

# Progetto Architetture 2025

## Calcolo approssimato dei K vicini

Fabrizio Angiulli, Fabio Fassetti

a.a. 2025-26

### 1 Introduzione

Il problema della ricerca dei  $K$  vicini ( $K$ -Nearest Neighbors,  $K$ -NN) è uno dei problemi fondamentali nell'analisi dei dati e nell'apprendimento automatico. Dato un insieme di punti in uno spazio metrico e un punto query, l'obiettivo è identificare i  $K$  punti dell'insieme che sono più vicini al punto query secondo una certa misura di distanza. Questo problema trova applicazione in numerosi domini: dalla classificazione e regressione nel machine learning, ai sistemi di raccomandazione, dal recupero di informazioni alla bioinformatica (ad esempio, nella ricerca di sequenze simili o nell'analisi di espressione genica). Nonostante la sua apparente semplicità concettuale, il problema dei  $K$ -NN presenta sfide computazionali significative quando si lavora con dataset di grandi dimensioni o spazi ad alta dimensionalità. L'approccio naïve, che consiste nel calcolare la distanza tra il punto query e tutti i punti del dataset, ha complessità temporale  $O(n \cdot D)$ , dove  $n$  è il numero di punti e  $D$  la dimensionalità dello spazio. Questo diventa rapidamente proibitivo per dataset reali che possono contenere milioni di punti in spazi ad alta dimensione. Per questo motivo, sono stati sviluppati numerosi algoritmi di ricerca approssimata dei  $K$  vicini (Approximate Nearest Neighbor, ANN) che sacrificano una piccola quantità di precisione in cambio di guadagni sostanziali in termini di efficienza computazionale. Questi algoritmi rappresentano un esempio eccellente del trade-off tra accuratezza ed efficienza.

### 2 Descrizione del problema

Si consideri un dataset  $DS \subset \mathbb{R}^D$  composto da  $n$  punti, dove ogni punto è un vettore reale a  $D$  dimensioni, e una funzione di distanza  $d: \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}^+$ . Dato un punto query  $q \in \mathbb{R}^D$  l'insieme dei  $K$ -Nearest Neighbors ( $K$ -NN) di  $q$  è l'insieme  $N \subseteq DS$  di  $K$  punti tale che

$$\forall v \in N, \forall v' \in DS \setminus N : d(q, v) \leq d(q, v').$$

Trovare i  $K$  vicini esatti di un punto query  $q$  richiede quindi di calcolare la distanza tra  $q$  e tutti gli  $n$  punti del dataset e, tra queste, selezionare le  $K$  più piccole.

La funzione di distanza tra due punti  $v$  e  $w$  più ampiamente utilizzata è la *distanza euclidea*, ossia

$$d(v, w) = \sqrt{\sum_{i \in [1..D]} (v_i - w_i)^2} \quad (1)$$

dove il pedice  $\cdot_i$  indica l' $i$ -esimo elemento del vettore, il cui costo è lineare nel numero di dimensioni  $D$ . Complessivamente, quindi, trovare i  $K$  vicini di un punto  $q$  richiede  $O(n \cdot D)$  operazioni, costo che diventa troppo oneroso quando il dataset cresce di taglia (ossia per grandi valori di  $n$ ) e di dimensionalità (ossia per grandi valori di  $D$ ).

Per ovviare a questo problema, sono state proposte numerose soluzioni in letteratura. Quelle di interesse in questo progetto sono due:

- indicizzazione dei dati, per ridurre il numero di distanze da calcolare  $n$ ,
- calcolo di una distanza approssimata al posto della distanza euclidea, per ridurre il costo  $D$ .

### 3 Distanza approssimata

L'idea per ridurre il costo computazionale del calcolo della distanza si basa sulla *quantizzazione* dei vettori, di seguito illustrata. Per quanto detto, un punto del dataset è un vettore a  $D$  valori reali, quindi composto da 32 o 64 bit per elemento. L'idea della quantizzazione è di ridurre il numero di bit utilizzati per rappresentare ciascun elemento. In particolare la tecnica da implementare in questo progetto è definita come segue.

