Product Quantization for Nearest Neighbor Search

Francesco Tarsia

ftarsia@hotmail.com

Abstract

Questo lavoro descrive un approccio alla ricerca approximate nearest neighbor basato sul product quantization. L'idea è quella di decomporre lo spazio in sottospazi a bassa dimensionalità mediante il prodotto cartesiano, e quantizzare ciascun sottospazio separatamente. Un vettore è rappresentato da un codice breve composto dall'indice di quantizzazione relativo al proprio sottospazio. La distanza euclidea tra due vettori è così stimata dai loro codici. Una versione asimmetrica dell'algoritmo ANN incrementa la precisione, e viene utilizzata per calcolare la distanza approssimata tra il vettore e il codice. Per velocizzare la ricerca sui codici, si implementa un inverted file system (IVF).

1. Introduzione

La ricerca Nearest Neighbor (NN) è fondamentale per l'analisi e l'elaborazione dei dati su larga scala, in particolare per l'analisi di contenuti multimediali che sono spesso ad alta dimensionalità.

La ricerca NN, nota anche come ricerca di prossimità o di ricerca per similitudine, mira a trovare dati/items più vicini o più simili di dati/items.

Si tratta di una fondamentale tecnica in molti campi di applicazioni, come per esempio il recupero di immagini in base al contenuto (data mining multimediale), dove i dati sono spesso rappresentati con vettori multidimensionali.

Questo rende la ricerca NN su larga scala di dati ad alta dimensionalità molto onerosa dal momento che gli archivi sono limitati, così come lo sono le risorse computazionali.

Invece di realizzare l'esatto algoritmo NN, da alcuni anni, la ricerca è orientata ad approssimare l'algoritmo di ricerca NN, il cosiddetto algoritmo di ricerca Approximate Nearest Neighbor (ANN).

La Quantizazzione Vettoriale (VQ) è un metodo popolare e di successo per la ricerca ANN.

Il metodo VQ è utilizzato in due modi per ANN:

- per costruire l'inverted indexing per la ricerca non esaustiva;
- per codificare i vettori di ingresso in codici compatti[1] per la ricerca esaustiva.

Nella ricerca esaustiva, i dati sono quantizzati in codewords; le distanze dei vettori sono approssimati con le distanze delle codewords.

Il metodo di codifica compatta può essere combinato con l'inverted indexing.

Un Product Quantizer (PQ) è una soluzione di VQ quando si vogliono ottenere un gran numero di codewords.

L'idea chiave è quella di scomporre lo spazio vettoriale originale nel Prodotto Cartesiano di M sottospazi a bassa dimensionalità e quantizzare ogni sottospazio in k codewords.

Il numero effettivo di codewords nello spazio originale è k^M, ma il costo per la loro memorizzazione è solo $O(Dk)$ per vettori D-dimensionali.

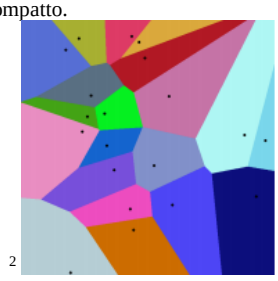
2. Vector Quantization

L'obiettivo di VQ è quello di ridurre la cardinalità dello spazio di rappresentazione, in particolare quando i dati di input sono valori reali.

Un quantizzatore è una funzione q che associa a un vettore D-dimensionale $x \in \Re^D$ un vettore $q(x) \in C = \{c_i ; i \in I\}, I = 0,...,k-1.$

I valori c_i sono chiamati *centroidi*. L'insieme dei valori C è il *codebook* di dimensione k. L'insieme di vettori v_i mappati a un dato indice i si riferisce a una *cella di Voronoi*[2] definita come

[1] Il codice di Huffman, che associa parole di codice di minore lunghezza a stringhe di maggiore frequenza relativa è un codice compatto.



[2] Un diagramma di Voronoi è un modo di dividere lo spazio in un certo numero di regioni. Dato un insieme di punti (il generatore del

$$v_i = \{ x \in \Re^D \ : q(x) = c_i \}.$$

Le *k* celle di un quantizzatore formano una partizione di $\Re^D$. Tutti i vettori che si trovano nella stessa cella v_i sono costruiti dallo stesso centroide c_i.

