TEXT RETRIEVAL

Query language: boolean, “Natural Language”, phrasal

Information retrieval vs Information filtering

IR models:

- Boolean
- Vector space
- Probabilistic

Term-document Incidence matrices
Incidence vectors

Incidence vectors

- Let’s now consider the query
- *BRUTUS* **AND** *CAESAR* **BUT CALPURNIA**

Same as NOT

110100
110111
010000

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Complemented!

110100 AND **110111** AND **101111** = 100100 -> Antony and Cleopatra, Hamlet

34

Fare la matrice è dispendioso, soprattutto in termini di spazio se consideriamo tutti gli 0 infatti la matrice è **sparsa**, sembrerebbe convenire usare direttamente i documenti. Invece se lavoriamo sulla matrice possiamo guadagnare molto.

*Ogni documento ha 1000 parole, quindi su una colonna ci saranno 1000 1, ed il tot degli 1 sarà N*W*

Sparsity

- The **total** number of 1s is given by *in all the matrix*

$$T_1 = N \times W = 1\text{G}$$

remember W is the number of words in a document

p.s. this is true if the words are distinct (we should talk about *types*)

N

	d1	d2	d3	d4	d5	d6	d7	d8	...
this	1	1	1	1	1	1	1	1	
the	1	1	1	1	1	1	1	1	
a	1	1	1	1	1	1	1	1	
over	1	1	0	0	1	1	0	1	
feature	0	0	0	0	0	1	0	0	
Matrix	1	0	0	0	0	0	0	0	
1917	0	0	0	0	0	0	0	0	
...									

W

37

Ci salviamo quindi il l’indice del documento che ha 1 nella matrice, es. il documento 6 avrà feature nel suo doc.

What's a better representation?

– We only record the 1 positions.

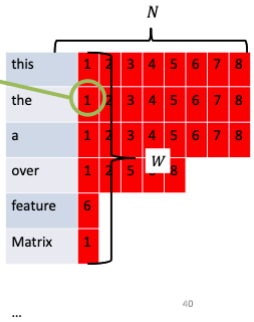
this	1	2	3	4	5	6	7	8
the	1	2	3	4	5	6	7	8
a	1	2	3	4	5	6	7	8
over	1	2	5	6	8			
feature	6							
Matrix	1							

DocID

- Warning the DocID can be very large!!!
- For addressing 1 million documents, we need 20bits (about 3Bytes)
  2^20 > 1Milion

3B

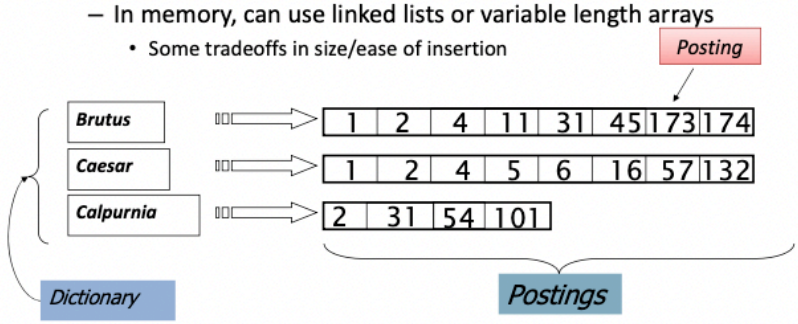
- The size of this representation is
  per salvare tutte le parole di tutti i documenti
  $3NW = 3 \times 10^6 \times 10^3 \approx$ 3GB
- In general the size of the inverted index is
- $N \times \log_2 N \times W$
- However, to compress the inverted index we use ***Variable Eyte Encoding***, which encode the gaps between the docIDs



40

Inverted index

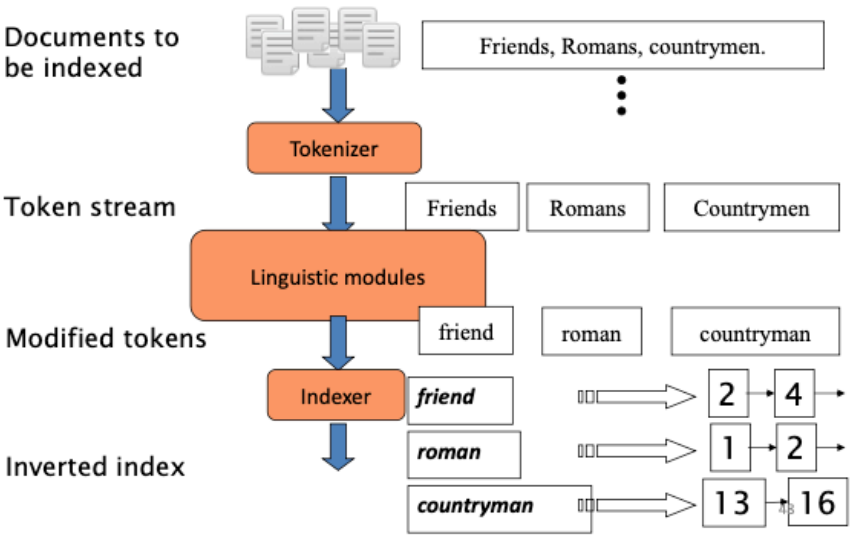
- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each doc by a **docID**, a document serial number
    - We need variable-size postings lists
      - On disk, a ***contiguous*** run of postings is normal and best (without explicit pointers) to minimize the size of the postings list and the number of disk seeks
      - In memory, can use linked lists or variable length arrays
        - Some tradeoffs in size/ease of insertion



43

Sorted by DocID (more later on why)

Inverted index construction



48

Per costruire l'indice di due documenti partiamo facendo tokenizer e applicando il linguistic model, successivamente ordiniamo in ordine alfabetico e otteniamo la tabella a sx.
Ci facciamo di conseguenza le posting list tenendo un contatore di frequenza associato al termine.

Ricordiamoci che il dizionario viene salvato in memoria mentre le posting list che sono molto più grandi in termini di spazio vengono salvate su disco.

n: dimensione collezione (es. tokens)

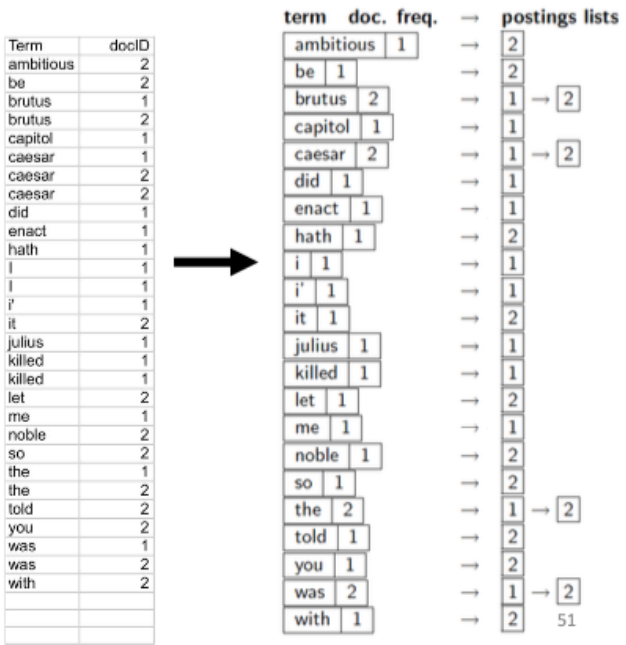
– Heaps' law

$$M = kT^b$$

– where *M* is the number of distinct words in an instance text of size *T*.
– k = 30-100 for English

b compreso tra 0 e 1 (0,5 in practice)

BOOLEAN QUERY PROCESSING



Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Query processing: AND

- Consider processing the query:

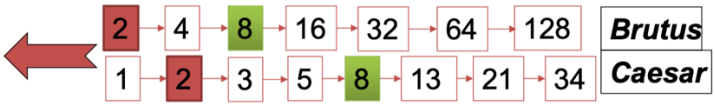
  Brutus *AND* ***Caesar***

  – Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.

  – Locate ***Caesar*** in the Dictionary;
    - Retrieve its postings.

  – "Merge" the two postings (intersect the document sets ***Brutus***∩***Caesar***):

In realtà le liste sono ordinate e fare il merge è meno complesso:

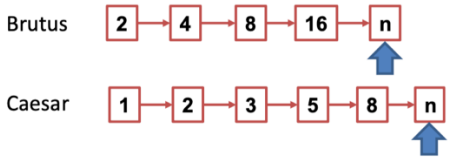


If the list lengths are *n* and *m*, the merge takes O(*n*+*m*) Operations

Or

- Brutus OR Caesar

Aggiungo al risultato ogni docID contenente Brutus o Cesar senza duplicatiduplicati



- ...and so on

Compexity again is O(*n*+*m*)

$A = \{1,2,3,4,5,8,16\}$

Possiamo **ottimizzare** l'AND partendo dalla lista più corta, in questo modo ci fermeremo prima facendo meno comparazioni.
Le boolean query possono rispondere a espressioni booleane.
Abbiamo un **problema** infatti spesso i risultati sono troppi pochi (AND) o troppi (OR).
In **ranked retrieval** andiamo a ritornare i documenti in ordine, viene spesso associato al **free text queries** dove andiamo a specificare come query una o più parole nel linguaggio umano, viene dato per ogni documento uno score tra 0 e 1 che misura quanto il documento matcha con la query.

TERM WEIGHTING

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a **count vector** in $\mathbb{N}^v$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	2	0	0
Caesar	232	227	0	2	1	0
Calpurnia	0	1	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	8	5	8
worser	2	0	1	1	1	5

La **term frequency $tf_{t,d}$** del termine t in un document d viene definito come il numero di volte che t occorre in d. Usiamo il tf quando computiamo il match score tra document e query.

Lo **score** viene definito come segue

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf}_{t,d}$$

Per ogni documento andiamo a trovare uno Score, che si calcola prendendo ogni termine t che appartiene sia al documento che alla query e andiamo a sommare la frequenza di questi termini.
Q = antony calpurnia
D = documento
T = termine (brutus, …)

	AC	JC	TT	H	O	M
Antony	157	73	0	0	0	1
Brutus	4	157	0	2	0	0
Caesar	232	227	0	2	1	0
Calpurnia	0	1	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	8	5	8
worser	2	0	1	1	1	5

Query = "Antony Calpurnia"

157 = 157 + 0	AC
74 = 73 + 1	JC
1 = 1 + 0	M

Per ogni documento andiamo a trovare uno Score, che si calcola prendendo ogni termine t che appartiene sia al documento che alla query e andiamo a sommare la frequenza di questi termini.
Q = antony calpurnia
D = documento
T = termine (brutus, …)

La **rilevanza** non incrementa proporzionalmente alla term frequency, se un termine appare 10 volte di più non vuol dire che è 10 volte più importante. A tal proposito andiamo ad utilizzare la **log-frequency weighting**

The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

0 → 0,
1 → 1,
2 → 1.3,
10 → 2,
1000 → 4,

Invece di usare la term frequency normale, andiamo a dare dei pesi w, calcolati con un log, in tal modo non vediamo un grandissimo cambiamento nello score anche se la frequenza cambia molto

Otteniamo lo score come $\mathrm{Score}(q,d) = \sum_{t \in q \cap d} (1 + \log_{10} \mathrm{tf}_{t,d})$

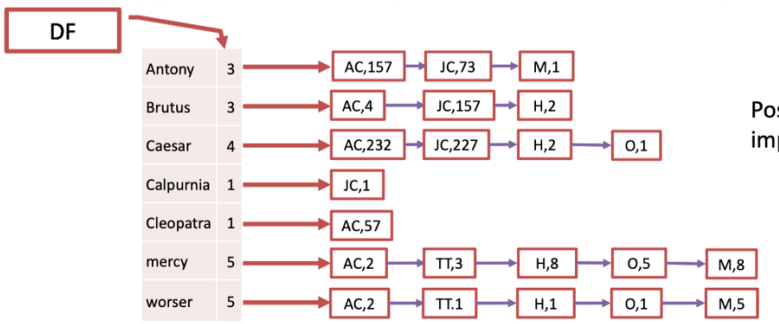
DOCUMENT FREQUENCY

I **rare item** sono più informativi rispetto alla term frequency, vogliamo a tal proposito dare un peso a questi termini rari.

Con **document frequency df_t** rispetto al termine t andiamo a definire il numero di documenti che contengono t.

	AC	JC	TT	H	O	M
Antony	157	73	0	0	0	1
Brutus	4	157	0	2	0	0
Caesar	232	227	0	2	1	0
Calpurnia	0	1	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	8	5	8
worser	2	0	1	1	1	5

«abstract» Vector representation



Posting list implementation

Df_t è l'inverso dell'informativeness di t, perché il termine è più importante se df è basso.

Ci definiamo quindi **inverse document frequency idf_t** come

$$\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$$

al solito usiamo il log per diminuire l'effetto di idf

Definiamo inoltre **tf-idf** di un termine t come il prodotto dei pesi del term frequency e dell'inverse document frequency.

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

$W_{t,d}$ aumenta al numero di occorrenze del termine in un documento (tf) e aumenta all'aumentare della rarità del termine tra tutti i documenti (idf).

Lo score di conseguenza sarà

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

Esempio

	idf	AC	JC	TT	H	O	M
Antony	log(6/3)	157	73	0	0	0	1
Brutus	log(6/3)	4	157	0	2	0	0
Caesar	log(6/4)	232	227	0	2	1	0
Calpurnia	log(6/1)	0	1	0	0	0	0
Cleopatra	log(6/1)	57	0	0	0	0	0
mercy	log(6/5)	2	0	3	8	5	8
worser	log(6/5)	2	0	1	1	1	5

$$\text{idf}_t = \log_{10}(N / \text{df}_t)$$

	idf	AC	JC	TT	H	O	M
Antony	0,30	0,96	0,86	0,00	0,00	0,00	0,30
Brutus	0,30	0,48	0,96	0,00	0,39	0,00	0,00
Caesar	0,18	0,61	0,60	0,00	0,23	0,18	0,00
Calpurnia	0,78	0,00	0,78	0,00	0,00	0,00	0,00
Cleopatra	0,78	2,15	0,00	0,00	0,00	0,00	0,00
mercy	0,08	0,10	0,00	0,12	0,15	0,14	0,15
worser	0,08	0,10	0,00	0,08	0,08	0,08	0,14

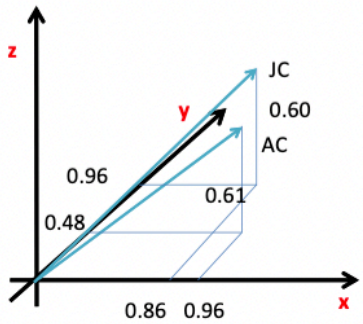
$$w_{t,d} = (1 + \log_{10} \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

VECTOR SPACE MODEL

Abbiamo uno spazio vettoriale di dimensione |M| (vocabolario), i documenti sono punti o vettori nello spazio, ovviamente siamo in high dimensional. I vettori sono molto sparsi infatti la maggior parte delle entry è a zero.

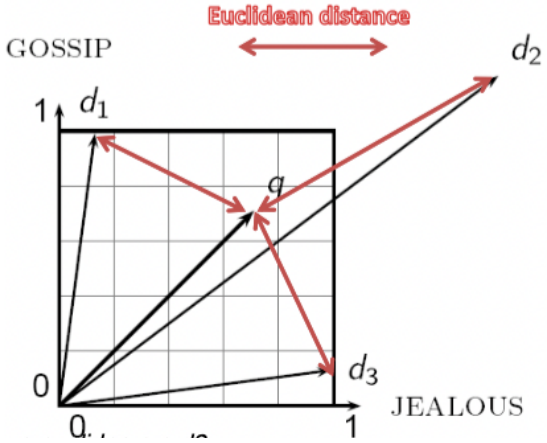
	AC	JC	TT	H	O	M	
Antony	0,96	0,86	0,00	0,00	0,00	0,30	x
Brutus	0,48	0,96	0,00	0,39	0,00	0,00	y
Caesar	0,61	0,60	0,00	0,23	0,18	0,00	z
Calpurnia	0,00	0,78	0,00	0,00	0,00	0,00	
Cleopatra	2,15	0,00	0,00	0,00	0,00	0,00	
mercy	0,10	0,00	0,12	0,15	0,14	0,15	
worser	0,10	0,00	0,08	0,08	0,08	0,14	

Vector 2



Le parole del vocabolario sono gli assi.

Non si può usare **l'euclidian distance** nell'high dimensionality



Usiamo invece invece **l'angolo** tra vettori che funziona meglio ma è ancora una distanza, introduciamo quindi una misura di similarità (**cosine similarity**).
Possiamo normalizzare con la L_2 norm formando veddori di lunghezza unitaria, adesso i documenti lunghi o corti che siano hanno pesi comparabili.

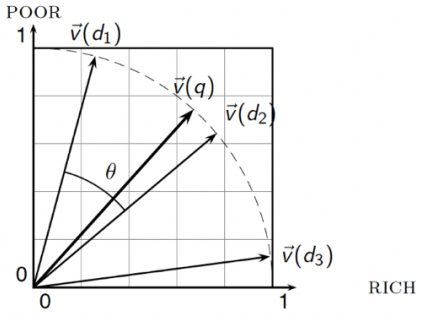
Dot product

Unit vectors

$$\cos(q,d) = \frac{q \cdot d}{\|q\|_2 \|d\|_2} = \frac{q}{\|q\|_2} \cdot \frac{d}{\|d\|_2} = \frac{\sum_i q_i d_i}{\sqrt{\sum_i q_i^2}\sqrt{\sum_i d_i^2}}$$

q_i is the tf-idf weight of term i in the query
d_i is the tf-idf weight of term i in the document

cos(q,d) is the cosine similarity of q and d … or, equivalently, the cosine of the angle between q and d.



Se ci pensiamo la normalizzazione sulla <u>query</u> non è necessaria perché è costante e non influisce sul rank.

Cosine example

w = 1+log(tf)

	idf	AC	JC	TT	H	O	M
Antony	0,30	3,20	2,86	0,00	0,00	0,00	1,00
Brutus	0,30	1,60	3,20	0,00	1,30	0,00	0,00
Caesar	0,18	3,37	3,36	0,00	1,30	1,00	0,00
Calpurnia	0,78	0,00	1,00	0,00	0,00	0,00	0,00
Cleopatra	0,78	2,76	0,00	0,00	0,00	0,00	0,00
mercy	0,08	1,30	0,00	1,48	1,90	1,70	1,90
worser	0,08	1,30	0,00	1,00	1,00	1,00	1,70
Norm		5,92	5,54	1,78	2,83	2,21	2,74

normalization cosine

Division (normalization)

$$w_{t,d} = \frac{1+\log_{10} tf_{t,d}}{\sqrt{\sum_{i=1}^{|d|}\left[1+\log_{10} tf_{i,d}\right]^2}}$$

w (t,d) della formula sopra

	idf	AC	JC	TT	H	O	M
Antony	0,30	0,54	0,52	0,00	0,00	0,00	0,36
Brutus	0,30	0,27	0,58	0,00	0,46	0,00	0,00
Caesar	0,18	0,57	0,61	0,00	0,46	0,45	0,00
Calpurnia	0,78	0,00	0,18	0,00	0,00	0,00	0,00
Cleopatra	0,78	0,47	0,00	0,00	0,00	0,00	0,00
mercy	0,08	0,22	0,00	0,83	0,67	0,77	0,69
worser	0,08	0,22	0,00	0,56	0,35	0,45	0,62

Non consideriamo ora l'idf perché all'aggiunta di un documento dovremmo ricalcolare tutti i pesi, appunto calcoliamo i pesi solo consideranto tf.