Ad esempio, si consideri il vettore  $v = [2.3, 1.5, -6.7, 4.5]$  a  $D = 4$  dimensioni e  $x = 2$ . La quantizzazione di  $v$  produce i vettori

$$v^+ = [0, 0, 0, 1]$$
$$v^- = [0, 0, 1, 0]$$

---

**Function** quantizing( $v, x$ )

---

**Input:** Il vettore  $v$  a  $D$  dimensioni da quantizzare, il parametro  $x$

**Output:** I vettori  $v^+$  e  $v^-$

```
// Inizializzare i vettori  $v^+$  e  $v^-$  a 0
1  $v^+ = [0, \dots, 0]$  // vettore a  $D$  dimensioni
2  $v^- = [0, \dots, 0]$  // vettore a  $D$  dimensioni
// identificare gli  $x$  elementi di  $v$  con massimo valore assoluto
3  $i_1, \dots, i_x = \arg \max_x |v|;$ 
4 foreach  $i \in [i_1, \dots, i_x]$  do
5   if  $v_i \geq 0$  then
6      $v_i^+ = 1$ 
7   else
8      $v_i^- = 1$ 
9 return ( $v^+, v^-$ )
```

---

in quanto, i due elementi di  $v$  con valore assoluto più grande sono  $-6.7$  e  $4.5$ . L'elemento  $-6.7$  è negativo e si trova in posizione 2. Quindi, l'elemento in posizione 2 di  $v^-$  viene posto uguale a 1. L'elemento  $4.5$  è positivo e si trova in posizione 3. Quindi, l'elemento in posizione 3 di  $v^+$  viene posto uguale a 1.

La quantizzazione dei vettori può essere utilizzata per calcolare una distanza approssimata. In particolare, dati due vettori  $v$  e  $w$ , la distanza approssimata  $\tilde{d}$  tra  $v$  e  $w$  è definita come:

$$\tilde{d}(v, w) = (v^+ \circ w^+) + (v^- \circ w^-) - (v^+ \circ w^-) - (v^- \circ w^+), \quad (2)$$

dove il simbolo  $\circ$  rappresenta il prodotto scalare tra i due vettori.

## 4 Indicizzazione mediante pivot

Per quanto riguarda l'indicizzazione del dataset, in questo progetto si adotta una tecnica basata su *pivot*. Dato un numero  $h$ , si sceglie nel dataset un insieme  $P$  di  $h$  punti  $P = \{p_1, \dots, p_h\} \subset DS$  che svolgano il ruolo di pivot. L'idea è quella di costruire un indice che conservi per ogni elemento  $v$  del dataset la distanza approssimata (Eq. (2)) tra  $v$  ed ogni pivot  $p_i$ , come illustrato nella funzione indexing. L'indice viene pre-calcolato ed utilizzato per ogni query.

---

**Function** indexing( $P, DS$ )

---

**Input:** L'insieme dei pivot  $P$ , il dataset  $DS$

**Output:** L'indice  $\mathcal{I}$

```
1 foreach  $v \in DS$  do
2   foreach  $p \in P$  do
3     memorizza in  $\mathcal{I}$  il valore  $\tilde{d}(v, p);$ 
4 return  $\mathcal{I}$ 
```

---

Quando il sistema viene interrogato per individuare i  $K$ -NN di un punto di query  $q$ , non vengono valutate le distanze tra  $q$  ed ogni punto del dataset ma viene sfruttato l'indice come descritto nella funzione querying. L'idea alla base è quella di sfruttare la disuguaglianza triangolare. Questa afferma che, data una qualsiasi terna di punti  $q, v, p$  vale che

$$\tilde{d}(q, p) \leq \tilde{d}(v, p) + \tilde{d}(q, v)$$

da cui si può ricavare che

$$\tilde{d}(q, v) \geq \tilde{d}(q, p) - \tilde{d}(v, p).$$

Analogamente,

$$\tilde{d}(v, p) \leq \tilde{d}(q, p) + \tilde{d}(q, v)$$

da cui si può ricavare che

$$\tilde{d}(q, v) \geq \tilde{d}(v, p) - \tilde{d}(q, p).$$

Quindi, se le distanze tra i punti del dataset  $v$  e i pivot  $p$  sono memorizzate nell'indice, il calcolo della sola distanza tra la query  $q$  e i pivot  $p$  è sufficiente per ottenere un limite inferiore della distanza tra la query  $q$  e il punto  $v$  del dataset e, quindi, un'informazione circa l'appartenenza o meno di  $v$  ai  $K$ -NN di  $q$ , senza il calcolo esplicito della distanza tra  $q$  e  $v$ . In particolare, con  $h$  pivot è possibile considerare solo il massimo valore tra i limiti inferiori calcolati come detto. Infatti, se tra i  $k$  vicini attualmente in lista, il più lontano dista  $d$ , un punto  $v$  entra in lista solo se la distanza tra

$q$  e  $v$  è minore di  $d$ . Tramite i pivot è possibile stimare un limite inferiore alla distanza tra  $q$  e  $v$  e, se questo risulta maggiore di  $d$ , è possibile scartare  $v$  senza ulteriori valutazioni. Di conseguenza, l'informazione più utile è data dal maggiore limite inferiore che è possibile calcolare mediante pivot.