La qualità di un quantizzatore è generalmente misurata dall'errore quadratico medio (MSE) tra il vettore di input x e il suo valore di riproduzione $q(x)$:

$$MSE(q) = E_X\left[d(q(x), x)^2\right] = \int p(x)\, d(q(x), x)^2\, dx,$$

dove $d(x, y) = \|x - y\|$ è la distanza euclidea tra x e y e dove $p(x)$ è la funzione di distribuzione della probabilità corrispondente alla variabile aleatoria X.

Affinchè il quantizzatore sia ottimale, deve soddisfare due proprietà conosciute come le condizioni di ottimalità di Lloyd. In primo luogo, un vettore x deve essere quantizzato al suo codebook centroide più vicino, in termini di distanza euclidea:

$$q(x) = \arg\min_{c_i \in C} d(x, c_i)^3.$$

La seconda condizione di Lloyd è che la ricostruzione deve essere il valore atteso dei vettori che si trovano nella cella di Voronoi:

$$c_i = E_X\left[x|i\right] = \int_{v_i} p(x)\, x\, dx.$$

Il quantizzatore di Lloyd, che corrisponde all'algoritmo di clustering k-means, trova un near-optimal codebook iterativamente assegnando i vettori a un training set di centroidi e ristimando questi centroidi dai punti assegnati.

Nel seguito, assumiamo che le due condizioni di Lloyd siano valide. Da notare, comunque, che k-means trova un ottimo locale in termini di errore di quantizzazione.

3. Product Quantizers

Consideriamo un vettore 128-dimensionale, per esempio il descrittore SIFT[4]. Un quantizzatore produce codici a 64 bits con $k = 2^{64}$ centroidi.

Pertanto, è impossibile utilizzare l'algoritmo di Lloyd o addirittura il k-means gerarchico (HKM), dal momento che il numero di campioni richiesto e la complessità di apprendimento del quantizzatore crescono esponenzialmente con k. E' anche impossibile memorizzare i $D \times k$ valori floating point che rappresentano i k centroidi.

PQ è una soluzione efficiente per risolvere questi problemi. Si tratta di una tecnica usata in teoria dell'informazione, più precisamente nella codifica di sorgente, che ci permette di scegliere il numero di componenti da quantizzare congiuntamente (per esempio, gruppi di 24 componenti possono essere quantizzati utilizzando il potente Leech lattice[5].

Il vettore di input x è diviso in m subvectors distinti u_j, $1 \le j \le m$, di dimensione $D^* = \frac{D}{m}$, dove D è un multiplo di m. I subvectors sono quantizzati separatamente utilizzando m quantizzatori distinti. Un dato vettore x è quindi mappato come segue:

$$\underbrace{x_1, ..., x_{D^*}}_{u_1(x)}, ..., \underbrace{x_{D-D^*+1}, ..., x_D}_{u_m(x)} \rightarrow q_1(u_1(x)), ..., q_m(u_m(x)),$$

dove q_j è un quantizzatore a bassa complessità associato al j-esimo subvector. Con il subquantizer q_j associamo l'insieme indice I_j, il codebook C_j e i valori di riproduzione corrispondenti $c_{j,i}$.

Un valore di riproduzione del product quantizer è identificato con un elemento del prodotto dell'insieme indice $I = I_1 \times ... \times I_m$. Il codebook è quindi definito dal Prodotto Cartesiano

$$C = C_1 \times ... \times C_m,$$

e un centroide di questo insieme è la concatenazione di centroidi di m subquantizer. Da ora in poi, assumiamo che tutti i subquantizers hanno lo stesso numero finito k^* di valori di riproduzione. In tal caso, il numero totale di centroidi è dato da $k = (k^*)^m$.

Nota che nel caso estremo dove $m = D$, i componenti di un vettore x sono tutti quantizzati separatamente. Allora il product quantizer risulta essere un quantizzatore scalare, dove la funzione di quantizzazione associata a ciascun componente può essere differente.

Il punto di forza del product quantizer è quello di produrre un vasto insieme di centroidi da diversi piccoli gruppi di centroidi: quelli associati ai subquantizers. Quando l'apprendimento dei subquantizers utilizza l'algoritmo di Lloyd, viene usato un numero limitato di vettori, ma il codebook è, in parte, ancora adattato alla distribuzione dei dati che rappresenta. La complessità di

---

diagramma), associamo ad ognuno la regione che contiene i punti che sono più vicini ad esso che agli altri punti del generatore. Le regioni sono chiamate celle di Voronoi, in onore di Georgy Voronoy.

[3] $\arg\min_{c_i \in C} d(x, c_i)$, argomento del minimo, è l'insieme dei valori di x per i quali $d(x, c_i)$ raggiunge il suo valore minimo.