Vediamo come teniamo di conto dell'idf nella prossima img:

	idf	AC	JC	TT	H	O	M
Antony	0,30	0,54	0,52	0,00	0,00	0,00	0,36
Brutus	0,30	0,27	0,58	0,00	0,46	0,00	0,00
Caesar	0,18	0,57	0,61	0,00	0,46	0,45	0,00
Calpurnia	0,78	0,00	0,18	0,00	0,00	0,00	0,00
Cleopatra	0,78	0,47	0,00	0,00	0,00	0,00	0,00
mercy	0,08	0,22	0,00	0,83	0,67	0,77	0,69
worser	0,08	0,22	0,00	0,56	0,35	0,45	0,62

Query = "Antony Calpurnia"

$1*idf_{antony}$ 0.30/0.83 = 0.36

$1*idf_{calpurnia}$ 0.78/0.83 = 0.93

Query length = $\sqrt{0.30^2 + 0.78^2} \approx 0.83$

0.30/0.83
0
0
0.78/0.83
0
0
0

Final query vector

Ricordiamo che i d è già normalizzato nella pagina precedente, dobbiamo normalizzare anche q.

	idf	AC	JC	TT	H	O	M
Antony	0,30	0,54	0,52	0,00	0,00	0,00	0,36
Brutus	0,30	0,27	0,58	0,00	0,46	0,00	0,00
Caesar	0,18	0,57	0,61	0,00	0,46	0,45	0,00
Calpurnia	0,78	0,00	0,18	0,00	0,00	0,00	0,00
Cleopatra	0,78	0,47	0,00	0,00	0,00	0,00	0,00
mercy	0,08	0,22	0,00	0,83	0,67	0,77	0,69
worser	0,08	0,22	0,00	0,56	0,35	0,45	0,62

Query = "Antony Calpurnia"

$$\text{score} = \cos(q,d) = q \cdot d = \sum_i q_i d_i$$

0.30/0.83
0
0
0.78/0.83
0
0
0

q e d sono già normalizzati

	idf	AC	JC	TT	H	O	M
Antony	0,30	0,54	0,52	0,00	0,00	0,00	0,36
Brutus	0,30	0,27	0,58	0,00	0,46	0,00	0,00
Caesar	0,18	0,57	0,61	0,00	0,46	0,45	0,00
Calpurnia	0,78	0,00	0,18	0,00	0,00	0,00	0,00
Cleopatra	0,78	0,47	0,00	0,00	0,00	0,00	0,00
mercy	0,08	0,22	0,00	0,83	0,67	0,77	0,69
worser	0,08	0,22	0,00	0,56	0,35	0,45	0,62

Query = "Antony Calpurnia"

score_{JC} = 0.30*0.52/0.83+0.78*0.18/0.83 = **0.30**/0.83

0.30/0.83
0
0
0.78/0.83
0
0
0

For the purpose of ranking the documents the query normalization (/0.83) does not change the relative order of the documents

Come detto prima possiamo evitare di normalizzare la query perché è comune a tutti.

EVALUATION

- How fast does it index
  - Number of documents/hour
- How fast does it search Quanto è veloce a restituire il risultato
  - Latency as a function of index size
- Expressiveness of query language
  - Ability to express complex information needs
  - Speed on complex queries Capacità di poter utilizzare query complesse senza intaccare le performace
- User-friendly UI
- Is it free?

Lucene

Scalable, High-Performance Indexing

- over 150GB/hour on modern hardware
- small RAM requirements -- only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20-30% the size of text indexed

I criteri detti prima sono misurabili ma l'importante è la soddisfazione dell'utente che non tiene di conto solo della velocità di risposta ma anche della sua qualità. Dobbiamo trovare un modo per valutare la **happiness**.
La misura di rilevanza si basa su:

- Benchmark sulla **collezione** dei documenti
- Benchmark su un set di **query**
- Binary **assessment** per relevant o not relevant per ogni query e ogni documento

La rilevanza non si bassa sulla query ma sull'**informazione necessaria**, per esempio per info necessaria possiamo intendere bere vino rosso riduce di più attacchi di cuore del vino bianco. Le **query** potrebbero essere per esempio:

- Vino rosso bianco attacco cuore
- Vino AND rosso AND bianco AND attacco AND cuore

Quindi date le query andiamo a prendere i documenti e valutiamo i documenti in base all'informazione necessaria per poi definire se i documenti ritornati sono rilevanti o meno.

METRICHE

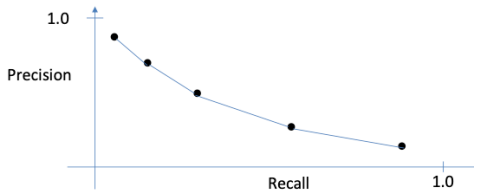
$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$$

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|}$$

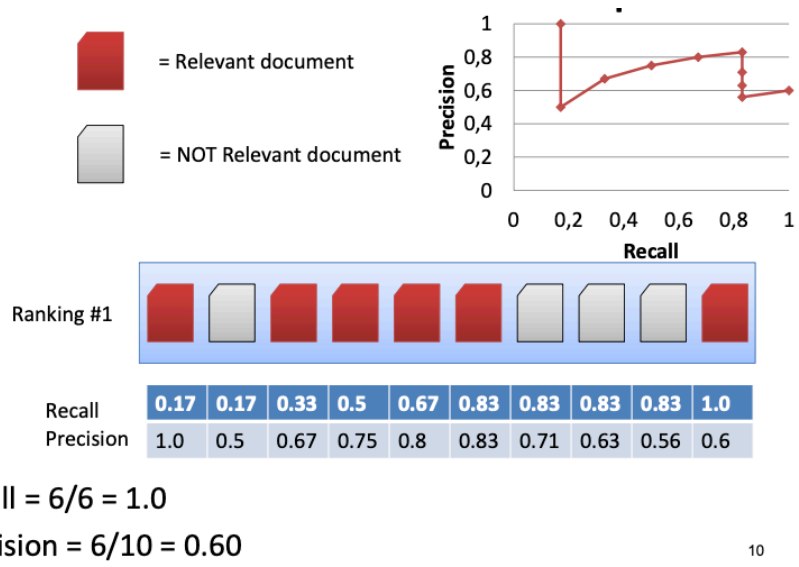
$$Accuracy = \frac{|(Relevant \cap Retrieved) \cup (\neg Relevant \cap \neg Retrieved)|}{\#Alldocs}$$

Con l'accuratezza come metrica di valutazione per IR c'è il solito problema che se la parte a dx del numeratore è molto grande e la parte a sx è 0, abbiamo comunque un'alta accuratezza.

Quando si fa una query quanti documenti devo ritornare? Se tutti allora avrò alta recall e bassa precision. Se consideriamo un numero di documenti ritornati incrementale possiamo produrre una curva **precision-recall.**



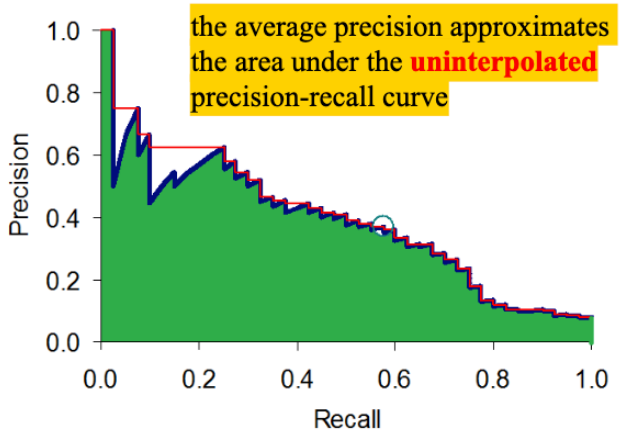
In un buon sistema la precisione diminuisce all'aumentare dei documenti ritornati mentre la recall aumenta.



Possiamo usare **F1metric** che è una metrica combinata.

MAP (Mean Average Precision)

Fornisce una visione compatta della qualità del retrieval usando un set di query.



Data un'informazione necessaria q, abbiamo m documenti rilevanti $\{d_1,\ldots,d_m\}$, R_k è il set ordinato dei risultati ritornati

$$AP(q) = \frac{1}{m}\sum_{k=1}^{m} Precision(R_k)$$

$$Precision(R_k) = \begin{cases} 1 \text{ if } d_k \text{ is relevant} \\ 0 \text{ if } d_k \text{ is not relevant} \end{cases}$$

Evaluation Metrics: Example

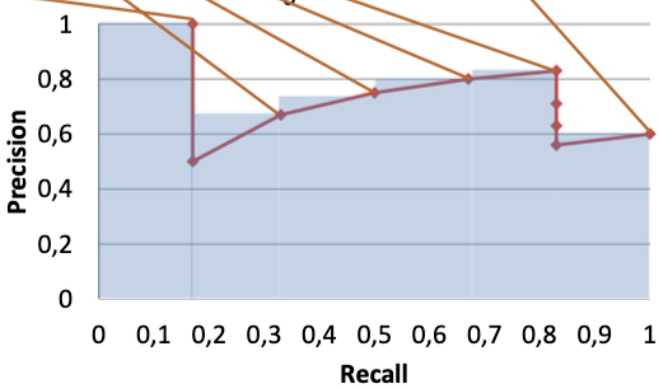
Ranking #1										
Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83	0.83	0.83	1.0
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71	0.63	0.56	0.6

$$AP = \frac{1.0 + 0 + 0.67 + 0.75 + 0.8 + 0.83 + 0 + 0 + 0 + 0.6}{6} = 0.78$$

- … and divide the result by the **total number of relevant** documents.

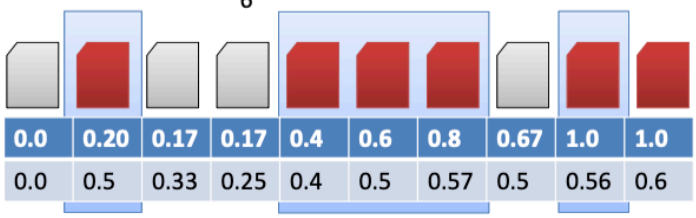
il numero totale di relevent document è indicato con m nella slide precedente

$$AP = \frac{\boxed{1.0} + 0 + \boxed{0.67} + \boxed{0.75} + \boxed{0.8} + \boxed{0.83} + 0 + 0 + 0 + \boxed{0.6}}{6} = 0.78$$



Ranking #1										
Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83	0.83	0.83	1.0
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71	0.63	0.56	0.6

$$AP = \frac{1.0 + 0 + 0.67 + 0.75 + 0.8 + 0.83 + 0 + 0 + 0 + 0.6}{6} = 0.78$$



Ranking #2										
Recall	0.0	0.20	0.17	0.17	0.4	0.6	0.8	0.67	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.5	0.57	0.5	0.56	0.6

$$AP = \frac{0 + 0.5 + 0 + 0 + 0.4 + 0.5 + 0.57 + 0 + 0.56}{5} = 0.51$$

mean average precision = MAP = $\frac{0.78 + 0.51}{2} = 0.64$

- We have a set *Q* of information needs $\{q_1, \dots, q_{|Q|}\}$, and for the $q_j \in Q$ we have m_j relevant documents $\{d_1, \dots, d_{m_j}\}$, R_{jk} is the set of ranked retrieval results from the top result until you get to document *k-th* d_k, then

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \underbrace{\boxed{\frac{1}{m_j} \sum_{k=1}^{m} Precision(R_k)}}_{AP(q_j)}$$

Approximate Nearest Neghbors

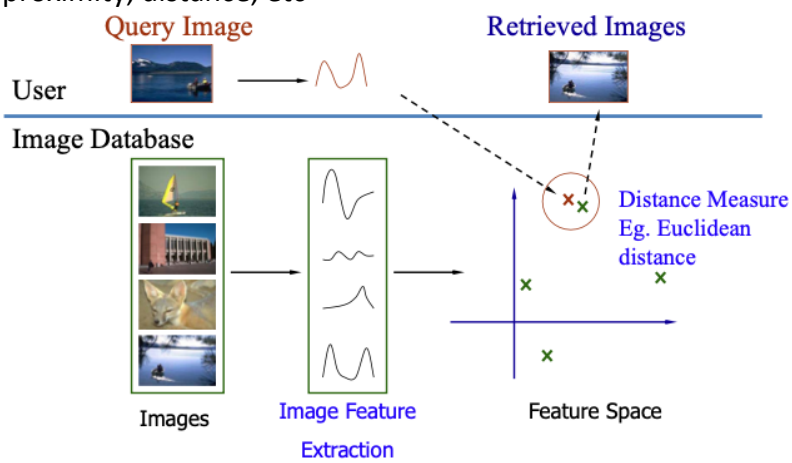
- Intro: Text-based image retrieval vs Content-based image retrieval
- Similarity search
  - Similarity search & query-by-example
  - Global & local Features
  - Query paradigms
  - Metric space
  - Exact approach
  - Approximate approach: permutation-based approach
- CBIR through standard text retrieval techniques
  - Surrogate Text Representation
  - Reordering
  - Other approaches

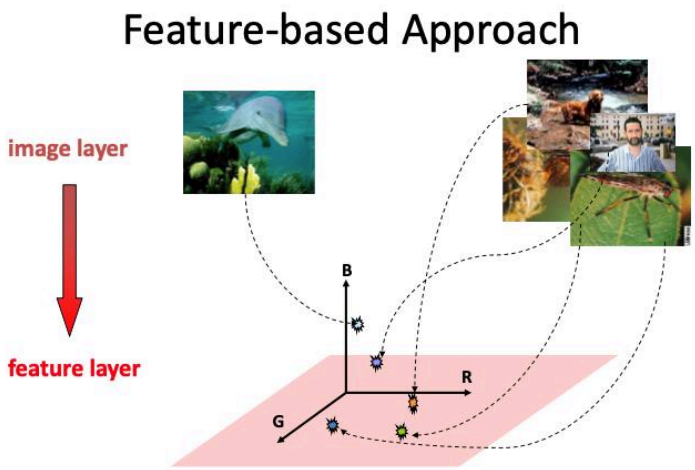
Metadata-Driven Image Retrievial

Il search engine può basare la sua ricercar in modo tradizionale utilizzando i metadati dell'immagine, ciò prevede però un'annotazione manuale delle immagini ed usa dei metodi di text-based retrievial. Abbiamo però dei **problemi** nell'annotazione manuale: si basa su scelte umane e sono valide solo per una lingua (per l'image retrievial questa limitazione non dovrebbe esistere). Inoltre, la scelta dell'uomo può essere soggettiva ed inoltre viene data troppa responsabilità a chi deve scegliere. Possono presentarsi anche problemi di ambiguità, come succede ad esempio se cerchiamo "Apple" ci viene restituito sia il logo che una mela vera. Infine, abbiamo archivi contenenti immagini che non sono ben descritti e quindi non possono essere utilizzati nel text-based retrievial, a meno che non vengano annotate, un esempio può essere ad esempio la foto di un monumento con su scritto il nome della città, veramente inutile!

Content based image retrievals

Tramite l'approccio CBIR andiamo ad analizzare il **contenuto** dell'immagine, ottenendo feature più effettive per descrivere l'immagine invece di usare metadata definiti dallo user. L'immagine è ricca di contenuti come ad esempio texture, colori, forme e altre info che possono essere derivate direttamente. Quando viene data una query (immagine di cui vogliamo trovare le simili) il sistema estrae le features e viene ritornate le **nearest neighbors** usando delle similarity measure. Un esempio può essere confrontare gli istogrammi dei colori nel HSV color space per valutare la somiglianza tra immagini.
Notiamo che il **paradigma di ricerca** è cambiato: siamo passati dal tradizionale SI-NO con dominio dei dati sortable ad una *comparazione graduale* e/o ranking basato su similarity, dissimilarity, proximity, distance, etc





Nelle due immagini sopra vediamo che vengono estratte features dalle immagini, esse poi vengono rappresentate nello spazio delle features per poi utilizzare delle misure della distanza per ritornare immagini simili (le più vicine).

Global and Local Features

Per **global features** si intende:

- La rappresentazione matematica dell'apparenza visiva di tutta l'immagine (colori, textures, edges).
- Le immagini possono essere comparate usando una funzione di distanza o (dis)similarità tra le features.

Per **local features** si intende:

- La rappresentazione matematica dell'apparenza visiva di specifiche aree di interesse dell'immagine.
- La misura della distanza o (dis)similarità può essere calcolata tra i punti di interesse nell'immagine.
- Le immagini possono essere comparate analizzando i matching points.

Global Features

Chiamate anche global descriptor oppure visual descriptor, abbiamo un descrittore (es un vettore) per immagine, andiamo a catturare l'intero contenuto dell'immagine.
Degli esempi di global features sono: istogrammi di colori, frequanza o quantizzazione dei valori RGB.
Le deep CNN sono CNN composte da molti layer, una ben conosciuta deep CNN è AlexNet, è stata addestrata a riconoscere 1000 concetti da ImageNet. A seguito due caratteristiche delle deep CNN:

- Se due immagini date in input ad AlexNet sono simili allora lo stato interno dei layers della rete è simile
- Solitamente il vettore a virgola mobile (floating point vector) deve essere normalizzato alla lunghezza unitaria e successivamente viene comparato utilizzando la L_2 distance.

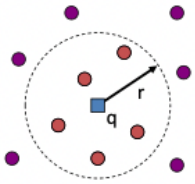
Local Features

Chiamati anche local descriptors, ce ne sono molti per ogni immagine, sono invarianti a trasformazioni, cambi di illuminazione, rotazione, ecc.

Similarity Rand Query

$$R(q,r) = \{x \in X \mid d(q,x) \leq r\}$$

Tutte le immagini che hanno distanza dalla query più piccola di …



kNN Query

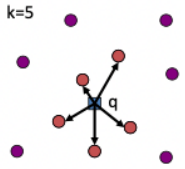
the nearest neighbor query

- *1NN*(q) = x
- x ∈ X, ∀y ∈ X, d(q,x) ≤ d(q,y)

k-nearest neighbors query

- *kNN*(q,k) = A
- A ⊆ X, |A| = k
- ∀x ∈ A, y ∈ X – A, d(q,x) ≤ d(q,y)

Le 5 immagini più vicine alla query



Metric Space

- M = (D,d)
  - Data domain D
  - Total (distance) function d: D × D → R (metric function or metric)
- The metric space postulates:
  - Non-negativity $\forall x, y \in \mathcal{D}, d(x,y) \geq 0$
  - Symmetry $\forall x, y \in \mathcal{D}, d(x,y) = d(y,x)$
  - Identity $\forall x, y \in \mathcal{D}, x = y \Leftrightarrow d(x,y) = 0$
  - Triangle inequality $\forall x, y, z \in \mathcal{D}, d(x,z) \leq d(x,y) + d(y,z)$

This must hold true!

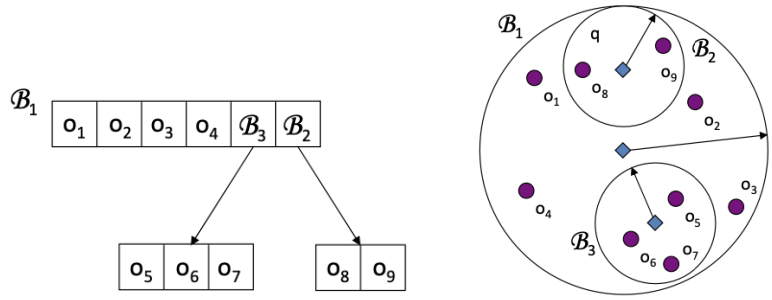
L'identità può non essere vera in casi particolari, la simmetria solitamente è vera ma non sempre, la non-negativity deve essere sempre vera.

Index Organization

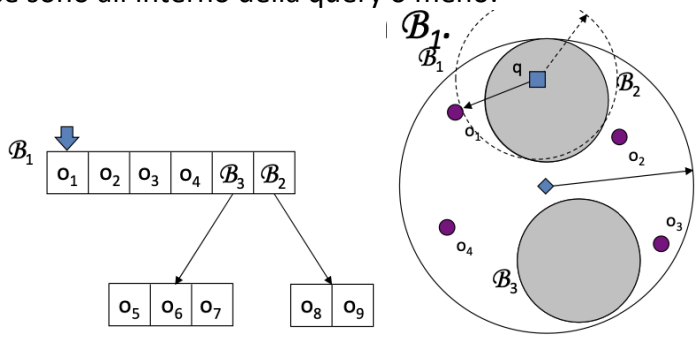
Ogni cosa che vediamo, sentiamo, leggiamo può essere in formato digitale, nei prossimi anni creeremo una quantità di dati immensa e sempre più grande. Necessitiamo quindi di una organizzazione tramite indici in modo tale che la ricerca sia più efficiente.
Gli indici metrici usano solo gli assiomi metrici per organizzare gli oggetti e sfruttano la disuguaglianza triangolare per sfoltire (prune) lo spazio di ricerca.

Tree based index organization

Viene fatta una decomposizione gerarchica dello spazio tramite delle **ball regions**, il root node organizza oggetti e qualche regione, anche i nodi interni fanno lo stesso.



