

Information Retrieval

L'information Retrieval consiste nel recupero di informazioni rilevanti da data sources non strutturate.

Vogliamo quindi fornire l'informazione corretta e in modo efficiente all'utente.

↳ Possono essere comunque strutturate
di base, come le voci nei paper
o l'oggetto/email in una email.

Differenze tra DB e Information Retrieval:

	Databases	Information Retrieval
Unit	Structured data Clear semantics based on a formal model	Mostly unstructured data Free text with some organisation
Queries	Formally define queries Unambiguous	Vague and imprecise information needs
Results	Correct in a formal sense Closed-world assumption	Sometimes relevant , often not Open-world assumption
Match	Exact match No ranking	Best match Ranked
User Interaction	One-shot queries No interaction	Interaction is fundamental

Closed world assumption: Sappiamo che le info restituite sono connesse dal punto di vista Matematico

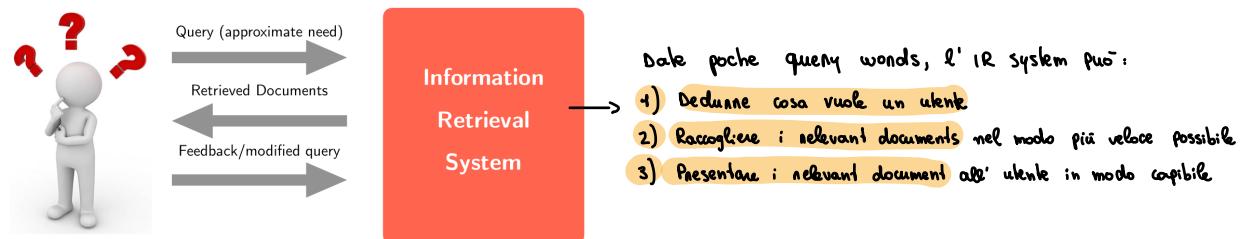
Open world assumption: le info restituite possono essere rilevanti o meno

Ranked: I documenti sono ordinati secondo quanto sono rilevanti ($\downarrow \exists$)

Non sempre: i risultati sono ciò che cerca l'utente, bisogna spesso modificare la query.

Gli algoritmi usati nell'information retrieval sono **Euristici** (cercano di produrre un risultato vicino a quello corretto)

Il processo di information retrieval è il seguente:

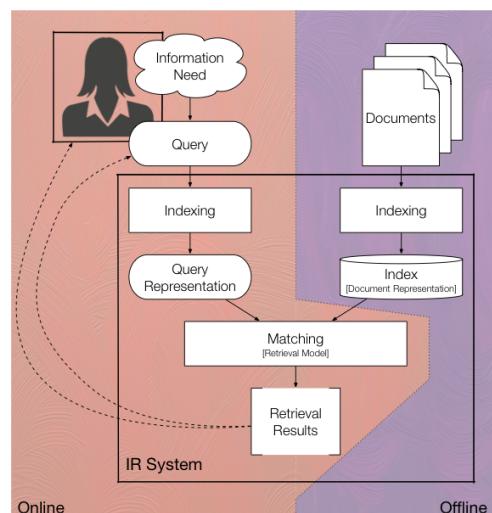


Y-Diagram of Information Retrieval

Information Need: Mancanza di conoscenza che l'utente necessita per performare un human task

Query: Traduzione da parte dell'utente dell'information need in un insieme di keywords (Query)

Relevant Document: Documento che contiene info di valore per l'utente.



Text Processing

Text Encoding

1) ASCII Character Encoding Scheme

l' ASCII encoding mappa i caratteri in 7-bit integers (2^7 caratteri = 2^7).

Un carattere viene solitamente salvato in un byte (8 bit), quindi l'ottavo bit è fillato con uno 0.

Oss: ASCII non permette di fare encoding di caratteri di altri alfabeti.

2) UTF-8 encoding scheme

UTF-8 si basa sull' UNICODE STANDARD il quale permette di rappresentare i caratteri di quasi tutti i linguaggi conosciuti.