[4] Scale Invariant Feature Transform (SIFT) di David G. Lowe è del 1999, trasforma una immagine in descrittori locali (SIFT keys) che sono invarianti per scala, rotazione e traslazione.

[5] Il reticolo di Leech, di dimensione 24, in matematica riguarda il problema dell'impacchettamento, consiste nello stabilire una configurazione per inserire una classe di oggetti (non deformabili e non compenetrabili) all'interno di un contenitore, rispettando un qualche criterio di ottimalità. La configurazione ottima si ottiene tassellando con esagoni regolari (come nel caso di un alveare) e collocando i centri dei cerchi sui vertici e sui centri degli esagoni.

apprendimento del quantizzatore è m volte la complessità di esecuzione del clustering k-means con k^* centroidi di dimensione D^*.

La memorizzazione del codebook C non è efficiente. Invece, memorizziamo $m \times k^*$ centroidi di tutti i subquantizers, vale a dire, $mD^*k^* = k^*D$ valori floating points. Quantizzare un elemento richiede k^*D operazioni floating point. Il product quantizer è l'unico che può essere indicizzato in memoria per grandi valori di k.

Al fine di fornire buone proprietà di quantizzazione quando si sceglie un valore costante di k^*, ciascun subvector dovrebbe avere, in media, un'energia comparabile. Un modo per garantire questa proprietà è di moltiplicare il vettore con una matrice ortogonale random prima della quantizzazione. Tuttavia, per la maggior parte dei tipi di vettore questo non è richiesto e neanche raccomandato. Poiché i sottospazi sono ortogonali, la distorsione quadrata associata con il product quantizer è

$$MSE(q) = \sum_j MSE(q_j),$$

dove $MSE(q_j)$ *è la distorsione associata al quantizzatore* q_j.

4. Ricerca con Quantizzazione

La ricerca Nearest Neighbor dipende dalle distanze tra il vettore di query e i vettore del database, o equivalentemente le distanze al quadrato. Il metodo introdotto confronta i vettori base sui loro indici di quantizzazione così come è indicato dalle tecniche che riguardano la codifica di sorgente. Per prima cosa spieghiamo come le proprietà del product quantizer vengono utilizzate per calcolare le distanze. Poi mettiamo a disposizione una statistica legata alla stima dell'errore della distanza, e viene proposto uno stimatore raffinato per il quadrato della distanza Euclidea.

Consideriamo il vettore di query x e il vettore di database y. Proponiamo due metodi per calcolare una distanza Euclidea approssimata tra questi vettori, una simmetrica e una asimmetrica.

Symmetric distance computation (SDC): entrambi i vettori x e y sono rappresentati dai loro rispettivi centroidi $q(x)$ e $q(y)$. La distanza $d(x,y)$ è approssimata dalla distanza $\hat{d}(x,y) = d(q(x),q(y))$ che viene ottenuta in modo efficiente mediante un product quantizer $\hat{d}(x,y) = d(q(x),q(y)) = \sqrt{\sum_j d(q_j(x),q_j(y))^2}$, dove la distanza $d(c_{j,i},c_{j,i'})^2$ viene letta da una look-up table[6] associata con il j-esimo subquantizer. Ogni look-up table contiene tutte le distanze al quadrato tra coppie di centroidi (i,i') del subquantizer.

Asymmetric distance computation (ADC): il vettore di database y è rappresentato da $q(y)$, ma la query x non è codificata. La distanza $d(x,y)$ è approssimata dalla distanza $\tilde{d}(x,y) = d(x,q(y))$, che viene calcolata utilizzando la decomposizione $\tilde{d}(x,y) = d(x,q(y)) = \sqrt{\sum_j d(u_j(x),q_j(u_j(y)))^2}$, dove le distanze al quadrato $d(u_j(x),c_{j,i})^2 : j = 1,...,m, i = 1,...,k^*$, sono calcolate prima della ricerca.

Per la ricerca nearest neighbors, in pratica non calcoliamo la radice quadrata: la funzione radice quadrata è monotona crescente e le distanza al quadrato producono lo stesso vector ranking.

SDC e ADC hanno lo stesso costo computazionale nella preparazione di query, che non dipende dalle dimensione n del dataset. Quando n è grande $(n > k^*D^*)$, le operazioni che hanno il costo più alto sono le somme. La complessità per la ricerca di k elementi più piccoli è la complessità media per $n >> k$ e quando gli elementi sono ordinati in modo arbitrario.