Data una query R(q,r) dove q è il punto e r il raggio, vogliamo trovare tutti gli elementi che appartengono a tale query. Partiamo ispezionando gli elementi di B1 e per ognuno controlliamo se se sono all'interno della query o meno.

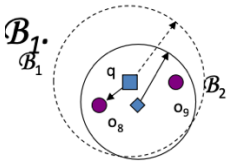


Arriviamo fino ad o4 e nessuno di essi per ora è all'interno della query, quindi nessuno di essi viene scelto.

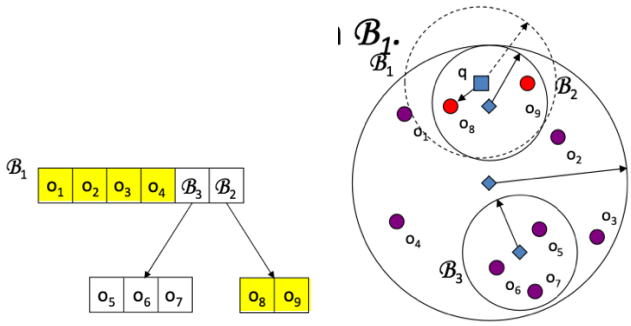
Adesso dobbiamo esaminare le ball B3 e B2, prima di esaminarle internamente andiamo a vedere se le ball intersecano la query perché se così non fosse sarebbe inutile andare a vedere se i punti interni alla ball rientrano nella query, questo viene controllato utilizzando la disuguaglianza triangolare.

$$o \in \mathfrak{B} \wedge \mathfrak{B} \cap Q = \emptyset \rightarrow o \notin Q$$

Vediamo appunto che B3 non interseca mentre B2 si, ispezioniamo quindi solo gli elementi di B2. Entrambi gli elementi o8 e o9 appartengono alla query e vengono quindi insertiti nella response.



Vediamo sotto in giallo gli elementi controllati mentre i punti rossi sono la risposta alla ricerca.



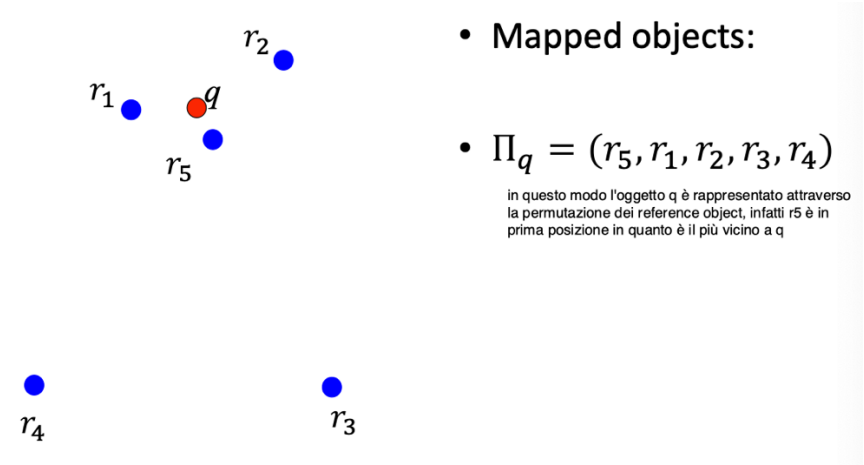
Approximate similarity search

La ricerca approssimata per similarità supera i problemi di ricerca esatta quando si usano metodi di accesso tradizionali. Possiamo avere problemi dimensionality curse ed un moderato incremento delle perfomance rispetto allo scan sequenziale.
E' ragionevole in molte applicazioni fare una ricerca di similarità approssimata perché la **feature extraction** o la modellizzazione del metric space già fanno un'**approssimazione della realtà**, quindi, fare una seconda approssimazione è solitamente accettabile.

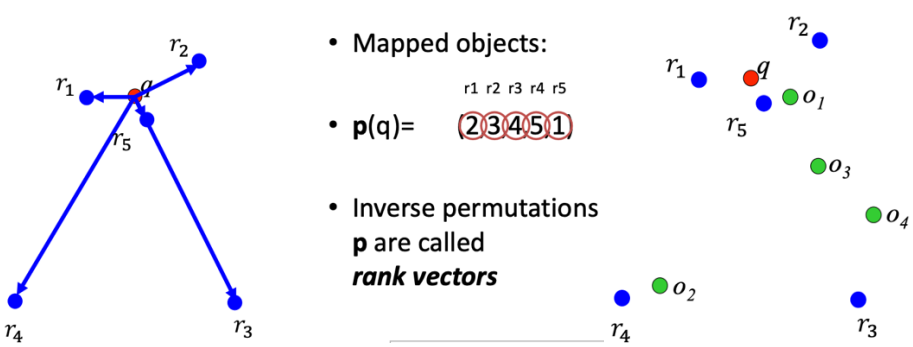
Surrogate text representations for metric space

Permutation based index

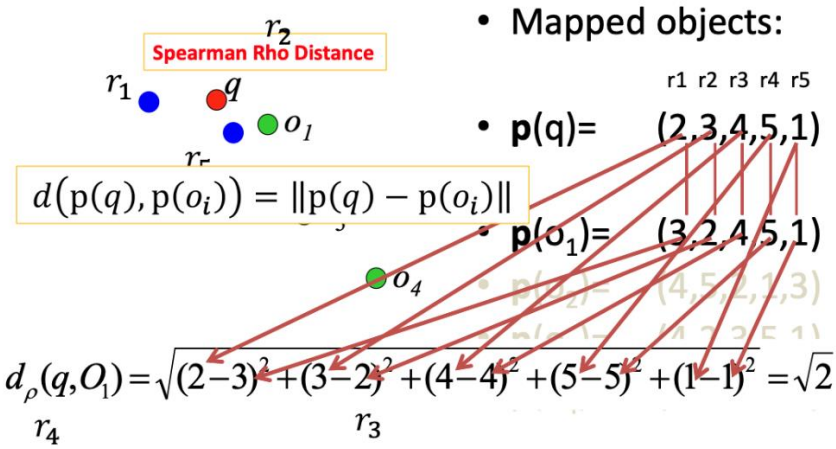
Ci si basa sul concetto che oggetti vicino nello spazio vedono lo spazio in modo simile, possiamo rappresentare ogni data object o come un insieme ordinato di **Reference Object** (random object called also Pivots) a seconda della distanza da o. Possiamo misurare la similarità tra due data object misurando la similarità tra l'ordine dei Reference Object.
Sotto vediamo che q viene rappresentato come la permutazione dei RO ordinati in base alla distanza.



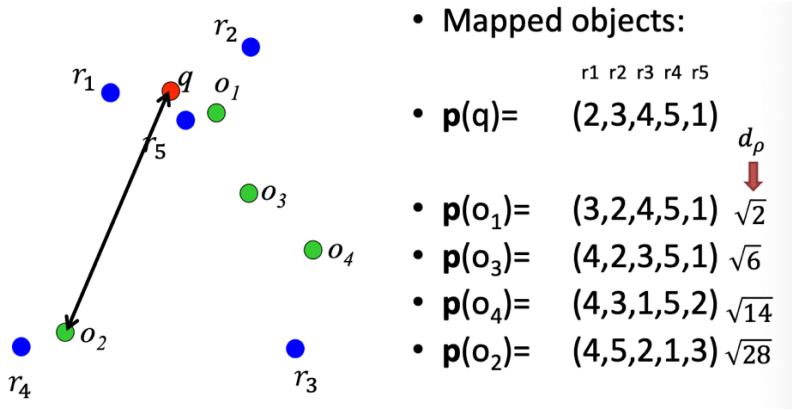
Sotto vedimo **l'inverse permutation**: con questo approccio si stabilisce a priori un ordine dei reference object e si riempie il vettore inserendo il numero nella cella corrispondente. Dato che r5 è il più vicino inserirò 1 nella casella corrispondente a r5.



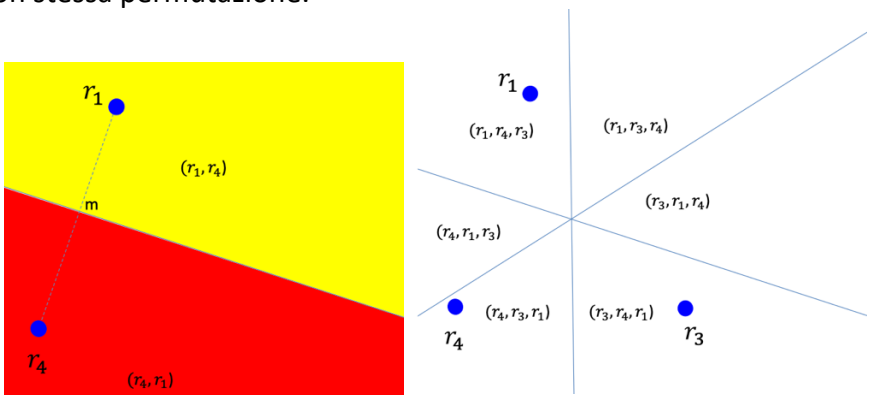
Vediamo adesso la **Spearman Rho Distance**: vado a calcolare la distanza fra le permutazioni per vedere quanto sono differenti, più è bassa la distanza e più gli oggetti sono vicini nello spazio.



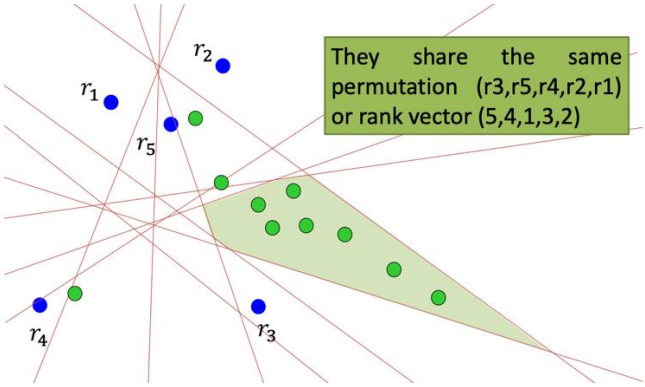
Calcoliamo la Rho Distance per ogni oggetto, il più vicino è appunto o1 mentre il più lontano è o2.



Se prendiamo un caso estremamente semplice con due soli RO, avremo che tutti gli oggetti che cadono in una delle due zone avranno la stessa permutazione. Aggiungendo RO aumenteranno anche le zone con stessa permutazione.



Ovviamente più RO aggiungiamo e più le regioni saranno piccole ed approssimeranno meglio gli oggetti.



Towards indexing with text retrieval techniques

La permutazione può essere salvata facilmente in un **inverted file**, dove i RO rappresentano le parole del **dizionario** e gli oggetti sono invece i documenti (**posting list**).
Paragoniamo la query a tutti gli oggetti per vedere quindi il più simile.

	r1	r2	r3	r4	r5
O_1 =	3	2	4	5	1
O_2 =	4	5	2	1	3
O_3 =	4	2	3	5	1
O_4 =	4	3	1	5	2

q		O_1	O_2	O_3	O_4
2	r_1 ->	3	4	4	4
3	r_2 ->	2	5	2	3
4	r_3 ->	4	2	3	1
5	r_4 ->	5	1	5	5
1	r_5 ->	1	3	1	2

Expolting standard search engine

Vogliamo usare un motore standard per full text search, come ad esempio Lucene, e vogliamo ritornare i documenti in base alla media della cosine similarity. I motori di ricerca usano **term-frequency** come meccanismo principale per pesare i termini nel documento (no Inverse Document Frequency), i **termini**, nel contesto delle permutazioni, rappresentano i **reference object**.
La cosine similarity promuove i termini con **high term-frequency**, vogliamo privilegiare i termini (reference object) con **alto rank**.
Alto rank -> alta frequenza Complementiamo i rank!
In poche parole, la cosine similarity privilegia chi ha il valore più alto (termine più frequente), dato che noi abbiamo associato i valori più piccoli a chi è più vicino allora basta complementare i rank così la cosine privilegerà gli oggetti più vicini.

Mapped objects:

r1 r2 r3 r4 r5

p(q)= (2,3,4,5,1)

Reference object

m = number of pivots

$\boldsymbol{m} = (m+1, \dots, m+1) =$
$= (6,6,6,6,6)$

$$\overline{\boldsymbol{p}}(q) = \boldsymbol{m} - \boldsymbol{p}(q) = (6-2, 6-3, 6-4, 6-5, 6-1) = (4,3,2,1,5)$$

Mapped objects:

r1 r2 r3 r4 r5

$\boldsymbol{p}(q) = (2,3,4,5,1)$

$\boldsymbol{p}(o_1) = (3,2,4,5,1)$
$\boldsymbol{p}(o_3) = (4,2,3,5,1)$
$\boldsymbol{p}(o_4) = (4,3,1,5,2)$
$\boldsymbol{p}(o_2) = (4,5,2,1,3)$

$\overline{\boldsymbol{p}}(q) = \boldsymbol{m} - p(q) = (4,3,2,1,5)$

$\overline{\boldsymbol{p}}(o_1) = \boldsymbol{m} - \boldsymbol{p}(o_1) = (3,4,2,1,5)$
$\overline{\boldsymbol{p}}(o_3) = \boldsymbol{m} - \boldsymbol{p}(o_3) = (2,4,3,1,5)$
$\overline{\boldsymbol{p}}(o_4) = \boldsymbol{m} - \boldsymbol{p}(o_4) = (2,3,5,1,4)$
$\overline{\boldsymbol{p}}(o_2) = \boldsymbol{m} - \boldsymbol{p}(o_2) = (2,1,4,5,3)$

Dato che i search engine usano la cosine similarity, cosa succede se la applichiamo ai vettori complementati? Dato che quando facciamo il knn non siamo interessati al valore di similarità ma al ranking la norma al denominatore è costante e può essere ignorata

$$sim(\overline{\boldsymbol{p}}(q), \overline{\boldsymbol{p}}(o_i)) = \frac{\overline{\boldsymbol{p}}(q) * \overline{\boldsymbol{p}}(o_i)}{\|\overline{\boldsymbol{p}}(q)\| \|\overline{\boldsymbol{p}}(o_i)\|}$$

For instance for $m = 5$,we have

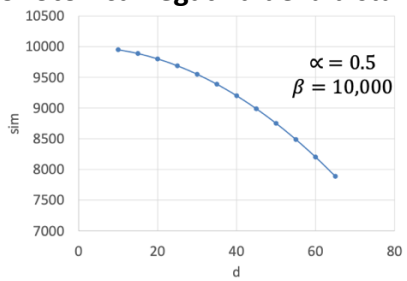
$$\|\overline{\boldsymbol{p}}(q)\| = \|\overline{\boldsymbol{p}}(o_i)\| =$$
$$= \sqrt{1^2 + 2^2 + 3^2 + 4^2 + 5^2} = \sqrt{55}$$
$$= const$$

Vediamo sotto che i rank utilizzando l'euclidian distance sono i medesimi ottenuti con la cosine similarity.

Complemented rank vectors			rank vectors		$\|p(q) - p(o_i)\|$
$\overline{\boldsymbol{p}}(q)$ = (4,3,2,1,5)			$\boldsymbol{p}(q)$ = (2,3,4,5,1)		
$\overline{\boldsymbol{p}}(o_1)$ = (3,4,2,1,5)	54		$\boldsymbol{p}(o_1)$ = (3,2,4,5,1)		$\sqrt{2}$
$\overline{\boldsymbol{p}}(o_3)$ = (2,4,3,1,5)	52		$\boldsymbol{p}(o_3)$ = (4,2,3,5,1)		$\sqrt{6}$
$\overline{\boldsymbol{p}}(o_4)$ = (2,3,5,1,4)	48		$\boldsymbol{p}(o_4)$ = (4,3,1,5,2)		$\sqrt{14}$
$\overline{\boldsymbol{p}}(o_2)$ = (2,1,4,5,3)	**39**		$\boldsymbol{p}(o_2)$ = (4,5,2,1,3)		$\sqrt{28}$
	$\overline{\boldsymbol{p}}(q) * \overline{\boldsymbol{p}}(o_i)$				

La cosine similarity è una **trasformazione monotonica negativa** della distanza Eucliudea

$$sim(\overline{\boldsymbol{p}}(q), \overline{\boldsymbol{p}}(o_i)) = \overset{\text{const}}{\alpha} - \beta \|\boldsymbol{p}(q) - \boldsymbol{p}(o_i)\|^2$$



Ora sappiamo che la **similarity** di due rank vectors può essere ottenuta dal **prodotto scalare** dei corrispondenti rank vectors **complementati.**
I sistemi di text IR usano il prodotto scalare (cosine) per calcolare la similarità di **vetors term-frequencies**, quindi, possiamo vedere i rank complementati come TF.

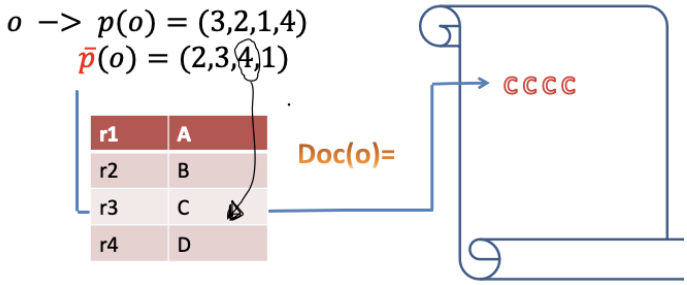
Step for Surrogate text representation

1. Assegnare uan keyword per ogni reference object e creiamo un dizionario chiamato **codebook**:

Rerefence obj	codeword
r1	A
r2	B
r3	C
r4	D

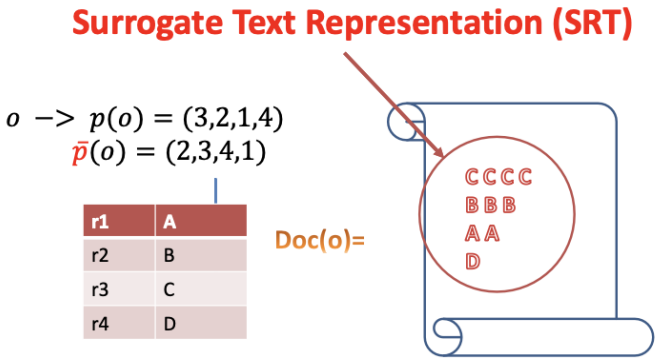
codebook

2. Per ogni oggetto creiamo un documento testuale **ripetendo** la codeword del reference object basandosi sul ranking:



Il vettore complementato può essere ricondotto alla TF come spiegato precedentemente; quindi, i valori di questo vettore rappresentano il numero di volte che va ripetuta una keyword.

3. Il documento generato è chiamato **surrogate text representation**



4. Indicizziamo questi documenti (uno per ogni oggetto) e usiamo la stessa trasformazione per la query, così da utilizzarla per ricercare oggetti simili

```
O1 = A A A A B C C C G G
O2 = A A A C C C C D E E
O3 = B B B C C D F F F F
O4 = A A A D D E H H H H
O5 = A A B B B B C E E E
O6 = A A A A E G G H H H
O7 = A A A A B B B G G H
O8 = B B B C C C C D E E
O9 = A A A B D D D D E E
```

Optimazing in space the inverted file

Solitamente la matrice di un inverted file di un testo completo è sparsa perché il dizionario è grande e quindi ci saranno tanti 0, nel nostro caso è tutta riempita perché ci inseriamo il rank che non ha zeri (full matrix!), vogliamo quindi ottenere un dizionario piccolo e dove ogni documento contiene tutti i suoi termini.

Example *m=8, rank vectors*

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	1	2	5	2	3	1	1	5	2
r2	4	6	2	5	1	6	2	2	4
r3	2	1	3	6	4	8	6	1	8
r4	5	4	4	3	5	5	5	4	1
r5	7	3	6	4	2	4	7	3	3
r6	6	5	1	7	6	7	8	7	7
r7	3	7	8	8	7	3	3	8	5
r8	8	8	7	1	8	2	4	6	6

***Complemented** rank vectors: $m+1-o_i$*

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	8	7	4	7	6	8	8	4	7
r2	5	3	7	4	8	3	7	7	5
r3	7	8	6	3	5	1	3	8	1
r4	4	5	5	6	4	4	4	5	8
r5	2	6	3	5	7	5	2	6	6
r6	3	4	8	2	3	2	1	2	2
r7	6	2	1	1	2	6	6	1	4
r8	1	1	2	8	1	7	5	3	3

It is a Full matrix!