UNICODE STANDARD:

a) Define un codespace (numerical values 0 - 10FFFF)

b) Il numero assegnato a ogni character è chiamato code point.

c) Un codepoint è denotato da "U+" seguito da un numero esadecimale lungo 4-8 digits

UTF-8 rappresenta gli unicode code points tramite sequenze di bytes (1-6 bytes).

Oss: Gli ASCII characters vengono rappresentati con 1 byte che ha il primo bit (sx) a 0.

Gli caratteri che non sono ASCII characters vengono rappresentati da 2 o più bytes: quali hanno tutti il primo bit (sx) diverso da 0.

- Il primo byte è un numero che va da 110xxxxx a 111110xx ed il numero di 1 prima dello 0 indica il numero di bytes della sequenza.
- Tutti i byte seguenti hanno i primi due bit settati come 10.

0xxxxxxx → ASCII CHARACTERS

110xxxxx ↗ 10xxxxxx

1110xxxx ↗ 10xxxxxx ↗ 10xxxxxx

11110xxx ↗ 10xxxxxx ↗ 10xxxxxx ↗ 10xxxxxx

111110xx ↗ 10xxxxxx ↗ 10xxxxxx ↗ 10xxxxxx ↗ 10xxxxxx

1111110x ↗ 10xxxxxx ↗ 10xxxxxx ↗ 10xxxxxx ↗ 10xxxxxx ↗ 10xxxxxx

Number of Bytes	Number of bits in Code Point	Range
1	7	00000000 - 0000007F
2	11	00000080 - 000000FF
3	16	00000100 - 00000FFF
4	21	00001000 - 001FFFFF
5	26	00200000 - 03FFFFFF
6	31	04000000 - FFFFFFFF

Conversione da Unicode code point a UTF-8:

- 1) Determinare il numero di bytes necessari e la forma dell' encoding →
- 2) Scrivere il code point in binario
- 3) Partendo dall' ultimo byte della sequenza, partendo dal bit meno significativo copia i bits dal code point da destra verso sinistra
- 4) Quando il byte corrente raggiunge gli 8 bits, continua a fillare i successivi byte con i successivi bit del code point
- 5) Ripeti fin quando tutti i bits non sono stati copiati nella byte sequence e filla con gli zero se necessario.

Regular Expression

Le RegEx è un linguaggio per specificare text search strings per cercare patterns in una collezione di testo (corpus).

La ricerca può ritornare tutti i match o solo il primo.

Il processo di costruzione delle RegEx expression consiste nella connessione di due elementi:

- 1) Matching di stringhe che non dovrebbero essere matchate → False Positives
- 2) Non matching di stringhe che dovrebbero essere matchate → False Negatives

Tokennization

La Tokennization consiste nel suddividere un corpus (collezione testo/speech) in tokens.

Token: Sequenza di caratteri che hanno un significato semantico.

Vocabolario: Un insieme di tutti i termini che compaiono in un corpus.

Heaps law: La grandezza del vocabolario cresce più velocemente rispetto alla radice quadrata del numero di token.

$$|V| = kN^{\beta} \text{ with empirically } 0.67 < \beta < 0.75$$

↑ grandezza vocabolario
↓ numero token

- 1) Space-based Tokenization: Separare le parole usando gli spazi e/o puntualizzazione, rimuovere la puntualizzazione, trasformare tutto lowercase
- 2) Subword Tokenization: Usare i dati per capire come tokenizzare.
 - a) Token learner: Prende il corpus e impara un vocabolario
 - b) Token segmenter: Prende una frase e la segmenta usando il vocabolario.