L'unico vantaggio di SDC su ADC è quello di limitare l'utilizzo della memoria associata con le query, come quando il vettore di query è definito da un codice. La versione asimmetrica ottiene una più bassa distorsione.

5. Ricerca non esaustiva

La ricerca ANN con product quantizer è veloce (solo m addizioni sono necessarie per il calcolo della distanza) e riduce in modo significativo i requisiti di memoria per la memorizzazione dei descrittori. Tuttavia, la ricerca è esaustiva. Se un'immagine viene descritta da un insieme di descrittori locali, è proibitivo usare la ricerca esaustiva, dal momento che abbiamo bisogno di indici dell'ordine di

[6] Per **lookup table** si intende una struttura dati, generalmente un array, usata per sostituire operazioni di calcolo a *runtime* con una più semplice operazione di consultazione (lookup). Il guadagno di velocità può essere significativo, poiché recuperare un valore dalla memoria è spesso più veloce che sottoporsi a calcoli con tempi di esecuzione dispendiosi. Un esempio classico sono le tabelle trigonometriche. Calcolare il seno di valore ogni qual volta serve può rallentare i processi di calcolo in certe applicazioni. Per evitare ciò l'applicazione all'avvio può impiegare qualche secondo per calcolarsi il valore del seno per un certo numero di valori. Poi, quando il programma ha bisogno del seno di un certo valore, usa la lookup table per recuperare il valore del seno dall'indirizzo della memoria, anziché calcolarlo usando una formula matematica.

miliardi per elaborare query multiple.

Per evitare la ricerca esaustiva combiniamo un inverted file system con il calcolo asimmetrica della distanza (IVFADC). Un inverted file quantizza i descrittori e memorizza gli indici dell'immagine nelle liste corrispondenti. Ciò consente un accesso rapido a una piccola frazione di indici dell'immagine ed è stato mostrato che ha ottime performance nella ricerca che coinvolge dataset a larghissima scala. Invece di memorizzare un solo image index, aggiungiamo un piccolo codice per ogni descrittore. Codifichiamo poi la differenza tra il vettore e il suo corrispondente coarse centroid con un product quantizer. Questo approccio accelera notevolmente la ricerca al costo di pochi bits/bytes aggiuntivi per descrittore. Per di più, migliora leggermente la precisione della ricerca, così come la codifica del residuo è più precisa della codifica del vettore stesso.

5.1. Coarse quantizer

In modo simile all'approccio "Video-Google"[7], si apprende un codebook con l'algoritmo k-means, producendo un quantizzatore q_c. Per i descrittori SIFT, il numero k' di centroidi associati con q_c varia tipicamente da $k'=1000$ a $k'=1000000$. È pertanto un numero piccolo rispetto a quello dei product quantizers descritti in precedenza.

Oltre al coarse quantizer, si adotta una strategia simile a quella proposta da Jégou[8], cioè, la descrizione di un vettore è raffinato da un codice ottenuto con un product quantizer. Però, al fine di tener conto delle informazioni fornite dal coarse quantizer, cioè il centroide $q_c(y)$ associato al vettore y, il product quantizer q_p viene utilizzato per codificare il vettore residuo

$$r(y) = y - q_c(y),$$

corrispondente all'offset nella cella di Voronoi. L'energia del vettore residuo è piccola rispetto a quella del vettore stesso. Il vettore è rappresentato da

$$\ddot{y} = q_c(y) + q_p(y - q_c(y)).$$

Esso è rappresentato dalla tupla $(q_c(y), q_p(r(y)))$. Per analogia con la rappresentazione binaria di un valore, il coarse quantizer fornisce i bit più significativi, mentre il codice relativo al product quantizer corrisponde ai bit meno significativi.

Lo stimatore di $d(x,y)$, dove x è la query e y il vettore di database, è calcolato come la distanza $\ddot{d}(x,y)$ tra x e $\ddot{y}$:

$$\ddot{d}(x,y) = d(x,\ddot{y}) = d\left(x - q_c(y), q_p(y - q_c(y))\right).$$

Indicando con q_{p_j} il j-esimo subquantizer, usiamo la seguente decomposizione per calcolare questo stimatore in maniera efficiente:

$$\ddot{d}(x,y)^2 = \sum_j d\left(u_j(x - q_c(y)), q_{p_j}\left(u_j(y - q_c(y))\right)\right)^2.$$

Simile alla strategia ADC, per ogni subquantizer q_{p_j} le distanze tra il vettore residuo parziale $u_j(x - q_c(y))$ e tutti i centroidi $c_{j,i}$ di q_{p_j} vengono preliminarmente calcolati e memorizzati.