Per ridurre la dimensione dell'inverted index possiamo sfruttare l'idea di prendere solo i **k_x reference objects più vicini** per rappresentare ogni oggetto che deve essere indexato, ovviamente $k_x << m$.

In pratica usiamo un ranking parziale che usa le cosiddette **distanze con parametro di posizione k_x** (distance wirt location parameter k_x), quello che si fa è lasciare come sono i primi k_x elementi ed assegnare k_x+1 ai seguenti

For example for $k_x = 3$

- Permutation — **p**(o_1)=(3,2,4,5,1)
- Partial permutation — **p**(o_1)=(3,2,**4**,**4**,1)

Example *m=8, k_x = 4, if value > 4 put 5*

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	1	2	5	2	3	1	1	5	2
r2	4	5	2	5	1	5	2	2	4
r3	2	1	3	5	4	5	5	1	5
r4	5	4	4	3	5	5	5	4	1
r5	5	3	5	4	2	4	5	3	3
r6	5	5	1	5	5	5	5	5	5
r7	3	5	5	5	5	3	3	5	5
r8	5	5	5	1	5	2	4	5	5

Ora dobbiamo complementare i rank e far si che i 5 diventino 0 (basta sottrarre la posizione dell'oggetto o_i)

Example *m=8, k_x = 4*

complement the ranks into the inverted file k_x+1-o_i ← posizione dell'oggetto

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	4	3	0	3	2	4	4	0	3
r2	1	0	3	0	4	0	3	3	1
r3	3	4	2	0	1	0	0	4	0
r4	0	1	1	2	0	0	0	1	4
r5	0	2	0	1	3	1	0	2	2
r6	0	0	4	0	0	0	0	0	0
r7	2	0	0	0	0	2	2	0	0
r8	0	0	0	4	0	3	1	0	0

1	4
2	3
3	2
4	1

5	0
6	0
7	0
8	0

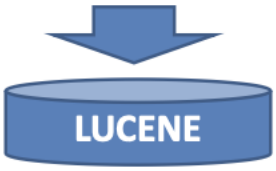
È evidente che abbiamo risparmiato molto spazio dato che nell'inverted index gli 0 non sono considerati per lo storage!
Definiamo quindi un **codebook** (dictionary) di pivots (reference objects), ad esempio:

r1	A
r2	B
r3	C
r4	D
r5	E
r6	F
r7	G
r8	H

E generiamo il **surrogate text representation** delle feature dell'immagine:

	o1	o2	o3	o4	o5	o6	o7	o8	o9
A	4	3	0	3	2	4	4	0	3
B	1	0	3	0	4	0	3	3	1
C	3	4	2	0	1	0	0	4	0
D	0	1	1	2	0	0	0	1	4
E	0	2	0	1	3	1	0	2	2
F	0	0	4	0	0	0	0	0	0
G	2	0	0	0	0	2	2	0	0
H	0	0	0	4	0	3	1	0	0

```
o1 = A A A A B C C C G G
o2 = A A A C C C C D E E
o3 = B B B C C D F F F F
o4 = A A A D D E H H H H
o5 = A A B B B B C E E E
o6 = A A A A E G G H H H
o7 = A A A A B B B G G H
o8 = B B B C C C C D E E
o9 = A A A B D D D D E E
```



Optimazing in space the inverted file

Si consiglia un minimo numero di reference object $m = 2\sqrt{N}$ per ottenere un buon risultato, N è il numero di immagini.
Ad esempio con N=100milioni allora abbiamo RO pari a m=20K, la dimensione dell'inverted file sarebbe m*N=2T items, se invece aggiungiamo k_x=50 la dimensione è drasticamente inferiore k_x*N=50*100M=5G items.

Facciamo ora un esempio nel quale data una query vogliamo trovare il ranking rispetto agli oggetti utilizzando la distanza euclidea e la cosine similarity e vogliamo verificare che siano identici anche se il parametro K è differente: K_x=4 K_q=3

$d(p(q), p(o))^2$

	o1	o2	o3	o4	o5	o6	o7	o8	o9	q
r1	1	2	5	2	3	1	1	5	2	2
r2	4	5	2	5	1	5	2	2	4	4
r3	2	1	3	5	4	5	5	1	5	1
r4	5	4	4	3	5	5	5	4	1	4
r5	5	3	5	4	2	4	5	3	3	3
r6	5	5	1	5	5	5	5	5	5	4
r7	3	5	5	5	5	3	3	5	5	4
r8	5	5	5	1	5	2	4	5	5	4
d^2	10	4	32	30	24	26	28	16	28	

ranking

2
1
8
5
6
7
9
4
3

- The result set is:
- o2, o1, o8, o5, o6, o7, o9, o4, o3

$\bar{p}(q) * \bar{p}(o)$

	o1	o2	o3	o4	o5	o6	o7	o	q
r1	4	3	0	3	2	4	4	0	2
r2	1	0	3	0	4	0	3	3	0
r3	3	4	2	0	1	0	0	4	3
r4	0	1	1	2	0	0	0	1	0
r5	0	2	0	1	3	1	0	2	1
r6	0	0	4	0	0	0	0	0	0
r7	2	0	0	0	0	2	2	0	0
r8	0	0	0	4	0	3	1	0	0
*	17	20	6	7	10	9	8	14	

Same ranking!

2
1
8
5
6
7
9
4
3

- The result set is:
- o2, o1, o8, o5, o6, o7, o9, o4, o3

Effettivamente sono uguali, vediamo il perché: sulle slide ANN da slide 110.

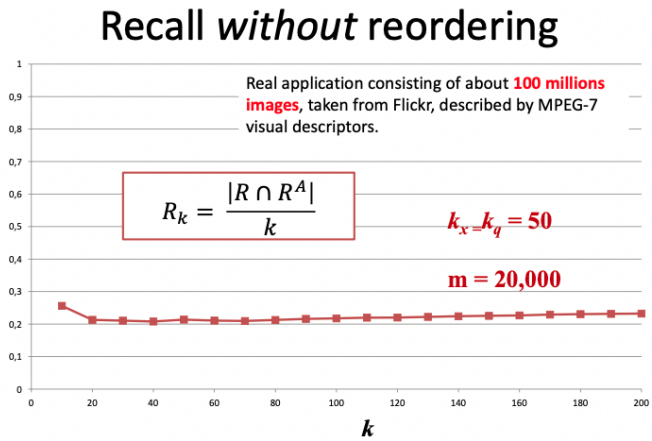
Reordering

L'approccio appena visto permette di ottenere buoni risultati sotto il punto di vista dell'efficienza ma nelle applicazioni reali manca di effectiveness. Ad esempio, se valutiamo il numero di risultati corretti rispetto alla variazione di k, avremmo recall @ k

$$R_k = \frac{|R \cap R^A|}{k}$$

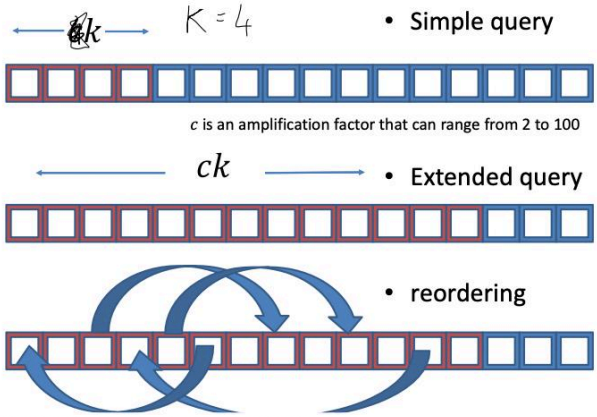
Se il risultato tende a 0 abbiamo performance molto basse.

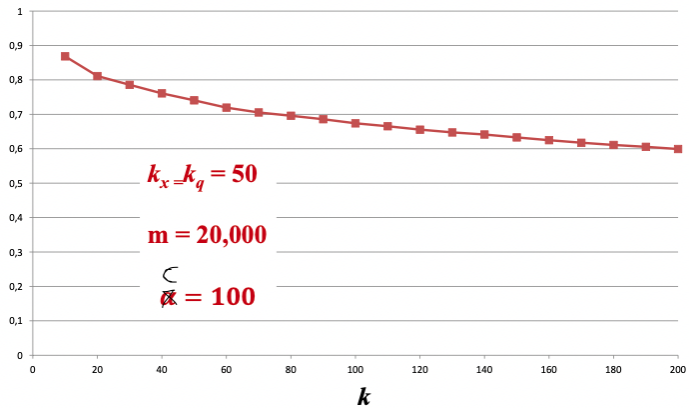
Dove R è il resut set della ricerca esatta tramite k-NN nel metric space originale e R^A è il result set approssimato.



1. Invece di cercare solo le top-k immagini, estendiamo la query a più risultati, quindi un k più alto
2. Usiamo quindi la similarity function per riordinare il result set

L'estensione della query non impatta troppo sull'efficienza se abbiamo un rapido e casuale accesso al DB delle features associate alle immagini.





Drawback of Surrogate Text Representation

- É approssimativo
- La scelta dei pivots può inficiare le performance
- Necessita un database dei metric object originali per fare il reordering su un disco SSD

Advantages of Surrogate Text Representation

- 0 codice da scrivere
- Indipendente dal search engine
- Affidabile su librerie di ricercar supportate:
    - Nuove versioni efficienti
    - Multithread, multi-query, parallel query
- Combina query testuali e image similarity

Summary

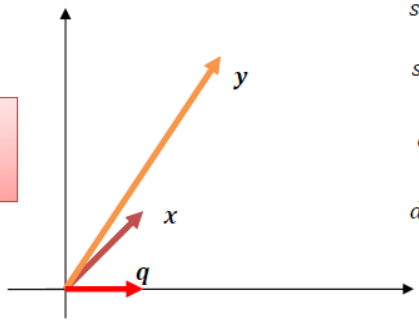
- CBIR problem (kNN) using Metric (gloabal) Features
  - Indexing time
    1. Feature extraction
    2. Select random pivots a.k.a. codebook
    3. Generate Surrogate Text Representation using codebook/pivots from features
    4. Index with a Text Retrieval Engine
    5. Store original features using a Random-access DB
  - Query Time
    1. Generate Surrogate Text Representation using codebook/pivots from features of query image
    2. Query search engine
    3. Reordering
    4. Get the top-k results

Quantization Techiques for Multimedia Retrieval

- Quantization
  - Vector Quantization
  - Product Quantization
  - Bag-of-Words
  - Inverted file and asymmetric distance computation (FAISS)
- Scalar Quantization with Threshold

Come abbiamo visto precedentemente abbiamo trasformato un problema di similarità in un problema di distanza fra vettori, un buon modo per cercare vettori simili è quello di adottare la **quantizzazione**.
Prima di tutto parliamo di *distanza* e *similarità*. Possiamo usare questi due modi per comparare due vettori. La distanza più usata è quella **Euclidea** chiamata anche L2, mentre la misura di similarità più usata è il **prodotto scalare**. Per prima cosa ci chiediamo se utilizzare la prima o la seconda possa risultare indifferente, la prima risposta è che la differenza fra le due è che l'ordine è **invertito**. Ad esempio, in una ricerca che usa il KNN e vogliamo avere il risultato più simile alla query dovremo scegliere quello più vicino. Quello che ci chiediamo adesso è se possiamo utilizzare indipendentemente queste due misure occupandoci solo dell'ordinamento. Come si può vedere dal seguente esempio non è sempre vero, data la query **q**, con la distanza otteniamo **x** mentre con la similarità otteniamo **y**.

y is the closest vector but at the same time the furthest away, not good!



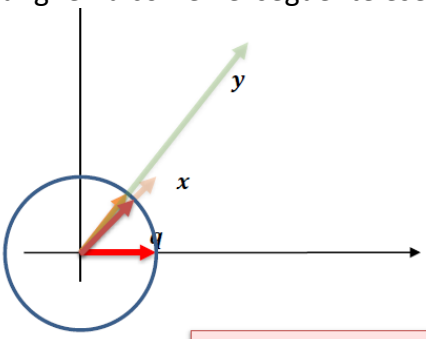
$$sim(\boldsymbol{q},\boldsymbol{x}) = \boldsymbol{q} * \boldsymbol{x} = 1 * 1 + 0 * 1 = 1$$

$$sim(\boldsymbol{q},\boldsymbol{y}) = \boldsymbol{q} * \boldsymbol{y} = 1 * 2 + 0 * 3 = 2$$

$$d(\boldsymbol{q},\boldsymbol{x})^2 = \|\boldsymbol{q} - \boldsymbol{x}\|^2 = (1-1)^2 + (0-1)^2 = 1$$

$$d(\boldsymbol{q},\boldsymbol{y})^2 = \|\boldsymbol{q} - \boldsymbol{y}\|^2 = (1-2)^2 + (0-3)^2 = 10$$

Per ottenere dei valori accettabili si deve normalizzare i vettori dividendo ogni vettore per la sua lunghezza come nel seguente esempio.



$$sim(\boldsymbol{q}',\boldsymbol{x}') = \boldsymbol{q}' * \boldsymbol{x}' = \frac{\sqrt{2}}{2} = 0.707$$

$$sim(\boldsymbol{q}',\boldsymbol{y}') = \boldsymbol{q}' * \boldsymbol{y}' = \frac{2}{\sqrt{13}} = 0.5547$$

$$d(\boldsymbol{q},\boldsymbol{x})^2 = \|\boldsymbol{q} - \boldsymbol{x}\|^2 = 0.5858$$

$$d(\boldsymbol{q},\boldsymbol{y})^2 = \|\boldsymbol{q} - \boldsymbol{y}\|^2 = 0.8906$$

When vector are normalized there is a **negative monotonicity** relationship between Euclidean distance and scalar product.
This similarity is often called "cosine similarity"

Quantization

Partiamo con una definizione, per quantizzazione si intende la divisione di una quantità in un numero discreto di piccole parti, spesso queste parti sono multipli interi di una quantità comune. In generale si vuole mappare un valore continuo in intervalli discreti.

Lo scopo della quantizzazione è ridurre la **cardinalità dello spazio di rappresentazione** in particolar modo quando i dati di input sono numeri reali. In altre parole, lo scopo è dividere una quantità in un numero discreto di piccole parti.
L'esempio più semplice di quantizzazione è il **rounding off**(approssimazione), consiste nell'approssimare ogni numero reale **x** all'intero più vicino **q**.
For example: $q(x) = \lfloor x + 0.5 \rfloor$ (where $\lfloor . \rfloor$ is the floor operator).

- $q(2.35) = \lfloor 2.35 + 0.5 \rfloor = \lfloor 2.85 \rfloor = 2$
- $q(3.84) = \lfloor 3.84 + 0.5 \rfloor = \lfloor 4.34 \rfloor = 4$
- $q(1) = \lfloor 1 + 0.5 \rfloor = \lfloor 1.5 \rfloor = 1$
- $q(-1.2) = \lfloor -1.2 + 0.5 \rfloor = \lfloor -0.7 \rfloor = -1$

Chiaramente questa quantizzazione porta ad un **errore** calcolato $e = q(x) - x$, nell'immagine sotto è mostrato un esempio.

$$e(2.35) = q(2.35) - 2.35 = 2 - 2.35 = -0.35$$

$$e(3.84) = q(3.84) - 3.84 = 4 - 3.84 = 0.16$$

$$\text{Therefore: } -0.5 \leq e \leq 0.5$$

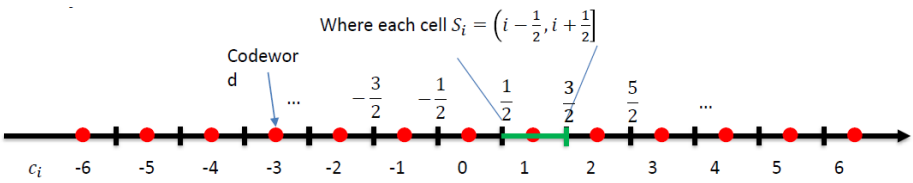
L'errore medio è nullo, mentre la deviazione standard ovviamente no, di seguito viene mostrata quella riguardante l'esempio sopra.

$$\mu(e) = 0$$

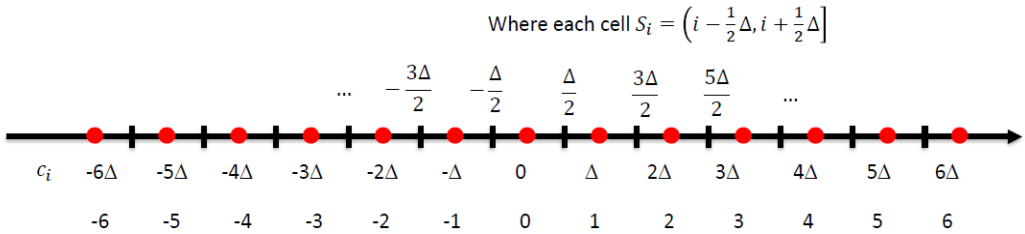
$$\sigma(e) = \frac{1}{\sqrt{12}} \approx 0.289$$

Scalar Quantization

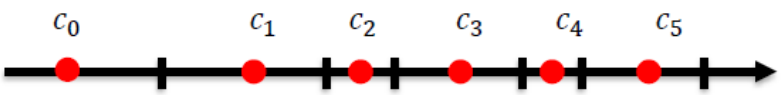
In maniera generale si può definire uno **scalar quantizer** come composto da $S = \{S_i | \ i \in I\}$ intervalli con $I = \{0 \dots k-1\}$, in I sono raccolti tutti gli indici. I valori che approssimano i nostri valori reali sono $C = \{c_i \ | i \in I, c_i \in \ \mathbb{R}\}$, questi valori sono chiamati **codewords**. Il quantizer sarà quindi definito come $q: \mathbb{R} \to C$.
$q(x) = \sum_i c_i 1_{S_i}(x)$ dove la funzione $1_{S_i}(x)$ vale 1 se $x \in S_i$ e 0 altrimenti. Facciamo un esempio sulla linea dei numeri reali dove ogni cella è disgiunta, ma le celle coprono l'interezza dello spazio come mostrato nella figura sotto.



La somma nella funzione del quantizer ha lo scopo di scorrere tutte le celle per trovare la cella corrispondente al valore che voglio quantizzare.
L'esempio che abbiamo appena fatto è con la dimensione della cella unitaria, si può applicare anche un Δ che serve a variare la larghezza della cella.

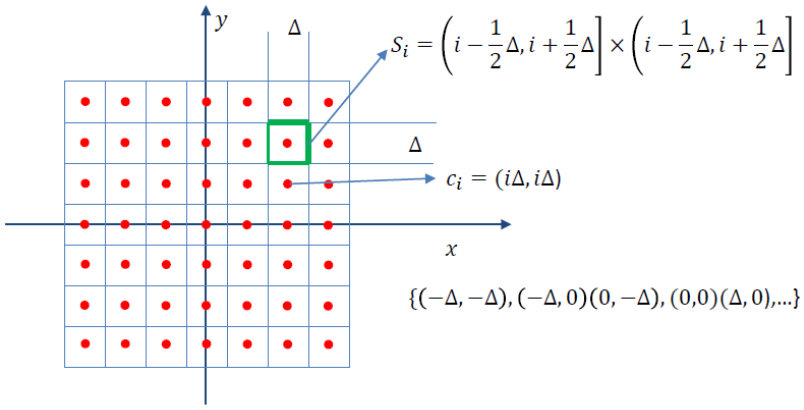


La quantizzazione può anche non essere uniforme come mostrato nella seguente immagine.

