
2   foreach j ∈ [0, k) do
3     if q_j ≤ H_j^min then
4       p_j = H_j^min;
5     else if q_j ≥ H_j^max then
6       p_j = H_j^max;
7     else
8       p_j = q_j;
9   return EuclideanDistance(P, Q)
```

dell'algoritmo di PCA. In particolare, il punto Q dovrà essere centrato rispetto alla media del dataset e trasformato moltiplicandolo per la matrice dei load V. Quindi, sia $\widehat{Q}$ il punto Q centrato sulla media del dataset, la range query deve essere effettuata sul punto $Q' = \widehat{Q} \cdot V^T$.

2 Descrizione dell'attività progettuale

Obiettivo del progetto è mettere a punto un'implementazione dell'algoritmo di PCA e del *k-d-Tree* in linguaggio C e di migliorarne le prestazioni utilizzando le tecniche di ottimizzazione basate sull'organizzazione dell'hardware.
L'ambiente sw/hw di riferimento è costituito dal linguaggio di programmazione C (`gcc`), dal linguaggio assembly x86-32+SSE e dalla sua estensione x86-32+AVX (`nasm`) e dal sistema operativo Linux (`ubuntu`).
In particolare il codice deve consentire di effettuare la PCA del dataset in input (parametro `-pca <h>`), la costruzione del *k-d-Tree* (parametro `-kdtree`) e range queries (parametro `-rq <QD> <r>`). Quindi la chiamata avrà la seguente struttura:

```
./kdtreepca_idGruppo D [-pca <h>] [-kdtree [-rq <r>]]
    D: nome del dataset
    -pca <h>: esecuzione PCA con h componenti
    -kdtree: costruzione dell'indice
    -rq <r>: range query con raggio r
```

Qualora un valore di un parametro (sia esso di default o specificato dall'utente) non sia applicabile, il codice deve segnalarlo con un messaggio e terminare.

Di seguito si riportano ulteriori linee guida per lo svolgimento del progetto:

- Si consiglia di affrontare il progetto nel seguente modo:

    1. Codificare l'algoritmo interamente in linguaggio C, possibilmente come sequenza di chiamate a funzioni;

ALGORITMO 1: NIPALS

Input: dataset $\mathcal{D}$ di dimensione $(n \times k)$, numero $h < k$ di componenti principali desiderate
Output: matrice $(n \times h)$ degli *score* U, matrice $(d \times h)$ dei *load* V

```
1  imposta la soglia θ a 1e − 8
2  centra D sulla media
3  sia u la prima colonna di D
4  for j ∈ [0, h) do
5      v = (X^T · u)/(u^t · u)// Calcola il vettore dei load
6      v = v/norm(v)// Normalizza il vettore dei load
7      t = u^T · u
8      u = (D · v)/(v^T · v)// Aggiorna il vettore degli score
9      t' = u^T · u
10     if abs(t' − t) >= θ · t' then
11         goto line 5
12     inserisci u come j-esima colonna di U
13     inserisci v come j-esima colonna di V
14     D = D − u · v^T// Aggiorna il dataset
15 return U, V
```

2. Sostituire le funzioni scritte in linguaggio ad alto livello che necessitano di essere ottimizzate con corrispondenti funzioni scritte in linguaggo assembly.

Ciò consentirà di verificare che l'algoritmo che si intende ottimizzare è corretto e di gestire più facilmente la complessità del progetto.

- Al fine di migliorare la valutazione dell'attività progettuale è preferibile presentare nella relazione un confronto tra le prestazioni delle versioni intermedie, ognuna delle quali introduce una particolare ottimizzazione, e finale del codice. Obiettivo del confronto è sostanziare la bontà delle ottimizzazioni effettuate.

- Occorre lavorare in autonomia e non collaborare con gli altri gruppi. Soluzioni troppo simili riceveranno una valutazione negativa. I progetti verranno messi in competizione.

- Sono richieste due soluzioni software, la prima per l'architettura x86-32+SSE e la seconda per l'architettura x86-64+AVX.

  Per i dettagli riguardanti la redazione del codice fare riferimento ai files `kdtreepca32c.c`, `kdtreepca32.nasm`, `runkdtreepca32` (versione `x86-32+SSE`) e `kdtreepca64c.c`, `kdtreepca64.nasm`, `runkdtree64` (versione `x86-64+AVX`) disponibili sulla piattaforma `didattica.dimes.unical.it`.

  Per l'interfacciamento tra programmi in linguaggio C e programmi in linguaggio assembly fare riferimento al documento allegato alla descrizione del progetto.

- Le soluzioni base non devono far uso delle istruzioni `OpenMP`. Opzionalmente, è possibile consegnare delle soluzioni aggiuntive che facciano uso anche di istruzioni `OpenMP`. I nomi dei relativi file dovranno contenere il suffisso "`_omp`" (es. `kdtreepca32c_omp.c`).

- Il software dovrà essere corredato da una relazione. Per la presentazione del progetto è possibile avvalersi di slide.

- Prima della data di consegna del progetto verranno pubblicate le convenzioni da rispettare riguardanti i nomi e la collocazione dei file/directory al fine della compilazione e l'esecuzione dei codici di programma mediante script appositamente predisposti. **Dato l'elevato numero di progetti, sarà cura del candidato accertarsi di aver rispettato pienamente le convenzioni di consegna.**

Buon lavoro!