Il product quantizer è appreso su un insieme di vettori residui provenienti da un learning set. Sebbene i vettori siano quantizzati da indici diversi dai coarse quantizer, la risultante dei vettori residui viene utilizzata per apprendere un unico product quantizer. Partiamo dal presupposto che lo stesso product quantizer è accurato quando la distribuzione del residuo è marginalizzata su tutte le celle di Voronoi. Questo dà probabilmente risultati inferiori rispetto all'approccio che utilizza un apprendimento distinto del product quantizer per ogni cella di Voronoi. Tuttavia, questo sarebbe computazionalmente costoso e richiederebbe k' memorizzazioni del product quantizer codebook, cioè $k' \times d \times k^*$ valori in floating point, che risulterebbero intrattabili dal punto di vista della memoria per valori comuni di k'.

5.2. Struttura indicizzata

Usiamo la coarse quantizer per implementare una struttura inverted file come un vettore di liste $L_1, ..., L_{k'}$. Se Y è il vettore dataset per indicizzare, la lista L_i associata al centroide c_i di q_c memorizza l'insieme $\{y \in Y : q_c(y) = c_i\}$.

Nella lista invertita L_i, una entry corrispondente a y contiene un vettore identificatore e il residuo codificato $q_p(r(y))$:

campo	lunghezza (bits)
identificatore	8-32
codice	$m\lceil \log_2 k^* \rceil$

Il campo identificatore risulta essere un sovraccarico, dovuto proprio alla struttura inverted file. A seconda della natura dei vettori di essere memorizzati, l'identificatore non è necessariamente unico. Esempio: per descrivere immagini attraverso descrittori locali, identificatori immagine possono sostituire identificatori vettore, cioè, tutti i vettori della stessa immagine hanno lo stesso

[7] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos", in *Proceedings of the International Conference on Computer Vision*, pp. 1470-1477, 2003.

[8] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search", in *Proceedings of the European Conference on Computer Vision*, October 2008.

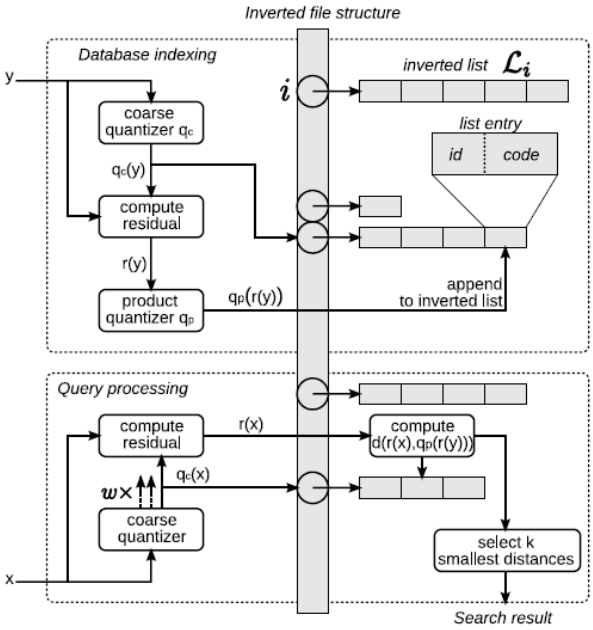
identificatore. Pertanto, un campo di 20 bit è sufficiente per identificare un'immagine da un dataset di un milione. Questo costo di memoria può essere ridotto ulteriormente utilizzando la compressione dell'indice[9], che possono ridurre il costo medio di memorizzazione dell'identificatore di circa 8 bit, a seconda dei parametri.

5.3. Algoritmo di ricerca

L'inverted file è la chiave per la versione non esaustiva del nostro metodo. Durante la ricerca dei nearest neighbors di un vettore x, l'inverted file fornisce un sottoinsieme di Y per cui le distanze sono stimate: soltanto la lista invertita L_i corrispondente a $q_c(x)$ viene sottoposta a scansione.