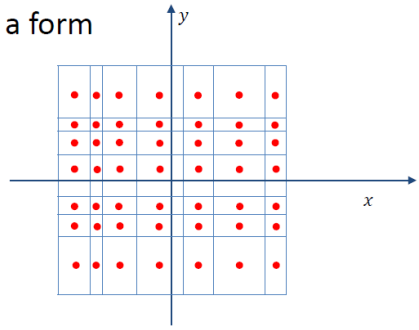


Vector Quantization

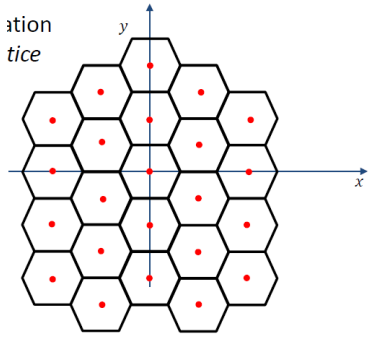
Adesso vediamo come applicare il concetto di quantizzazione ad un vettore.
Il primo approccio che possiamo utilizzare è quello di applicare quello che abbiamo appena visto per ogni coordinata del vettore. Dividiamo quindi ogni coordinata in intervalli indipendenti.
Nell'immagine successiva viene mostrato un esempio in due dimensioni per comodità, suddividendo entrambi gli assi si andranno a creare delle regioni che serviranno per quantizzare i valori continui.



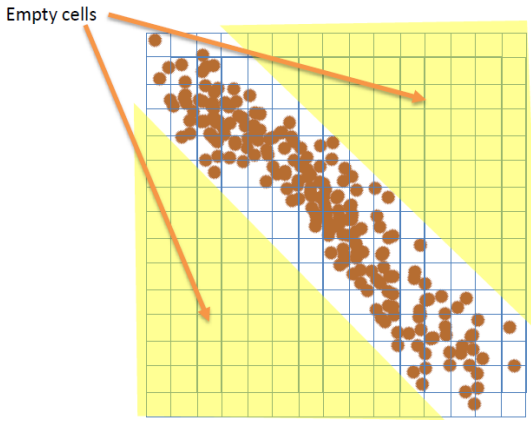
L'esempio appena fatto è il frutto di una quantizzazione uniforme, può essere anche applicata una quantizzazione non uniforme, il risultato sarà una griglia formata da celle irregolari.



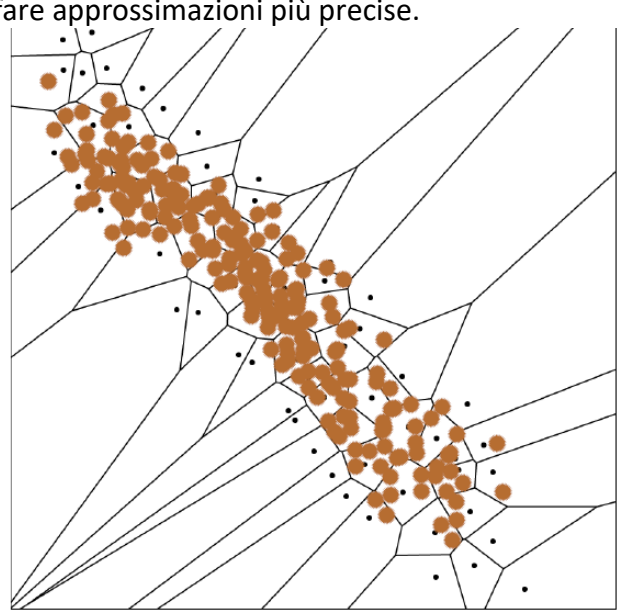
Si possono formare anche celle differenti rispetto a quelle quadrate. Si possono formare delle celle esagonali come quelle mostrate in figura, questo tipo di quantizzazione viene chiamata **Lattice quantization**.



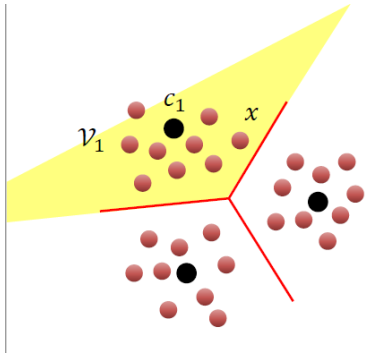
Tuttavia la **scalar quantization** non è utilizzata in quanto è molto difficile da implementare matematicamente e, in più, non tiene conto della distribuzione degli oggetti. Per questo ultimo motivo la maggior parte delle celle potrebbero rimanere vuote, mentre alcune potrebbero contenere un numero molto elevato di elementi come mostrato nella seguente immagine.



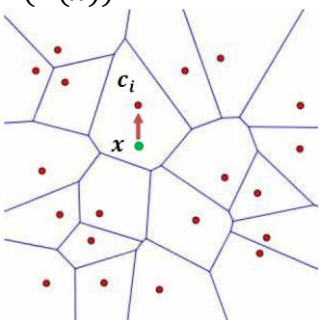
L'alternativa è quella di usare le **Voronoi** cell, con questo approccio si creano delle celle più grandi dove la concentrazione degli oggetti è bassa e celle più piccole dove la concentrazione è alta. In questo modo si possono fare approssimazioni più precise.



I punti neri sono i **raoresentation object**. Per creare questo tipo di celle si può utilizzare il k-means come metodo di clustering. Lo scopo finale è quello di rappresentare un grande numero di dati attraverso pochi **prototype vector**. Quindi si possono raggruppare i dati in cluster di informazioni simili. Di seguito si analizzerà come vengono formate le celle tramite il metodo di clustering.
Se il processo di quantizzazione comprende vettori $x \in \mathbb{R}^{\mathrm{d}}$ il quantizer sarà definito come $q: \mathbb{R}^d \to C$ e $q(x) \in C = \{c_i \,|i \in I, c_i \in \mathbb{R}^d\}$. I valori c_i che approssimano i vettori sono chiamati **centroids** o **codewords**. L'insieme C dei centroidi viene chiamato **codebook**.
Il set di $\mathcal{V}_i$ di vettori che vengono mappati ad un determinato index i è chiamato **Voronoi cell** ed è definito come$\mathcal{V}_i = \{x \in \mathbb{R}^d : q(x) = c_i\}$. In altre parole, tutti i vettori che hanno la stessa rappresentazione si trovano nella stessa cella.



Abbiamo definito la *cella* e i *centroidi*, dobbiamo definire la funzione che effettivamente serve per quantizzare i vettori. La funzione sarà $q(x) = \arg\min_{c_i \in C} d(x, c_i)$, quindi il vettore di input x viene approssimato al suo **codeword** più vicino. Ogni cella Voronoi non si sovrappone alle altre e insieme riempiono interamente lo spazio. Vogliamo ottimizzare il vector quantizer, per fare questo si vuole minimizzare l'errore che il quantizer commette. Generalizzando vogliamo minimizzare il **Mean Squared Error** definito come $MSE(q) = \sum_{i=1}^{N} \left|\left|x_i - q(x_i)\right|\right|^2$. Per minimizzare gli errori il centroide che approssima i vettori dovrà essere più o meno al centro del cluster, per un ripasso del funzionamento del k-mean guardare le slide da 163.
I vector quantizer possono essere suddivisi in due componenti: un **decoder** e un **encoder**. L'encoder E mappa ogni vettore ad un index I presente nel set di index, il decoder D si occupa di mappare l'indice prodotto dal decoder in un vettore presente nel **codebook** in modo da approssimare il vettore di input. L'operazione di Vector Quantizer sarà la seguente $q(x) = D(E(x))$.

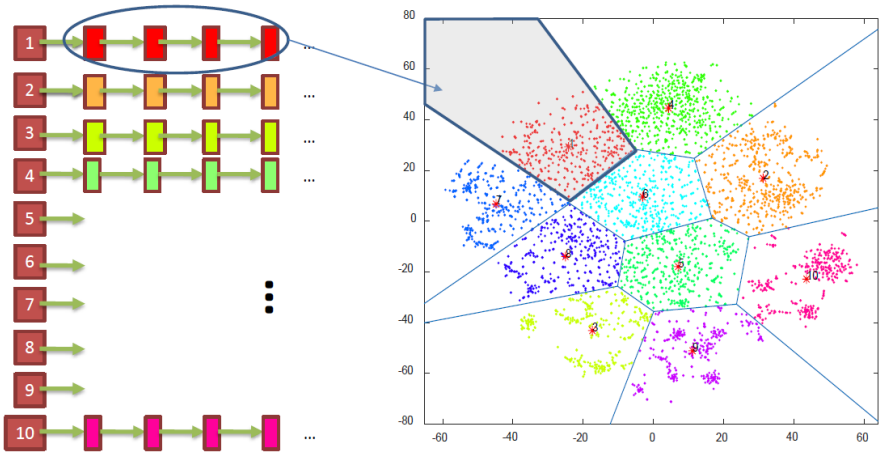


L'encoder mi da l'id della cella
Il decoder tramite l'id mi restituisce il vettore che fa da centroide

Nell'immagine seguente viene riportato un esempio pratico di un vector quantizer. Ci sono cinque regioni, ognuna delle quali è caratterizzata da un centroid indicato in blu. I punti in verde e rosso sono i vettori di input che voglio quantizzare, si capisce chiaramente che vengono approssimati al centroide in base alla regione a cui appartengono.

Vector Quantization for Approximate Nearest Neighbor (ANN)

Vediamo come approssimare la ricerca dei vicini tramite il metodo descritto precedentemente. Consideriamo di dover cercare delle immagini simili usando le feature estratte da una CNN, una volta estratte le feature pe ogni immagine presente nel database otteniamo un vettore di feature per immagine. Possiamo applicare il k-means per ottenere i cluster e di conseguenza le celle, una volta ottenuti i cluster possiamo usare il vector quantizer per costruire **l'inverted index**, che consiste nel creare una lista per ogni indice I contenente tutti i vettori appartenenti a quella determinata cella. Nella figura successiva è mostrato un esempio visivo dell'inverted index costruito.



Ogni elemento della lista è formato dall'id del vettore e dalle sue coordinate, nell'esempio saranno solo due.

Vediamo ora come si possono effettivamente ottenere le **k** immagini più vicine data una immagini di query. Data l'immagine di query se ne estraggono le feature ottenendo così un vettore, si applica l'encoder al vettore di query in modo da ottenere l'indice **i** che indica la cella a cui appartiene la query. In questo modo si può accedere alla lista dell'inverted index per calcolare la distanza fra la query e ogni membro della lista. Si mettono le distanze in un vettore che poi andrà riordinato e nelle prime **k** posizioni otteniamo le immagini più simili alla query.
Questo metodo porta ad un principale vantaggio, non si deve più calcolare la distanza fra la query e tutto il dataset per trovare il risultato, ma basta calcolare la distanza fra la query e tutti gli elementi della cella alla quale appartiene la query. In questo modo si diminuiscono di molto le distanze che devono essere calcolare e poi riordinate.

- Efficiency:
  - N = size of the DB (# of images)
  - k = # of clusters
- Brute force:
  - $N \times$ distance evaluations
- VQ:
  - $\frac{N}{k} \times$ distance evaluations

Ci sono tuttavia degli svantaggi, se la query si trova in una zona in cui confinano più celle non basta più accedere alle liste di un solo centroide, dobbiamo accedere alle liste dei centroidi più vicini e analizzare la distanza con tutti i componenti, questo è dovuto al fatto che se la query fosse troppo vicina al bordo di una cella, gli elementi più vicini potrebbero appartenere alla cella confinante. Il secondo svantaggio sta nel fatto che le informazioni relative a tutti i vettori devono essere tenute in memoria.
Di seguito sono mostrati gli algoritmi per fare index e per ricercare usando il metodo appena spiegato.

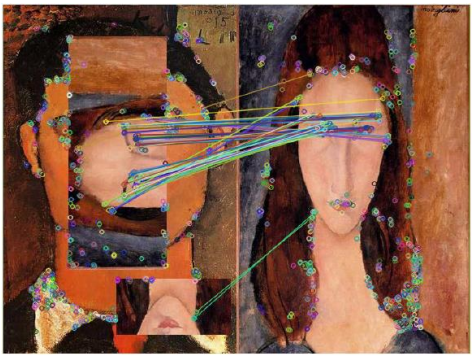
```
INDEX_VQ(Y)
for each y ∈ Y
do
        L_{E(y)} ← y
return L_0, ..., L_{k-1}
```

```
kNN_VQ(x, k)
i ← E(x)
return kNN(x, k, L_i)
```

Bag of Words Model

Uno degli ostacoli dei sistemi di imare retrieval basati sul contenuto è quello della **complessità** nella procedura di matching fra due immagini. Questo è dovuto al fatto che due immagini possono contenere centinaia di local feature e per poterle comparare è necessario calcolare la distanza fra tutte le coppie possibili come mostrato nella figura successiva.

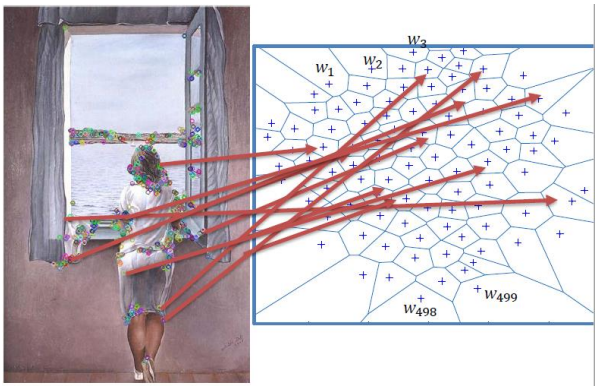
È possibile che addirittura due feature possano risultare in un match per casualità, si applica quindi uno step di filtraggio basato su trasformazioni geometriche, tuttavia questo approccio non è sufficiente.



La fase di matching risulta comunque molto pesante dal momento che calcolare la distanza fra tutte le possibili coppie di feature risulta molto complicato. Questa fase può essere alleggerita tramite le tecniche di quantizzazione, in particolare si fa uso del **Bag-Of-Words**. Con questo approccio viene scelto un set di **immagini candidate** in breve tempo, si applica poi solo a questo set il filtro.

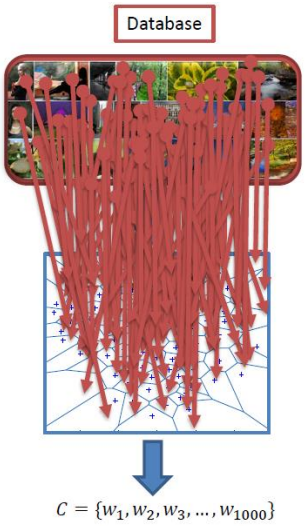
Questo approccio può essere usato per la classificazione o il retrieval di immagini, le feature di ogni immagine vengono viste come parole. L'immagine può essere quindi rappresentata come un vettore sparso contente il numero di occorrenze per ogni parola. Questo approccio è inoltre ottimo per costruire inverted index.

Vediamo come implementare questo approccio, per prima cosa si usa la vector quantization per quantizzare le local feature di un buon numero di immagini, solitamente qualche migliaio. Il questo modo si creai il **codebook** delle feature.

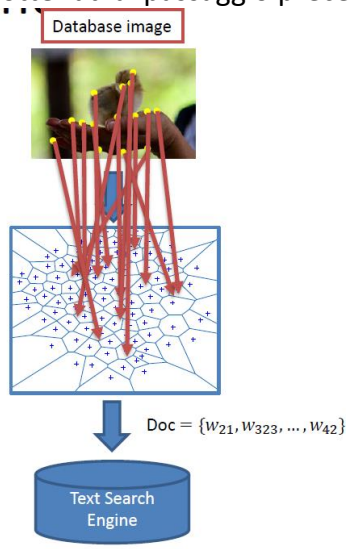


Dopo questa prima fase si comincia la fase di **indexing**, nella quale si possono quantizzare le local feature di ogni immagine che deve essere indicizzata creando una rappresentazione dell'immagine. Questa rappresentazione può essere vista come un documento, ad esempio **Surrogate Text Representazion**, e può essere usata dai motori di ricerca testuali come abbiamo visto precedentemente. Successivamente viene mostrato il processo appena descritto con delle immagini esplicative.

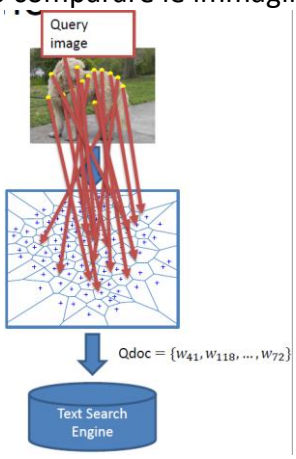
Per costruire il codebook si estraggono le feature dalle immagini del database e si trovano i cluster attraverso l'algoritmo di k-means.



Nella fase di indexing si analizza l'immagine singolarmente, se ne estraggono le feature e si crea a rappresentazione grazie ai centroidi ottenuti al passaggio precedente.



Nell'ultima fase si analizza l'immagine di query, se ne estraggono le feature e poi si crea la rappresentazione sempre usando i centroidi trovati nella prima fase. Grazie alle rappresentazioni i search engine basati sul testo possono comparare le immagini.



Questo approccio, tuttavia, presenta degli svantaggi. Prima di tutto una limitazione riguarda il fatto che lo step di quantizzazione deve essere applicato anche alla query. In secondo luogo, la grandezza del codebook può influire molto sull'accuratezza generale del sistema. Un codebook troppo piccolo può portare a molti false match, mentre uno troppo grande può portare alla perdita di molti true match da momento che il codebook generale diventa troppo specifico. Abbiamo quindi un **trade-off** tra efficienza e accuratezza della soluzione.

Product Quantization

La vector quantization da sola non è molto usata, vene però usata come base per costruire strutture più complesse come ad esempio:

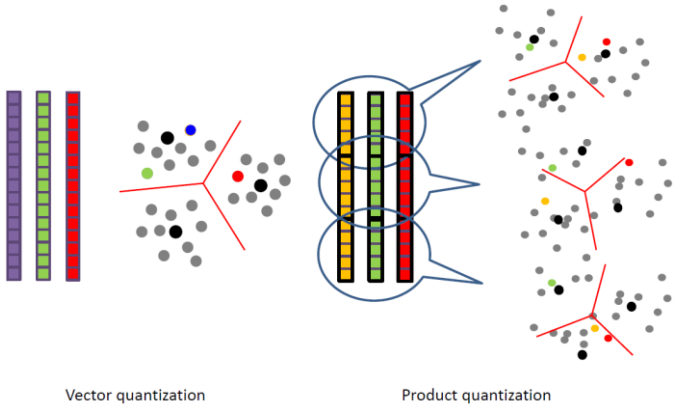
- Tree-structured VQ
- Multiple stage VQ
- Cartesian product VQ

Cartesian Product VQ

Il cartesian product VQ è un metodo che utilizza diversi sub-codebook e il codebook generale corrisponde al prodotto cartesiano dei sub-codebook. **Product quantization**, **cartesian k-means** e **transform coding** appartengono a questa categoria.

Product Quantization

La seguente immagine spiega l'idea dietro questo approccio. A sinistra si usa la vector quantization classica, si approssima quindi i vettori utilizzando solo un gruppo di cluster ottenuto tramite il k-means. Utilizzando la product quantization si suddividono i vettori in sottovettori e si approssimano usando differenti cluster.



Un input vector di dimensione **d** viene scomposto in **m** sottovettori ognuno di dimensione **t**. La suddivisione è fatta in modo tale che $t = \frac{d}{m}$.

$$\boldsymbol{x} = (\underbrace{x_1, x_2, \dots, x_t}_{\tilde{x}_1}, \underbrace{x_{t+1}, x_{t+2}, \dots, x_{2t}}_{\tilde{x}_2}, \dots, \underbrace{x_{d-t+1}, x_{d-t+2}, \dots, x_d}_{\tilde{x}_m})$$

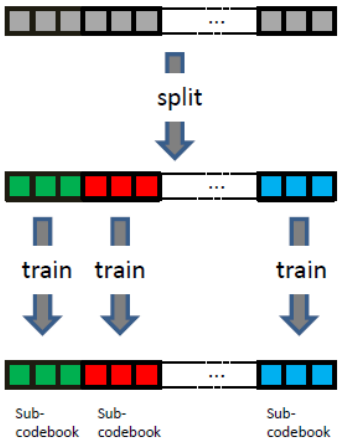
- where $\tilde{\boldsymbol{x}}_j$ is the j-th sub-vector of $\boldsymbol{x} \in \mathbb{R}^d$

Definiamo $q_j()$ come il j-esimo sub-quantizer per il j-esimo sub-vector. Di conseguenza il quantizer generale $q_p(x)$ sarà il seguente.

$$q_p(\boldsymbol{x}) = (q_1(\tilde{\boldsymbol{x}}_1), q_2(\tilde{\boldsymbol{x}}_2), \dots, q_m(\tilde{\boldsymbol{x}}_m)).$$

Chiaramente il risultato della quantizzazione è dovuto alla concatenazione dei risultati dei sub-quantizer.