I  $K$ -NN di un punto  $q$  sono, genericamente, memorizzati in una lista ordinata di coppie  $\langle id_i, \delta_i \rangle$  dove  $id_i$  è l'identificativo del punto  $v$  del dataset preso come vicino di  $q$  e  $\delta_i$  è la distanza tra questo punto e la query.

## 4.1 Selezione dei pivot

La selezione dei pivot è cruciale per la resa dell'algoritmo e, pertanto, in letterature sono state proposte numerose tecniche per una loro selezione efficace. Tuttavia, per semplicità, in questo progetto si è deciso di adottare una selezione casuale e, in particolare, dato il numero  $h$  di pivot desiderati, si considerano come pivot i punti del dataset in posizione

$$i = \lfloor n/h \rfloor \cdot j, \text{ with } j = [0..h-1].$$

Ad esempio, se in un dataset di 14 punti si volessero  $h = 4$  pivot, dato che  $\lfloor 14/4 \rfloor = 3$ , questi sarebbero i vettori  $v_0$ ,  $v_3$ ,  $v_6$  e  $v_9$ , ossia i punti a partire da 0 e muovendosi a passo di  $\lfloor n/h \rfloor = 3$ .

---

Function querying( $q, P, DS$ )	
<b>Input:</b> Un punto di query $q$ , l'insieme dei pivot $P$ , il dataset $DS$	
<b>Output:</b> $K$ -NN di $q$ in $DS$ , come coppie $\langle id, \delta \rangle$	
// inizializza i $K$ -NN di $q$ a distanza $+\infty$	
1	$K\text{-NN} = [(-1, +\infty), \dots, (-1, +\infty)];$
2	<b>foreach</b> $p \in P$ <b>do</b>
3	└─ calcola la distanza $\tilde{d}(q, p);$
4	<b>foreach</b> $v \in DS$ <b>do</b>
	└─ // calcolo il più grande limite inferiore alla distanza
5	└─ $d_{pvt}^* = \max_{p \in P}  \tilde{d}(v, p) - \tilde{d}(q, p) ;$
	└─ // prendo il vicino più lontano da $q$ presente in $K$ -NN
6	└─ sia $d_k^{\max}$ la distanza tra $q$ e il punto più lontano da $q$ presente in $K$ -NN;
7	└─ <b>if</b> $d_{pvt}^* < d_k^{\max}$ <b>then</b>
8	└─ └─ calcola il valore $\tilde{d}(q, v);$
9	└─ └─ <b>if</b> $\tilde{d}(q, v) < d_k^{\max}$ <b>then</b>
10	└─ └─ └─ inserisci $v$ in $K$ -NN (rimuovendo il punto a distanza massima);
	└─ // dopo aver individuato i vicini con distanza approssimata
	└─ // si può calcolare la distanza reale
11	<b>foreach</b> $\langle id, \delta \rangle \in K\text{-NN}$ <b>do</b>
12	└─ $\delta = d(q, v_{id})$ // usando l'Equazione (1)
13	<b>return</b> $K\text{-NN}$

---

## 5 Descrizione dell'attività progettuale.

Obiettivo del progetto è mettere a punto un'implementazione dell'algoritmo di indicizzazione e ricerca in linguaggio C e di migliorarne le prestazioni utilizzando le tecniche di ottimizzazione basate sull'organizzazione dell'hardware. Il programma C va reso disponibile come pacchetto Python.

L'ambiente sw/hw di riferimento è costituito dal linguaggio di programmazione C (gcc), dal linguaggio assembly 32+SSE e dalla sua estensione 64+AVX (nasm) e dal sistema operativo Linux (ubuntu).

In particolare il codice deve consentire di trovare i  $K$ -NN di una query  $q$  tramite l'algoritmo precedentemente illustrato, considerando i valori dei parametri:

```
DS: file contenente il dataset
Q:  file contenente le query
h:  numero di pivot
k:  numero di vicini
x:  parametro di quantizzazione
```

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;
2. Sostituire le funzioni scritte in linguaggio ad alto livello che necessitano di essere ottimizzate con corrispondenti funzioni scritte in linguaggio 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 che lavora in aritmetica a virgola mobile in Single Precision (`float`, 32 bit) facendo uso del repertorio `SSE` e la seconda che lavora in aritmetica a virgola mobile in Double Precision (`double`, 64 bit) facendo uso del repertorio `AVX`.

Per i dettagli riguardanti la redazione del codice fare riferimento ai file sorgenti disponibili sulla piattaforma.

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

- È inoltre richiesta per la soluzione `AVX`, una versione che faccia uso delle istruzioni `OpenMP`.
- 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!