Tuttavia, x e il suo nearest neighbor spesso non sono quantizzati allo stesso centroide, ma per quelli vicini. Per affrontare questo problema, si usa la strategia dell'assegnazione multipla[10]. La query x è assegnata a w indici invece di uno soltanto, che corrispondono ai w nearest neighbors di x nel codebook di q_c. Tutte le liste invertite corrispondenti vengono sottoposte a scansione. Assegnazioni multiple non vengono applicate ai vettori del database, in quanto ciò aumenterebbe l'utilizzo della memoria.

La figura seguente fornisce una panoramica di come un database viene indicizzato con una struttura IVFADC.



L'**Indexing** di un vettore y viene eseguito attraverso i seguenti passi:

1) quantizzare y con $q_c(y)$;
2) calcolare il residuo $r(y) = y - q_c(y)$;
3) quantizzare $r(y)$ con $q_p(r(y))$, che, per il product quantizer, equivale assegnare $u_j(y)$ a $q_j(u_j(y))$, per $j = 1,...,m$;
4) aggiungere una nuova entry alla lista invertita corrispondente a $q_c(y)$. Essa contiene il vettore identificatore (o immagine) e il codice binario (indici del product quantizer).

Searching del nearest neighbor(s) di una query x segue i seguenti passi:

1) quantizzare x ai suoi w nearest neighbors nel codebook q_c;

   Per semplicità, nei due passi successivi indichiamo con $r(x)$ i residui associati con le w assegnazioni.
2) calcolare la distanza al quadrato $d(u_j(r(x)), c_{j,i})^2$ per ciascun subquantizer j e ciascuno dei suoi centroidi $c_{j,i}$;
3) calcolare la distanza al quadrato tra $r(x)$ e tutti i vettori indici della lista invertita. Si utilizzano le distanze subvector-to-centroid calcolate nel passo precedente, questo consiste nel riassumere m valori looked-up;

[9] M. Perdoch, O. Chum, and J. Matas, "Efficient representation of local geometry for large scale object retrieval", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2009.

H. Jégou, M. Douze, and C. Schmid, "Packing bag-of-features", in *Proceedings of the International Conference on Computer Vision*, September 2009.

[10] H. Jégou, H. Harzallah, and C. Schmid, "A contextual dissimilarity measure for accurate and efficient image search", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

4) selezionare i K nearest neighbors di x in base alle distanza stimate. Questo è implementato efficientemente mantenendo una struttura Maxheap[11] di dimensione fissa, che memorizza i valori minimi K visti finora. Dopo ogni calcolo della distanza, l'identificatore viene aggiunto alla struttura solo se la sua distanza è inferiore alla distanza più grande nel Maxheap.

Solo il passo 3 dipende dalla dimensione del database. Comparato all'ADC, l'ulteriore passo di quantizzazione x a $q_c(x)$ è costituito nel calcolare k' distanze tra vettori D – dimensionali. Supponendo che le liste invertite siano bilanciate, all'incirca $n \times w / k'$ entry devono essere analizzate.

6. Valutazione di NN Search

In questa sezione, presentiamo il dataset utilizzato per la valutazione. Il nostro approccio è confrontato con spectral hashing (SH)[12].

6.1. Dataset

Eseguiamo i nostri esperimenti su due dataset, ANN_SIFT1M[13] e ANN_SIFT10K[14]. Abbiamo tre sottoinsiemi di vettori per dataset:

- learning;
- database;
- query.

Entrambi i dataset sono stati costruiti utilizzando i dati e il software di pubblico dominio. Per i descrittori SIFT, il learning set è stato estratto da immagini di Flickr[15] e i descrittori di database e query provengono da immagini INRIA Holidays[16]. La seguente tabella riassume il numero di descrittori estratti per i due dataset.

vettore dataset:	SIFT	SIFTsmall
dimensione descrittore *d*	128	128
learning set size	100.000	25.000
database set size	1.000.000	10.000
queries set size	10.000	100

La qualità della ricerca è misurata con recall@*R*, vale a dire, la percentuale di vettori di query per i quali il nearest neighbor è classificato nelle prime *R* posizioni. Questa misura indica la frazione di query per le quali il nearest neighbor è recuperato correttamente, se si verifica una short-list di *R* vettori utilizzando distanze euclidee.

In pratica, siamo spesso interessati a recuperare i K nearest neighbor $(K > 1)$e non solo il singolo nearest neighbor.