Successivamente abbiamo che $C_p = C_1 \times C_2 \times \dots \times C_m$ è il codebook generale, mentre C_j è il j-esimo codebook corrispondente a $q_j(x)$, questo codebook può essere ottenuto applicando il k-means al j-esimo sub-vector. Nell'immagine successiva vediamo una rappresentazione grafica e in pratica come viene calcolato l'insieme dei centroidi di riferimento.



$$C_p = C_1 \times C_2 \times \dots \times C_m =$$
$$\{c_1^1, c_2^1, \dots, c_k^1\} \quad \{c_1^2, c_2^2, \dots, c_k^2\} \quad \{c_1^m, c_2^m, \dots, c_k^m\}$$

$$\{(c_1^1, c_1^2, \dots, c_1^m), \dots, (c_2^1, c_1^2, \dots, c_1^m), \dots, (c_1^1, c_2^2, \dots, c_1^m), \dots (c_k^1, c_k^2, \dots, c_k^m)\}$$

Parlando di indici, in questo approccio lo spazio degli indici è definito come $I_p = I_1 \times I_2 \times \dots \times I_m$ e l'encoder sarà una funzione del tipo $E_p: \mathbb{R}^d \to I^m$ed è definito come la concatenazione degli m sub-encoder.

$$E_p(\boldsymbol{x}) = (E_1(\tilde{\boldsymbol{x}}_1), E_2(\tilde{\boldsymbol{x}}_2), \dots, E_m(\tilde{\boldsymbol{x}}_m)).$$

$$I_p = \{(0,0, \dots, 0), (1,0, \dots, 0), (2,0, \dots, 0) \dots, (0,1, \dots 0), \dots (k-1, k-1, \dots, k-1)\}$$

Nelle due immagini successive è mostrato un esempio pratico di questo approccio

- 4-dimensional vectors, $\mathrm{d} = 4,\ m = 2, t = 2$

$$\boldsymbol{x} = (\underbrace{x_1, x_2}_{\tilde{\boldsymbol{x}}_1}, \underbrace{x_3, x_4}_{\tilde{\boldsymbol{x}}_2})$$

$C_1 = C_2 = \{\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4\} = \{(0,0), (0.5,0.5), (0.5, -0.5), (-0.5, -0.5), (0.5, -0.5)\}$

$C_p = C_1 \times C_2 = \{(0,0,0,0), (0,0,0.5,0.5), (0,0,0.5, -0.5), \dots, (0.5,0.5,0,0), \dots (0.5, -0.5,0.5, -0.5)\}$

$I = \{0,1,2,3,4\}$

$I_p = I \times I = \{(0,0), (0,1), \dots, (1,0), (1,1), \dots, (4,4)\}$

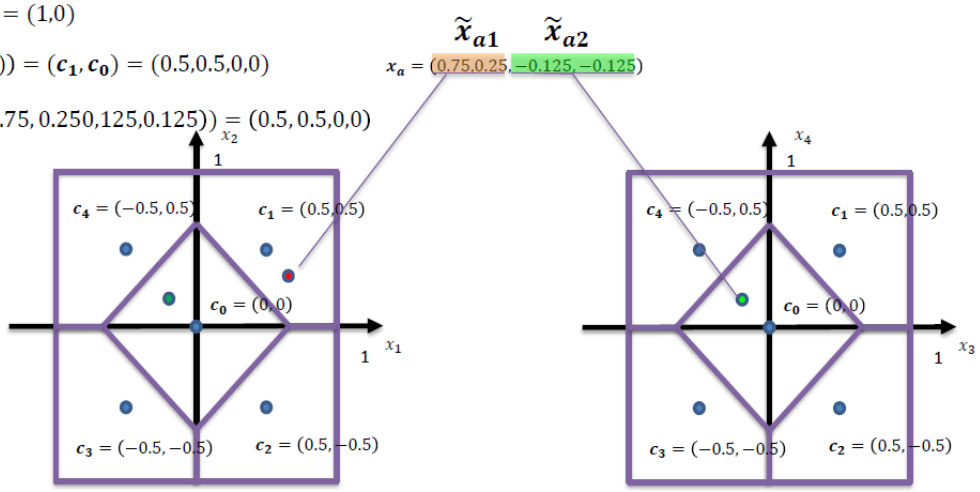
$q_p(\boldsymbol{x}) = (q_1(\tilde{\boldsymbol{x}}_1), q_2(\tilde{\boldsymbol{x}}_2))$.

$\boldsymbol{x_a} = (0.75,0.25, -0.125, -0.125) \rightarrow q_p(\boldsymbol{x_a}) = (q_1((0.75,0.25)), q_2(-0.125, -0.125)) = (\boldsymbol{c_1}, \boldsymbol{c_2}) = (0.5,0.5,0,0)$

$E(\boldsymbol{x_a}) = (1,0)$

$D((1,0)) = (\boldsymbol{c_1}, \boldsymbol{c_0}) = (0.5,0.5,0,0)$

$D(E(0.75, 0.250,125,0.125)) = (0.5, 0.5,0,0)$

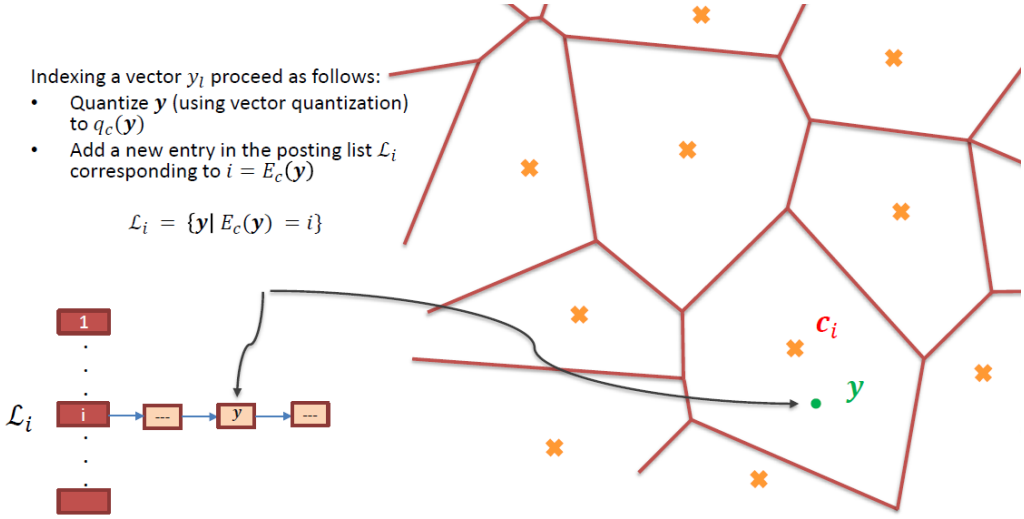


Inverted file and asymmetric distance computation with FAISS

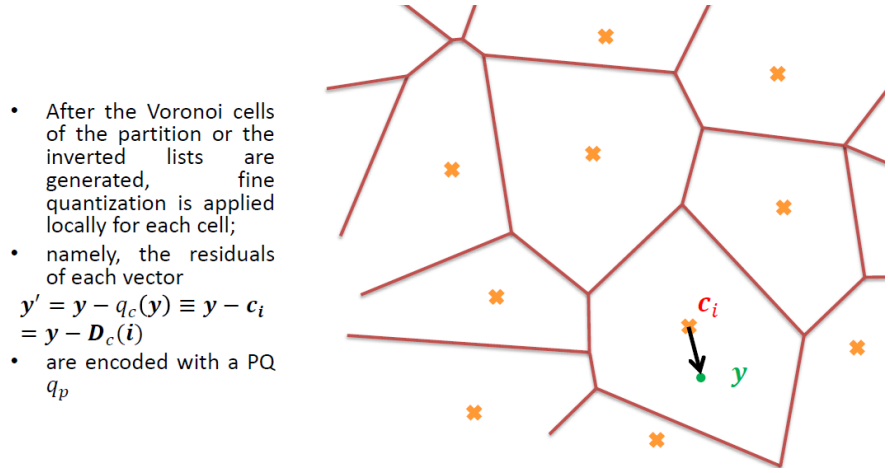
Faiss è una libreria per eseguire in maniera efficiente il similarity search e il clustering di vettori densi. Contiene algoritmi per la ricerca in set di vettori di qualunque dimensione, arrivando anche a quelli che non stanno tutti in RAM, contiene anche del codice che permette il tuning e la valutazione dei parametri. È scritta in C++ con però compatibilità completa con python. Ci concentreremo su un particolare metodo basato sul Product Quantization chiamato **IVFADC**.
In IVFADC abbiamo due tipi di quantizer:

- Il **coarse quantizer**, $q_c(x)$, è un semplice k-means quantizer applicato ai dati originali. È usato per costruire l'inverted list
- Il **fine quantizer,** $q_p(x)$, è un product quantizer usato per i residui.

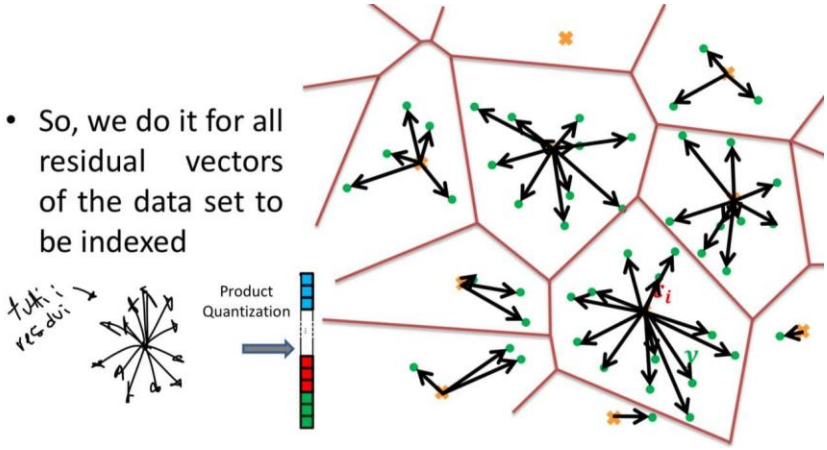
Vediamo in pratica il funzionamento di questo metodo. Per prima cosa, grazie al coarse quantizer, vengono formare le Voronoi Cell con i rispettivi centroidi. Successivamente, per indicizzare un vettore si va a inserire le caratteristiche del vettore all'interno della lista relativa al centroide della cella che lo contiene.



Dopo che le celle sono state create si può passare al secondo livello di quantizzazione. Prima però dobbiamo introdurre il concetto di **vettore residuo**. Un vettore residuo è il vettore differenza fra il vettore stesso e il centroide. Tutti questi vettori residuo vengono encodati tramite il Product quantizer.



Si calcolano i residui per tutti i vettori indicizzati e, per semplicità, si applica un solo quantizer a tutti i vettori residuo.



Quindi ogni vettore è calcolato come il centroide sommato al vettore residuo. Grazie però al secondo livello di approssimazione, il vettore residuo viene calcolato come il centroide del secondo livello più vicino al primo. (nella figura è disegnato in rosa).

- The final approximate vector is given by:

$$\boldsymbol{y} = q_c(\boldsymbol{y}) + \cancel{(\boldsymbol{y}')}$$

$$\Downarrow$$

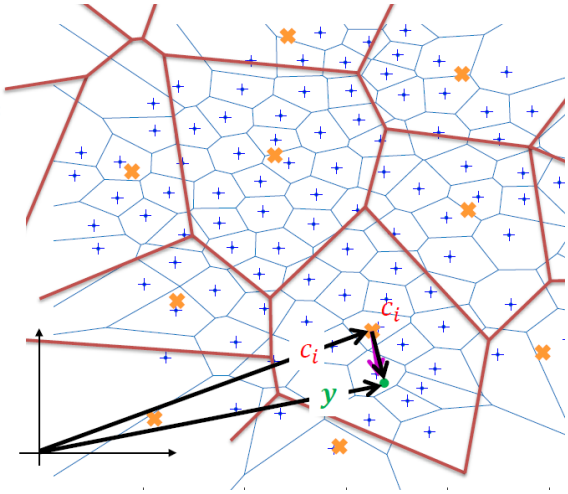
$$q_p\,(\boldsymbol{y}')$$

where

$$q_c: \mathbb{R}^d \to C \subset \mathbb{R}^d$$

and

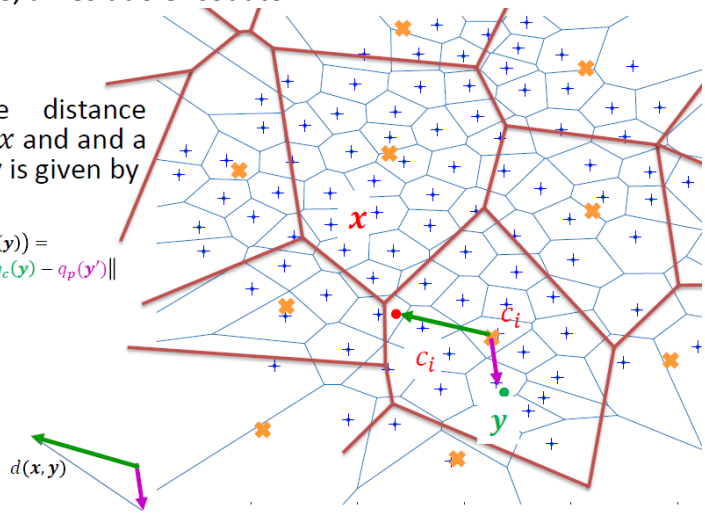
$$q_p: \mathbb{R}^d \to C_p \subset \mathbb{R}^d$$



Data una query **x**, la sua distanza con un vettore **y** del database è data dalla distanza di x con il centroide della cella di primo livello sommato alla distanza fra il centroide del primo livello e quello del secondo, in altre parole, al residuo encodato.

- the approximate distance between a query x and and a database vector $\boldsymbol{y}$ is given by

$$d(\boldsymbol{x},\boldsymbol{y}) \simeq d(\boldsymbol{x} - q_c(\boldsymbol{y}), q_p(\boldsymbol{y} - q_c(\boldsymbol{y})) = d(\boldsymbol{x} - q_c(\boldsymbol{y}), q_p(\boldsymbol{y}')) = \|\boldsymbol{x} - q_c(\boldsymbol{y}) - q_p(\boldsymbol{y}')\|$$



Nelle prossime immagini è riassunto questo metodo con tutte le formule.

- To summarize:
  - After the Voronoi cells of the partition or the inverted lists are generated, fine quantization is applied locally for each cell; namely, the residuals of each vector $\mathbf{y}' = \boldsymbol{y} - q_c(\boldsymbol{y})$ are encoded with a PQ q_p
  - The final approximate vector is given by:

$$\boldsymbol{y} \approx q_c(\boldsymbol{y}') + q_p\,(\boldsymbol{y}')$$

  - $d(\boldsymbol{x},\boldsymbol{y}) \simeq \|\boldsymbol{x} - (q_c(\boldsymbol{y}') + q_p(\boldsymbol{y}'))\| = \|\boldsymbol{x} - q_c(\boldsymbol{y}') - q_p(\boldsymbol{y}')\|$
  - Therefore, the approximate squared distance between a query $\boldsymbol{x}$ and a database vector $\boldsymbol{y}$ is given by

$$d(\boldsymbol{x},\boldsymbol{y})^2 \simeq d\left(\boldsymbol{x} - q_c(\boldsymbol{y}'), q_p(\boldsymbol{y}')\right)^2 = \|\boldsymbol{x} - q_c(\boldsymbol{y}') - q_p(\boldsymbol{y}')\|^2$$

- We define the PQ encoder E_p

$$E_p(\boldsymbol{y}') = \left(E_1(\tilde{\boldsymbol{y}}'_1), E_2(\tilde{\boldsymbol{y}}'_2), \ldots, E_m(\tilde{\boldsymbol{y}}'_m)\right) = \left(E_f(\tilde{\boldsymbol{y}}'_1), E_f(\tilde{\boldsymbol{y}}'_2), \ldots, E_f(\tilde{\boldsymbol{y}}'_m)\right)$$

- Where $E_f = E_1 = E_2 = \cdots = E_m$ is a single fine global encoder (simplifying assumption)
- $q_p(\boldsymbol{y}') = \left(q_f(\tilde{\boldsymbol{y}}'_1), q_f(\tilde{\boldsymbol{y}}'_2), \ldots, q_f(\tilde{\boldsymbol{y}}'_m)\right)$
- Where we remind that we have decomposed the residual vectors as:

$$\boldsymbol{x}' = \boldsymbol{x} - q_c(\boldsymbol{y}) = (\tilde{\boldsymbol{x}}'_1, \ldots, \tilde{\boldsymbol{x}}'_m) \longrightarrow \text{with no fine quantization}$$

$$\boldsymbol{y}' = \boldsymbol{y} - q_c(\boldsymbol{y}) = (\tilde{\boldsymbol{y}}'_1, \ldots, \tilde{\boldsymbol{y}}'_m) \longrightarrow \text{with fine quantization}$$

- if we neglect the calculation of the square root, in fact for nearest neighbors search, we do not compute the square roots in practice

$$d(\boldsymbol{x},\boldsymbol{y})^2 \simeq d\left(\boldsymbol{x} - q_c(\boldsymbol{y}'), q_p(\boldsymbol{y}')\right)^2 = \left\|\boldsymbol{x} - q_c(\boldsymbol{y}') - q_p(\boldsymbol{y}')\right\|^2$$

- That is:

$$d(\boldsymbol{x},\boldsymbol{y})^2 \cong \left\|\tilde{\boldsymbol{x}}'_1 - q_f(\tilde{\boldsymbol{y}}'_1)\right\|^2 + \left\|\tilde{\boldsymbol{x}}'_2 - q_f(\tilde{\boldsymbol{y}}'_2)\right\|^2 + \cdots + \left\|\tilde{\boldsymbol{x}}'_m - q_f(\tilde{\boldsymbol{y}}'_m)\right\|^2$$

- The quantizer q_c needs to have a relatively low number of reproduction values so that the number of inverted lists does not explode.
- Typically, $|C| \simeq \sqrt{N}$, trained via k-means.
- For instance, $N = 100{,}000{,}000 \rightarrow |C| = 10{,}000$

Da slide 223 a 236 è mostrato un esempio pratico di questo approccio.

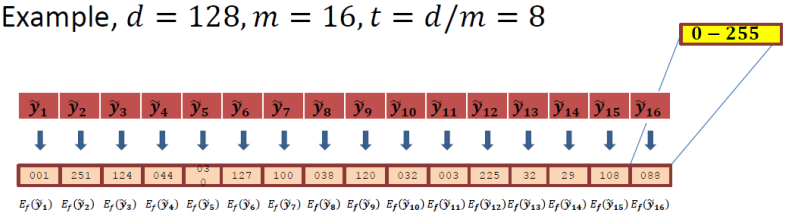
Questo approccio può essere ottimizzato. Prima di tutto è necessario notare che i residui quantizzati non sono salvati nella posting list dell'inverted index perché questo porterebbe ad un grande spreco di memoria.
La prima ottimizzazione riguarda lo spazio occupato dalla quantizzazione dei residui.

- The q_p sub-quantizers typically have 256 (k_f) reproduction values to fit in one byte.
- The quantization value of the product quantizer is then

$$E_p(\boldsymbol{y}) = E_f(\tilde{\boldsymbol{y}}_1) + 256 \times E_f(\tilde{\boldsymbol{y}}_2), \ldots, 256^m \times E_f(\tilde{\boldsymbol{y}}_m).$$

Which from a storage point of view is just the concatenation of the m bytes produced by each sub-quantizer.