6.2. Memoria vs accuratezza di ricerca: compromessi

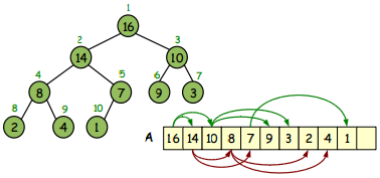
Il product quantizer è parametrizzato dal numero di subvector m e il numero di quantizzatori per subvector k^*, producendo un codice di lunghezza $m \times \log_2 k^*$. Il compromesso tra la lunghezza del codice e la qualità della ricerca del descrittore SIFT è importante. Per quanto riguarda la distorsione del quantizzatore, si può osservare che per un numero fisso di bit, è meglio usare un piccolo numero di subquantizer con molti centroidi che avere molti subquantizer con pochi bit. Lo stimatore asimmetrico ADC relativamente a recall@100 supera in modo significativo SDC. Per $m = 8$ (m subvector) la precisione (recall@100) di ADC con $k^* = 64$ (centroidi) è uguale a quella di SDC con $k^* = 256$ centroidi. La tabella seguente riassume SDC e ADC con $m = 8$.

distanza	m	k^*	recall@100	Code length $m \times \log_2 k^*$
SDC	8	256	**0.7**	64
ADC	8	64	**0.7**	48
SDC	8	16	0.23	32
SDC	8	64	0.5	48
SDC	8	1024	0.9	80
SDC	8	4096	0.98	96
ADC	8	16	0.4	32
ADC	8	256	0.96	64
ADC	8	1024	0.98	80
ADC	8	4096	0.99	96

Per l'approccio IVFADC, abbiamo, inoltre la dimensione dell'insieme codebook k' e il numero di neighboring cells w visitate durante l'assegnazione multipla.

La seguente tabella riassume l'approccio IVFADC:

[11] Un heap binario è una struttura dati composta da un array che può essere considerato come un albero binario quasi completo (possono mancare alcune foglie consecutive a partire dall'ultima foglia a destra).



Proprietà del Max-heap: in un Max-heap ogni nodo i diverso dalla radice è tale che A[parent[i]] ≥ A[i]

[12] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing", in *Advances in Neural Information Processing Systems*, 2008.

[13] ftp://ftp.irisa.fr/local/texmex/corpus/sift.tar.gz.

[14] ftp://ftp.irisa.fr/local/texmex/corpus/siftsmall.tar.gz.

[15] Flickr è un'applicazione online per la gestione e la condivisione di foto, è di proprietà del gruppo Yahoo.

[16] https://lear.inrialpes.fr/~jegou/data.php#holidays.

k'	m	w	recall@100	Code length
1024	8	1	0.4	64
1024	8	8	0.82	64
8192	8	8	0.7	64
8192	8	64	0.95	64
1024	4	1	0.4	32
1024	4	8	0.7	32
8192	4	8	0.69	32
8192	4	64	0.85	32

Osserviamo che recall@100 dipende fortemente da k' e w,
e che aumentare la lunghezza del codice è inutile se w non è abbastanza grande, così come i nearest neighbor che non sono assegnati a uno dei w centroidi associati con le query vengono definitivamente persi.

L'approccio IVFADC è molto più efficiente di SDC e ADC su grandi dataset, in quanto mette a confronto solo la query per una piccola frazione dei vettori di database.

7. Esperimenti

Il lavoro è principalmente sviluppato e implementato nel linguaggio Python, con una parte in linguaggio C che riguarda la parte più computazionalmente intensiva.
Sono usati 2 algoritmi:

- Spectral Hashing (SH);
- Product Quantization (PQ).

SH mappa il vettore di input in codici binari, e misura le similarità attraverso la distanza di Hamming.

PQ divide lo spazio vettoriale in m parti e mappa un vettori di input in k^* centroidi precalcolati in ogni sottospazio. Poi la distanza nello spazio originale può essere approssimata da quella tra questi centroidi. Utilizziamo *l'Asymmetric Distance Computation* (ADC) per approssimare la distanza. ADC mappa soltanto i vettori di base per i centroidi precalcolati, quindi approssima la distanza confrontando i centroidi e il vettore di query originale, invece che il vettore di query quantizzato. Il calcolo della distanza tra codici PQ è accelerata da tabelle look-up. Per una ricerca esaustiva su codici PQ, la complessità temporale è $O(N \log R)$, dove R è il numero di vettori base restituiti.

Il parametro principale è il numero di bit b. Per l'algoritmo PQ, fissiamo la dimensione del codebook (vocabolario) di ogni subquantizer a $k^* = 256$, in modo tale che ogni indice è codificato da 8 bit e il numero di subquantizer è $m = b/8$.

Per accelerare la ricerca sui codici PQ, si implementa un inverted file system (IVF) per eseguire una ricerca non esaustiva. I vettori di base vengono prima separati in k' gruppi da un coarse quantizer q_c, poi i vettori residuo in ciascun gruppo sono compressi con codici PQ. Il vettore residuo è l'offset rispetto al centroide in ciascun gruppo: $r(y) = y - q_c(y)$. Quando arriva una query, solo i w gruppi che corrispondono alle w nearest neighbors della query nel coarse codebook saranno controllati. I risultati finali si ottengono effettuando la ricerca dei codici PQ in questi gruppi in modo esaustivo.

La valutazione viene effettuata sui dataset SIFT1M e SIFTsmall. SIFT1M si compone di 3 sottogruppi:

1) 1 milione di vettori base su cui viene eseguita la ricerca;
2) 10.000 vettori di query;
3) 100.000 vettori di training per l'apprendimento del codificatore, dove ogni vettore ha dimensione 128.

recall@R è usato per misurare la qualità della ricerca. *recall@R* è la proporzione di vettori di query per i quali il NN è classificato nelle prime R posizioni.
Gli esperimenti sono stati condotti su un computer portatile con processore Intel Core 2 CPU T5200 1.60GHz 1.60GHz con 2GB di RAM. La RAM ridotta ha compromesso l'esperimento per quanto riguarda il metodo IVFADC relativamente al dataset SIFT1M.

La tabella seguente mostra che PQ è più veloce di SH, per piccole dimensioni del database (per SIFT è il contrario):

Search time per 100-*NN* search (ms) - SIFTsmall

Numero di bit	32	64
Spectral Hashing	2.905	3.940
Product Quantization	1.446	1.889
IVFPQ	3.644	5.439

La tabella seguente mostra alcuni risultati con codice di lunghezza 64 bit relativo al database SIFTsmall:

Metodo	***R=1***	***R=10***	***R=100***	***time cost (ms)***
SH	0.3700	0.6500	0.9400	3.940
PQ	0.4700	0.8800	1.0000	1.889
IVFPQ	0.5600	0.8200	0.8900	5.439

La tabella mostra un risultato importante, il costo del passo extra di quantizzazione richiesto da IVFPQ appare chiaramente quando il database è di piccole dimensioni (5.439 ms).

La seguente tabella riassume i risultati con diverse lunghezze di codice relativamente al database SIFTsmall:

Metodo	***R=1***	***R=10***	***R=100***	***time cost (ms)***

SH 16 bit	0.0100	0.2800	0.6500	5.521
SH 32 bit	0.1700	0.5200	0.8400	2.905
SH 64 bit	0.3700	0.6500	0.9400	3.940
PQ 16 bit	0.1700	0.4200	0.8600	4.437
PQ 32 bit	0.2600	0.7100	0.9600	1.446
PQ 64 bit	0.4700	0.8800	1.0000	1.889
IVFPQ 16 bit	0.2900	0.7200	0.8300	23.415
IVFPQ 32 bit	0.4800	0.8200	0.8900	3.644
IVFPQ 64 bit	0.5600	0.8200	0.8900	5.439

Da questa tabella si evince IVFPQ 32 bit in termini di recall@100 si comporta leggermente meglio rispetto a SH 32 bit e PQ 16 bit per database di piccola dimensione.

La tabella seguente mostra invece alcuni risultati con codice di lunghezza 64 relativo al database SIFT:

Metodo	***R=1***	***R=10***	***R=100***	***time cost (ms)***
SH	0.0953	0.2599	0.5303	84.828
PQ	0.2200	0.5840	0.9149	340.039
IVFPQ	Errore	Errore	Errore	Errore

Qui si vede chiaramente che per database di grosse dimensioni SH è molto più veloce di PQ. Purtroppo per le ridotte capacità della RAM abbiamo avuto un Memory error per quanto riguarda il metodo IVFPQ. La letteratura ci dice che IVFPQ è molto più veloce di SH e PQ.

8. CONCLUSIONI

Questo lavoro ha presentato soluzioni per l'indicizzazione su larga scala di dati ad alta dimensionalità. Vengono implementati 2 algoritmi di compressione SH e PQ, quest'ultimo con soluzione indicizzata IVF.

I risultati ottenuti hanno evidenziato solo alcuni aspetti dell'algoritmo di indicizzazione. In futuro, vogliamo continuare a migliorare l'algoritmo IVF.