- Thus, the product quantizer generates b-byte codes with $|C_P| = k_f^m = 256^m$ reproduction values.
- The k-means dictionaries of the quantizers are small and quantization is computationally cheap, ie., $k_f = 256$
- Example, $d = 128, m = 16, t = d/m = 8$



Il secondo metodo di ottimizzazione consiste nel creare delle **tabelle di lookup** che vengono usate quando si deve calcolare la distanza di una query da un vettore qualsiasi del database. Questo può essere fatto perché la fine quantization di un vettore qualunque del database è costante e non dipende dalla query.

$$d(\boldsymbol{x},\boldsymbol{y})^2 \simeq \|\boldsymbol{x}'\|^2 + \left\|q_p(\boldsymbol{y}')\right\|^2 - 2\boldsymbol{x}' * q_p(\boldsymbol{y}') =$$

- Substituting $\boldsymbol{x}' = \boldsymbol{x} - q_c(\boldsymbol{y})$, we get
- $\|\boldsymbol{x} - q_c(\boldsymbol{y})\|^2 + \left\|q_p(\boldsymbol{y}')\right\|^2 - 2\left(\boldsymbol{x} - q_c(\boldsymbol{y})\right) * q_p(\boldsymbol{y}') =$
- $\|\boldsymbol{x} - q_c(\boldsymbol{y})\|^2 + \left\|q_p(\boldsymbol{y}')\right\|^2 - 2\boldsymbol{x} * q_p(\boldsymbol{y}') + 2q_c(\boldsymbol{x}) * q_p(\boldsymbol{y}') =$

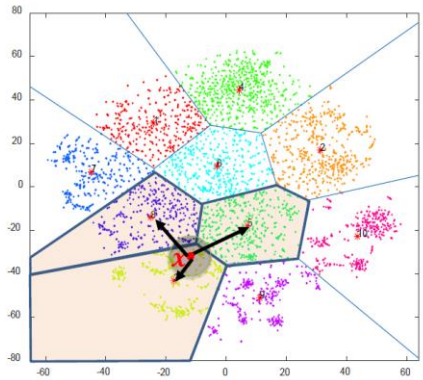
Dall'immagine precedente si può vedere come viene calcolata la distanza, il secondo e il terzo fattore possono essere calcolati in anticipo dato che non dipendono dalla query x. In questo modo si incrementa la velocità di calcolo della distanza.
Un altro problema capita quando una query si trova al limite della sua cella, in questo modo perderemmo dei vicini non appartenenti alla stessa cella della query, per risolvere questo modo spesso si può assegnare la query x a **w** indici che rappresentano le **w** celle più vicine alla query x per non perdere nessun vettore.

- multiple assignment strategy
- the query $\boldsymbol{x}$ is assigned to w indexes instead of only one, which correspond to the w nearest neighbors of $\boldsymbol{x}$ in the codebook of q_c.

$w = 1$

$w = 2$

$w = 3$



Audio Retrieval

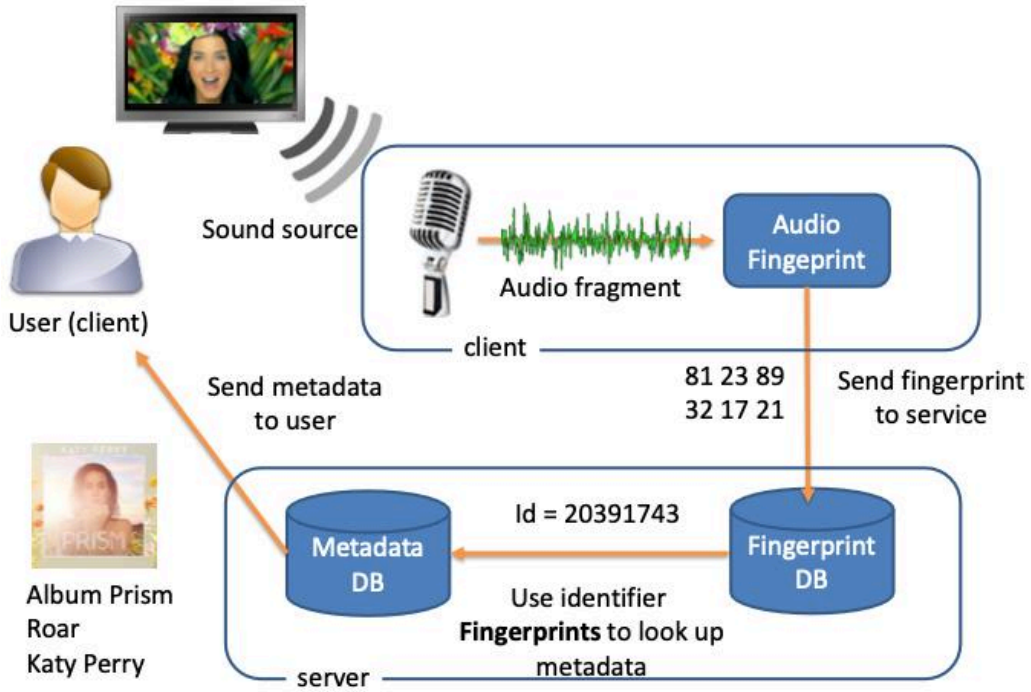
Content-Based Audio Retrieval

Parleremo di content-based retrieval seguendo il paradigma **query-by-example**: dato un piccolo frammento di audio come query vogliamo identificare la registrazione audio dalla quale la query proviene. Questo tipo di retrieval è conosciuto anche come **Audio identification**. Ci sono anche task più complessi come ad esempio l'Audio matching e Version identification.

Audio Identification e Fingerprint

Audio fingerprint: sono delle tecniche di Audio identification integrate in molte applicazioni commerciali broadcast, di monitoraggio del copyright e altre applicazione che servono per restituire metadata ed altri contenuti. Esempi: shazam, musichound, siri, alexa, ecc..

Audio Fingerprint Model



Audio Fingerprint Requirements

- High **specificity**: l'abilità di essere distinta tra milioni di altre
- **Robustness:** anche se sono presenti rumori di background oppure distorsione del segnale
- **Compactness:** di piccole dimensioni
- **Scalability:** l'impronta deve essere computazionalmente poco dispendiosa e efficiente da calcolare, dato può essere necessario applicarla a milioni di registrazioni

Dobbiamo tenere di conto di due **trade-off** tra requirments:

- *L'incremento della robustezza* (smussiamo di più) solitamente porta a fare delle *identificazioni sbagliate* (false positive)
- *Un'eccessiva riduzione della dimensione dell'impronta* inficia negativamente sulla *capacità discriminatoria.*

Fingerprint Based on Spectral Peaks

E' l'idea principale dell'audio fingerprint usata in shazam.

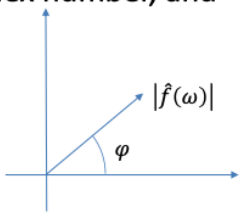
Gli **Spectral Peaks** hanno:

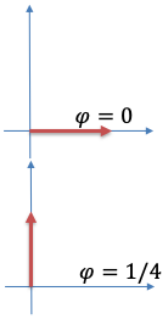
- *Temporal locality* (quando si calcolano gli spectral peak essi non sono influenzati da samples distanti).
- *Translation invariance* (inizio a registrare quando la traccia audio è già iniziata, non è un problema appunto per questa caratteristica).

Design of Audio Fingerprint

Il primo step è calcolare la discrete **Short-Time Fourier Transform**, questa trasformata ci fornisce informazioni sulla frequenza che sono averaged nell'intero dominio del tempo (necessito informazioni sulla frequenza che non vedo nel dominio del tempo).

$\hat{f}(\omega)$ is a complex number, and





$|\hat{f}(\omega)|$ is its **magnitude A** (*ampiezza*)

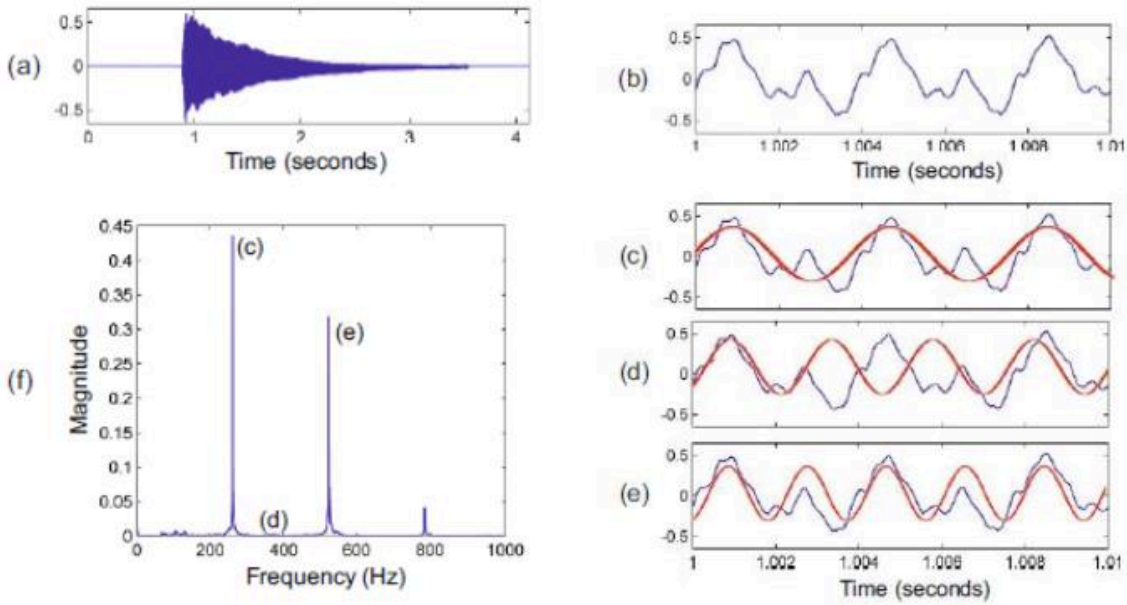
A point of the $\hat{f}(\omega)$ can be seen as:

$$A \sin(2\pi(\omega t - \varphi))$$

φ is the (normalized) phase between 0 and 1

Example

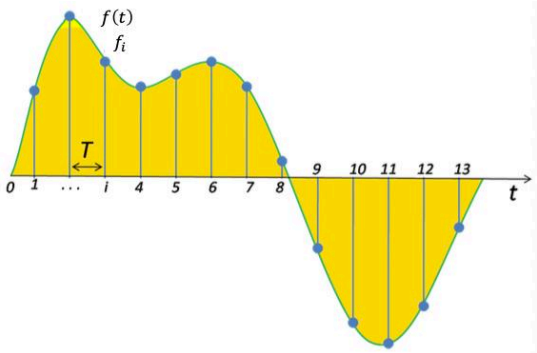
Waveform of a note C4 (261.6 Hz) played on a piano



Preso in esempio (a) che corrisponde al suono emesso da una nota suonata da un piano, campioniamo la prima parte ed otteniamo (b), se ci applichiamo la FT otteniamo (f) che indica il grafico in frequenza.

Discrete Fourier Transform (DFT)

Quando si lavora con tecnologie digitali dobbiamo andare a *discretizzare*, cioè andiamo a prendere dei sample nel dominio del tempo.



The signal f becomes a function x of $n \in \mathbb{Z}$.

$$x(n) := f\,(n\,T)$$

Where T is the so-called ***sampling period*** the inverse of the ***sampling rate*** (or **sampling frequency**):

$$F_s := 1/T$$

La parte che ci interessa di x è nell'intervallo [0 : N-1]

$$\hat{x}(\omega) := \sum_{n=0}^{N-1} x(n)\ e^{-2\pi i\omega n}$$

In practice, one often computes the Fourier transform only for a finite subset of frequencies (**sampling** in the **frequency domain**)

$$\omega\ =\ k/N \text{ for } k\ \in\ [0 : N-1]$$

Considero N = k

$$\hat{x}\left(\frac{k}{N}\right) := \sum_{n=0}^{N-1} x(n)\ e^{-2\pi i\frac{k}{N}n}$$

the Riemann approximation

$$\mathrm{X}(k) := \hat{x}\left(\frac{k}{N}\right) = \frac{1}{T}\hat{f}\left(\frac{k}{N}\cdot\frac{1}{T}\right)$$

Where

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i\omega t}dt$$

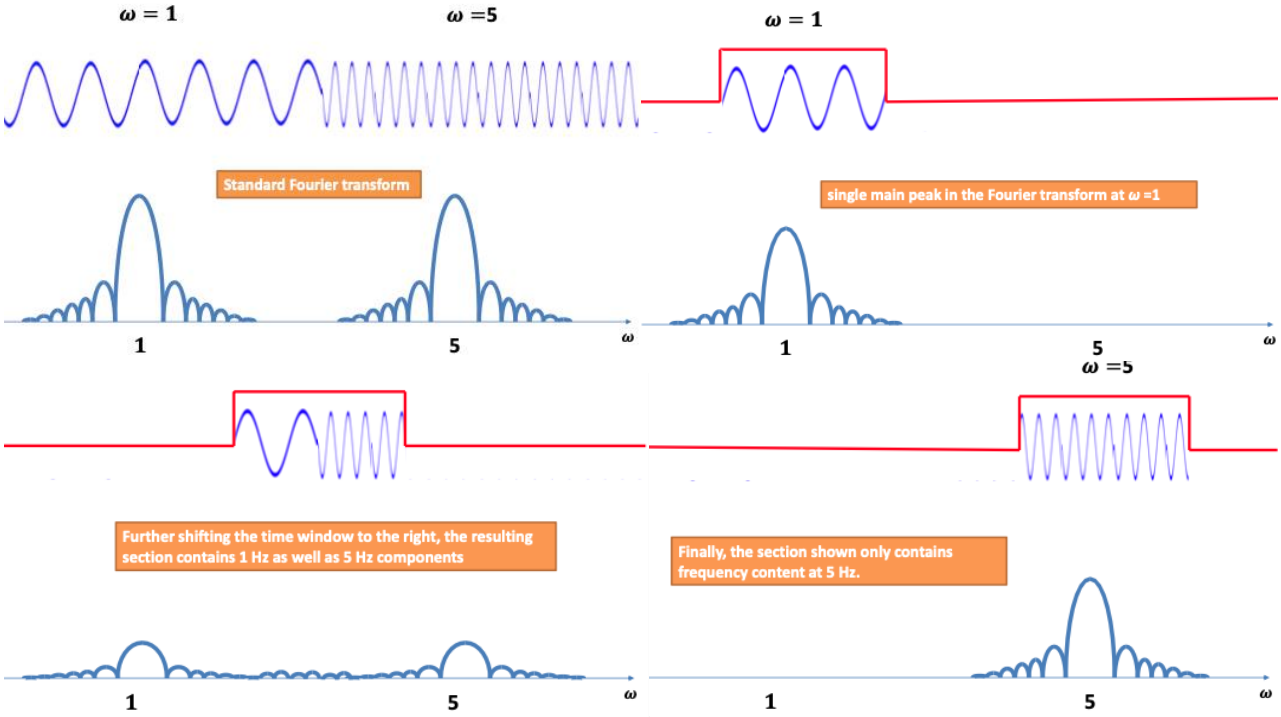
This approximation is only good if f does not contain frequency components above the Nyquist frequency $\frac{1}{2T}$.

Riguarda slide da 17 a 23.

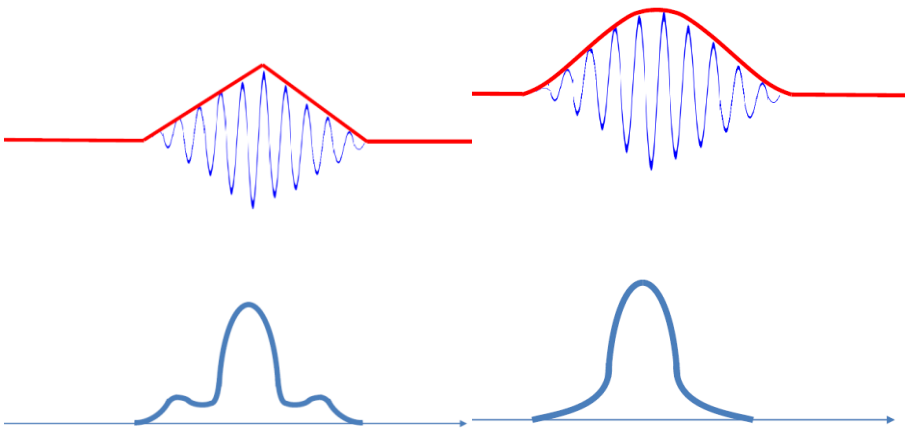
Short Time Fourier Tranform

La FT descrive la frequenza sull'intero segnale, è come se fosse una Global Feature. Passando alla frequenza andiamo a perdere informazioni sul tempo come ad esempio quando un evento particolare è accaduto. Un fenomeno locale del segnale diventa un fenomeno globale nella Fourier trasformation.

Nella **STFT** invece di considerare il segnale interno viene considerato solo una piccola parte del segnale. In poche parole prendiamo una window function, cioè 0 ovunque tranne che in un pezzetto, che scorre sul segnale ed applichiamo la FT all'interno di questa finestra.

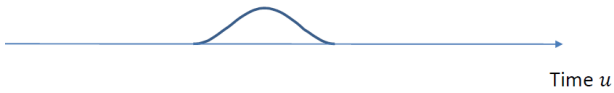


Come vediamo nell'esempio STFT non riflette soltanto le caratteristiche del segnale ma anche quelle della finestra utilizzata. In più la STFT dipende anche dalla lunghezza effettiva della finestra. Vediamo che utilizzando la finestra rettangolare si genera un ripple prima e dopo il segnale. Introduciamo quindi diverse forme di finestre per fare in modo che il segnale generato sia più pulito.



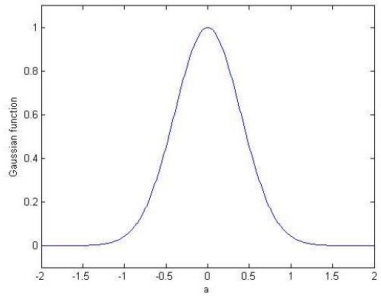
La funzione che definisce l'ultimo tipo di finestra è la **funzione Hann**. Questa funzione corrisponde ad un coseno rialzato definito come segue:

$$g(u) := \begin{cases} \dfrac{1+\cos \pi u}{2} & \text{if } -0.5 \le u \le 0.5 \\ 0 & \text{otherwise} \end{cases}$$



Se la funzione che definisce la finestra è Gaussiana allora abbiamo la **Trasformata Gabor** definita come segue:

$$g(u) := e^{-\pi u^2}$$



Dopo aver introdotto questi tipi di finestre possiamo definire effettivamente come viene calcolata STFT.

- Standard Fourier

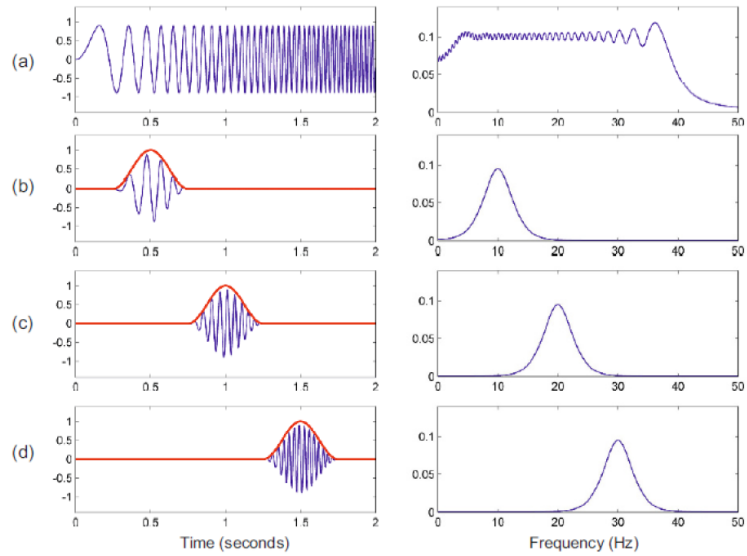
$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \omega t}$$

- STFT

This function is shifted across time t

$$\tilde{f}(t,\omega) = \int_{-\infty}^{\infty} f(u) g(u-t) e^{-2\pi i \omega u} du$$

La funzione g() è la funzione che definisce la finestra e viene traslata attraverso il parametro **t** come possiamo vedere nell'esempio seguente.



Ci sono diverse definizioni di STFT

- Original

$$\widetilde{f_g}(t,\omega) = \int_{-\infty}^{\infty} f(u)g(u-t)e^{-2\pi i\omega u}du$$

- Modulated window:

$$\widetilde{f^g}(t,\omega) = \int_{-\infty}^{\infty} f(u)\boxed{g(u-t)e^{-2\pi i\omega(u-t)}}du$$

Intuitively, this function may be thought of as a "musical note" of frequency ω that is active in a neighborhood of t

and:

- $\widetilde{f_g}(t,\omega) = \widetilde{f^g}(t,\omega)e^{2\pi i\omega t}$

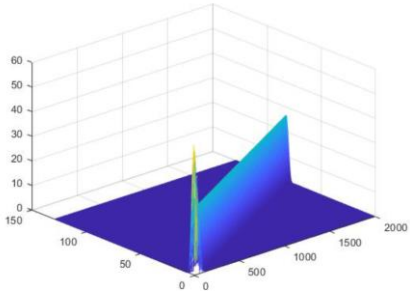
La STFT genera un numero complesso per ogni tempo **t** e per ogni frequenza omega. Questa informazione è spesso rappresentata tramite lo **spectrogram** che è una rappresentazione su due dimensioni dell'ampiezza al quadrato.

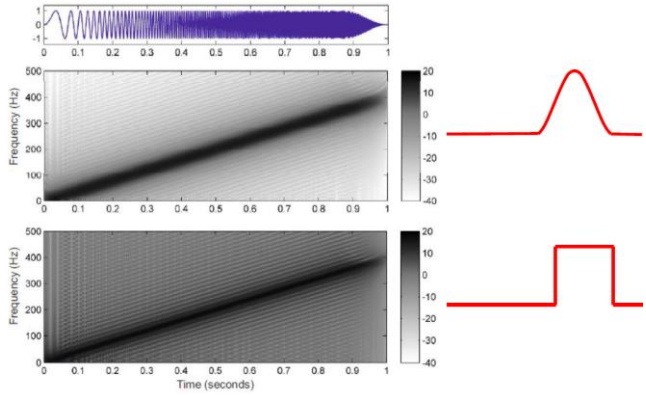
- It shows the intensity of the occurring frequencies over time (squared magnitude).

$$Spec(t,\omega) = \left|\tilde{f}(t,\omega)\right|^2$$

- since the "modulation factor" has a magnitude of one, we can use the alternative version of STFT:

$$Spec(t,\omega) = \left|\widetilde{f_g}(t,\omega)\right|^2 = \left|\widetilde{f^g}(t,\omega)\right|^2$$





where the intensity is reflected by the shade of gray

Quello che abbiamo considerato fino ad ora riguarda i segnali analogici e quindi la STFT è continua, in pratica quello che si fa è utilizzare il segnale campionato per calcolare la STFT solo in determinati punti.

$$\tilde{x}(m,\omega) := \sum_{n\in\mathbb{Z}} x(n)w(n-m)e^{-2\pi i\omega(n-m)}$$

$$= \sum_{n\in\mathbb{Z}} x(n+m)w(n)e^{-2\pi i\omega n}$$

Quello che si fa non è più spostare la finestra ma questa viene centrata a metà della larghezza nel centro e si sposta il segnale. Dato che abbiamo campionato nel dominio del tempo la trasformata risulterà comunque continua, bisogna campionare anche nel dominio della frequenza per ottenere una STFT discreta.

$$\tilde{x}\left(m,\frac{k}{N}\right) := \sum_{n\in\mathbb{Z}} x(n+m)w(n)e^{-2\pi i\frac{k}{N}n}$$

$$\mathrm{X}(m,k) := \tilde{x}\left(m,\frac{k}{N}\right)$$

$$\mathrm{X}(m,k) := \sum_{n=0}^{N-1} x(n+m)w(n)e^{-2\pi i\frac{k}{N}n}$$

which considers only the nonzero samples of the discrete window w for $n \in [0 : N-1]$.

DA CAPIRE slide 49-67 audio ret

Design of Audio Fingerprints

Nella seconda parte della computazione dell'impronta andiamo a ridurre la rappresentazione STFT del segnale ad un **set sparso di punti tempo-frequenza**.
Usiamo una strategia peak-picking che identifica i punti tempo-frequenza che hanno una **ampiezza più alta rispetto a tutti i vicini** dentro una regione intorno ai rispettivi punti.

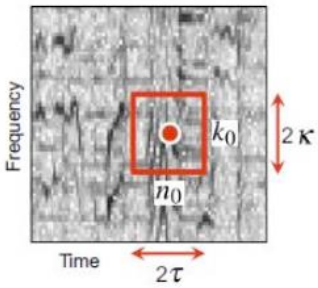
Dati due parametri $\tau > 0$ e $\kappa > 0$ che definiscono la grandezza del vicinato rispettivamente per tempo e frequenza, andiamo a scegliere un punto (n_0,k_0) come picco se

$$\mathrm{X}(n_0,k_0) > \mathrm{X}(n,k)$$

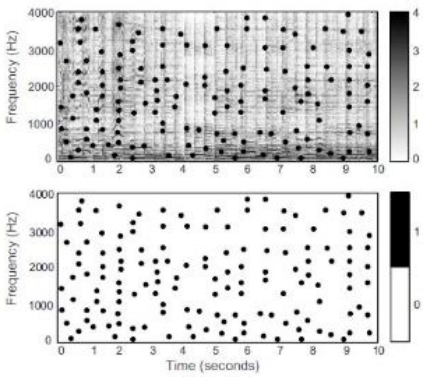
In the window

$$(n,k) \in ([n_0-\tau, n_0+\tau] \times [k_0-\kappa, k_0+\kappa])$$

In pratica se il punto selezionato è maggiore stretto di tutti gli altri all'interno della finestra.
Ovviamente più aumentiamo la grandezza della finestra e più difficile sarà che sia scelto come picco.



L'ampiezza dei picchi non viene più usata, salvo solamente le coordinate di tempo e frequenza dei picchi che considero, questo mi porta ad avere un alto grado di robustezza.



La constellation map consiste quindi in un set finito di coordinate (n, k) che consistono in un time stamp e in una frequency stamp.

For instance

$$\mathcal{F}\ (\mathcal{D}) = \{(0,2), (0,4), (1,3), (1,4), \dots\}$$

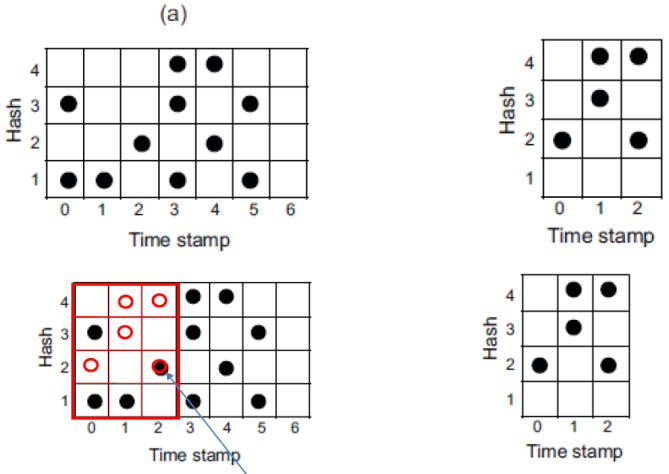
Adesso possiamo vedere come fare match fra un audio usato come query e un database di audio. Per prima cosa il database è formato da costellazioni di punti come appena descritto. Dobbiamo quindi calcolare la costellazione della query nello stesso modo, otteniamo quindi $\mathcal{F}(\mathcal{Q})$.
Spostiamo la query di **m** posizioni ottenendo così ma costellazione $m + \mathcal{F}(\mathcal{Q})$ definita come segue

$$\boxed{m} + \mathcal{F}(\mathcal{Q}) := \{(\boxed{m} + n, k) | (n, k) \in \mathcal{F}(\mathcal{Q})\}$$

Per valutare il match dobbiamo semplicemente intersecare la costellazione relativa alla query e quella relativa a un campione del database, più formalmente

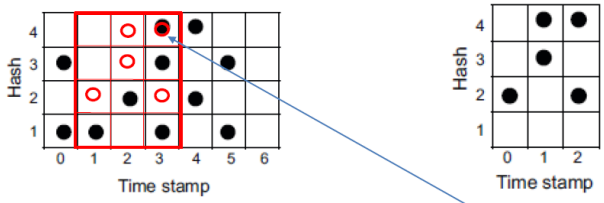
$$\Delta_{\mathcal{F}}(m) := |(m + \mathcal{F}(\mathcal{Q})) \cap \mathcal{F}(\mathcal{D})|$$

Nelle prossime slide è presentato un esempio pratico di un match fra una query formata da tre campioni nel tempo e un elemento del database. Si sposta progressivamente la query e se ne calcola l'intersezione.

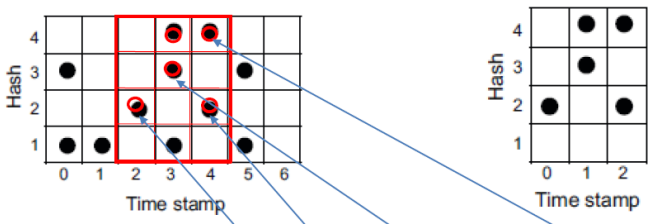


$\mathcal{F}(\mathcal{D}) =$
$\{(0,1), (1,1), (3,1), (5,1), (2,2), (4,2)(0,3), (3,3), (5,3), (3,4), (4,4)\}$
$\mathcal{F}(\mathcal{Q}) = \{(0,2), (2,2), (1,3)(1,4)(2,4)\}$
$\Delta_{\mathcal{F}}(0) = 1$

Si comincia adesso a spostare la costellazione della query



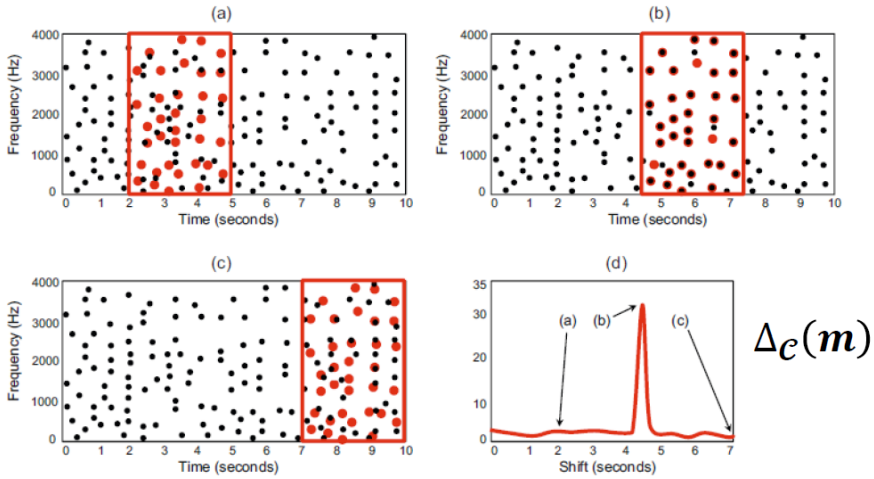
$\mathcal{F}(\mathcal{D})$
$= \{(0,1), (1,1), (3,1), (5,1), (2,2), (4,2)(0,3), (3,3), (5,3), (3,4), (4,4)\}$
$\mathcal{F}(\mathcal{Q}) + 1 = \{(0 + 1,2), (2 + 1,2), (1 + 1,3)(1 + 1,4)(2 + 1,4)\}$
$\Delta_{\mathcal{F}}(1) = 1$



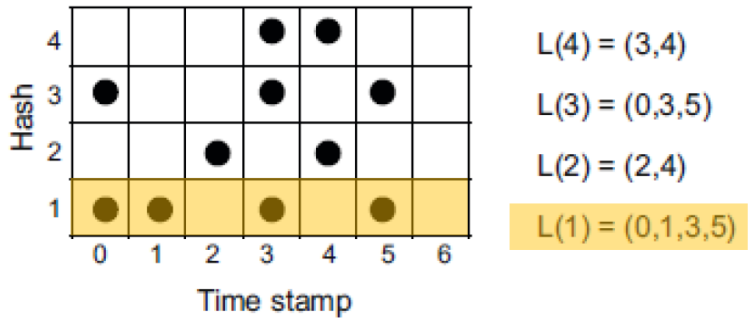
$\mathcal{F}(\mathcal{D}) =$
$\{(0,1),(1,1),(3,1),(5,1),(2,2),(4,2)(0,3),(3,3),(5,3),(3,4),(4,4)\}$
$\mathcal{F}(\mathcal{Q}) + 2 = \{(2,2),(4,2),(3,3)(3,4)(4,4)\}$
$\Delta_{\mathcal{F}}(2) = 5$

In questo caso abbiamo un'intersezione di 5 elementi, questo ci porta ad affermare che la query è molto simile al campione che stiamo analizzando.

Più in generale questo procedimento viene riprodotto su tutti i campioni del dataset e per ogni campione se ne calcola un grafico con **m** sull'asse delle x e il numero di intersezioni su quello delle y, in corrispondenza di un picco avremmo un'intersezione ricca di oggetti e quindi un buon match.



Questo metodo è funzionante, ma se il dataset nel quale dobbiamo cercare il risultato è molto grande, come di solito è, questo metodo risulta molto inefficiente.
Per risolvere questo problema si può creare una differente rappresentazione, in particolare per ogni elemento del database si prende una riga della constellation map che corrisponde ad una determinata frequenza e se ne calcola la posting list, ossia si inseriscono i punti presenti nella costellazione per quella riga. Si ripete quindi il processo per ogni riga per ottenere un risultato come il seguente.



Vediamo come si può fare il match fra una query e un campione rappresentato tramite le posting list. Nell'esempio seguente la query è la stessa di quello precedente. Dobbiamo arrivare a costruire la tabella seguente, vediamo come fare.

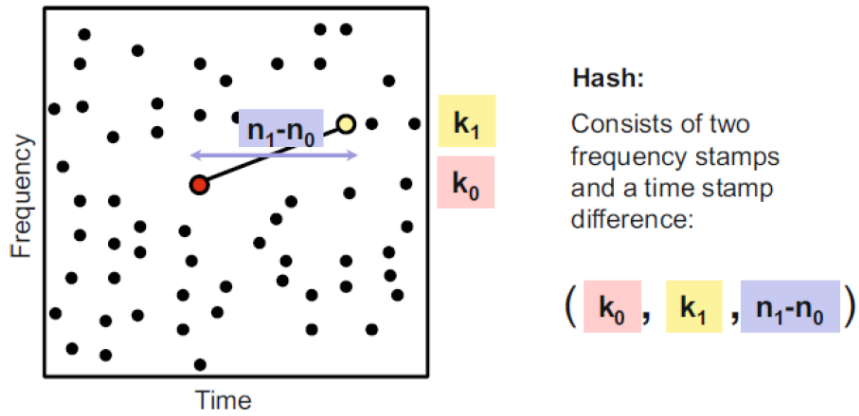
faccio partire la posting list dal punto della query

Query (n,h)	L(h) - n	Indicator functions										m
		...	-1	0	1	2	3	4	5	6	...	
(0,2)	(2,4)	0	0	0	0	1	0	1	0	0	0	
(1,3)	(-1,2,4)	0	1	0	0	1	0	1	0	0	0	
(1,4)	(2,3)	0	0	0	0	1	1	0	0	0	0	
(2,2)	(0,2)	0	0	1	0	1	0	0	0	0	0	
(2,4)	(1,2)	0	0	0	1	1	0	0	0	0	0	
Matching function		**0**	**1**	**1**	**1**	**5**	**1**	**2**	**0**	**0**	**0**	

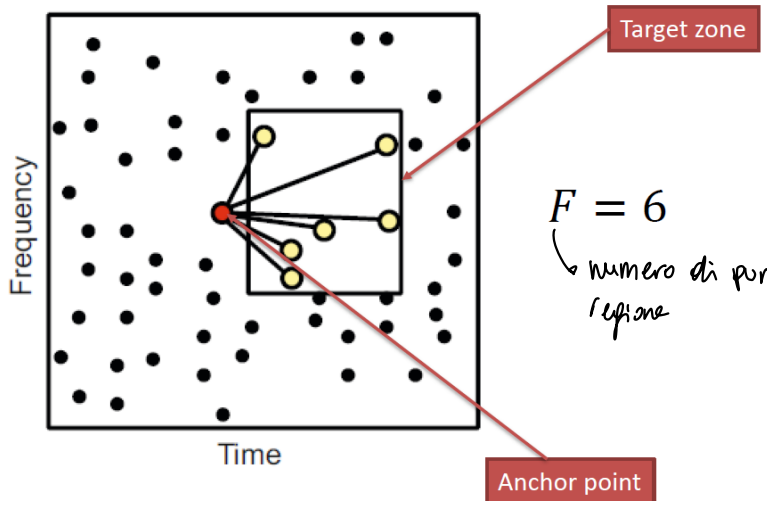
Gli indicator functions rappresentano il fattore di shit della query rispetto al campione che stiamo valutando.
Nella prima colonna inseriamo un punto per volta della query per analizzarlo, accediamo alla posting list corrispondente alla riga del punto(**h**) e andiamo a inserire nella seconda colonna i valori della posting list a cui abbiamo sottratto il valore della colonna del punto della query, quindi inseriamo L(h) – n. Successivamente segniamo un 1 per ogni colonna corrispondente al valore che abbiamo calcolato, nel primo caso inseriamo 1 nella colonna corrispondente a m=2 e M=4. Ripetiamo questo processo per ogni punto della query per ottenere tutti gli indicator function, sommiamo per colonna per ottenere la **matching function**, nell'esempio capiamo che per m=2 abbiamo un picco; infatti, spostando la query di 2 abbiamo la massima intersezione come abbiamo anche visto con il metodo precedente.

Analizziamo ora le performance che otteniamo utilizzando questo approccio. Prima di tutto definiamo con **L** il numero di frequenze e di conseguenza il numero di posting list, nel caso precedente L=4. Definiamo inoltre N come il numero di picchi all'interno di un campione del database. Assumiamo che i punti del campione siano **uniformemente distruibuiti** sul set di frequenze, allora ogni posting list conterrà approssimativamente N/L oggetti. Possiamo dunque dimostrare che abbiamo un incremento di efficienza pari ad **L.** Definiamo con **M** il numero di picchi della query, per ognuno di questi dovremo analizzare una posting list. Per questo motivo la complessità nel processare un query diventa $\frac{N}{L} \times M$ dove il primo elemento rappresenta il numero di oggetti presenti in ogni lista. Otteniamo quindi un incremento pari a L nelle performance.

Vediamo un altro modo per migliorare le performance è quello di considerare più picchi insieme, nello specifico si considerano le coppie utilizzando le due frequenze e le differenze tra i time stamp.



Ogni coppia è quindi una tripletta di dati che verrà encodata come frequenza al posto della frequenza singola. Chiaramente, se considerassimo tutte le possibili coppie di numeri otterremmo un numero di oggetti quadratico rispetto al numero iniziale, per ovviare a questo grosso problema verranno scelte solo le coppie in determinare aree.
Fissiamo quindi un punto (n0, k0) appartenente alla costellazione del sample che stiamo analizzando, lo chiameremo **anchor point**, fissiamo anche una **target region** vicina all'anchor point. La regione è un piccolo rettangolo nel piano frequenza-tempo. Se il numero di picchi è distribuito uniformemente nella costellazione e usiamo una target region sempre della stessa musura per ogni anchor point, allora ogni regione conterrà lo stesso numero di punti, chiamiamo questo punto **fan-out**.



$F = 6$

numero di punti compresi nella regione

Tramite la rappresentazione a terna, il numero di frequenze diventa **L³** rispetto a **L** di quando consideravamo la rappresentazione solo a frequenza singola. Se ad esempio utilizzassimo B byte per encodare ogni elemento della tripletta, per rappresentare le frequenze in base 2 avremmo bisogno di 2^{B+B+B} rispetto a 2^B del caso precedente. ?
Considerando poi F come fan-out e ogni punto possibile come anchor point, avremo **FM** coppie nella query, dove M è il numero di picchi nella query, e **FN** coppie nel sample, come N è il numero di picchi nel sample. Sembrerebbe che si siano effettivamente molti più punti rispetto al caso precedente ma se analizziamo la complessità possiamo vedere come le perfomence siano ulteriormente migliorate, come mostrato nella seguente figura.

The complexity of the search is now

$$\frac{M_{triple} N_{triple}}{L_{triple}} = \frac{FM\ FN}{L^3} = \frac{F^2}{L^2}\frac{MN}{L}$$

Thus, the time to process a query has been reduced by a factor of $\frac{F^2}{L^2}$.

For example, having $F = 10$ and using $L = 1024$, this results in a huge speed-up by a factor of

$$1024\text{^}2/100 \approx 10000.$$