Byte-Pair Encoding

- Let vocabulary be the set of all individual characters
 $\{A, B, C, D, \dots, a, b, c, d, \dots\}$
- Repeat
 - i) Choose the two symbols that are most frequently adjacent in the training corpus (say A, B)
 - ii) Add a new merged symbol AB to the vocabulary
 - iii) Replace every adjacent A-B in the corpus with AB
 - iv) Until k merges have been done
- Most subword algorithms are run inside space-separated tokens:
- So we commonly first add a special end-of-word symbol _ before space in training corpus
- Next, separate into letters

Text Processing Example



Word Normalization

La word normalization consiste nel mettere le parole/tokens in un formato standard.

- 1) Ridurre tutte le lettere in lowercase
- 2) Lemmatization: Rappresentare tutte le parole con il loro lemma
- 3) Stemming: Rappresentare tutte le parole con il loro stem.
 - a) Porter stemming: È il più comune stemming algorithm per l'inglese.
 - b) È un processo invertibile
 - c) Ogni lingua ha il suo stemmer

Stop Words Removing

le stopwords sono parole estremamente comuni in un linguaggio che hanno poco valore per un task.

Di solito sono parole che fanno parte di una stopwords list oppure termini che appaiono nell' 80% o più dei documenti.

Indexes

Con "indexes" facciamo riferimento alle strutture dati disegnate per supportare la ricerca.

Esempi Sono:

- 1) **Inverted Index**: Struttura dati che fornisce il mapping tra i termini e la loro posizione nei documenti.
- a) Ogni record dell'inverted index è una associazione term - posting list.
 - b) **Posting list**: Lista di record contenenti l'associazione docID - positioni.
 - DocID = Unique number associato ad un documento
 - Positioni = Posizioni nella quale il termine in considerazione appare all'interno del documento. (il conto parte da 0) e frequenza.

OSS:

- le posting lists sono ordinate usando il DocID
- le posting lists sono iterabili sui loro postings.

Implementiamo i seguenti metodi:

p. docid(): Ritorna il docID del posting corrente

p. score(): Ritorna lo score del posting corrente

p. next(): Muove sequentialmente l'iteratore al prossimo posting

p. nextGeo(d): Avanza l'iteratore fino al prossimo posting (skipping) → se d è minore del docID corrente, che ha un DocID maggiore o uguale di d il metodo non fa nulla.

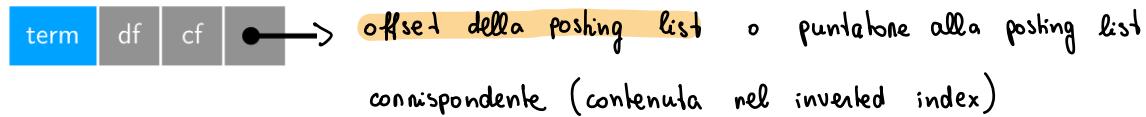
Dictionary	Posting List	Dictionary	Posting List
a	1:10,17 3:18	muscular	2:5
about	1:13	native	2:10
and	3:6	of	1:5,16 3:4
approach	3:8	on	3:12
are	2:1,9	only	1:3
australia	2:12 3:16	particularly	3:11
cat	1:19	people	3:9
closely	3:10	population	3:20
commonly	3:7	prevalent	3:19
domestic	1:18	quadrupedal	2:6
exists	3:21	quokka	1:1
fear	3:3	quokkas	3:0
genus	1:7	rottentest	3:13
have	3:1	setonix	1:8
humans	3:5	short	2:3
in	3:15	size	1:15
is	1:9	small	1:11 2:2
island	3:14	that	2:8
legged	2:4	the	1:0,2,6,14
little	3:2	to	2:11
marsupial	1:12	where	3:17
marsupials	2:7	wombats	2:0
member	1:4		

Attenzione:

- Vengono usate più inverted index con un numero costante di documenti (per farle entrare in memoria)
- le inverted indexes vengono salvate in forma compressa (per farle entrare in memoria)

- 2) **Vocabolario**: Struttura dati contenente un record per ogni distinct term all'interno della collezione. Per ogni termine salva inoltre:

- a) Contiene statistiche sui termini, come la document frequency (numero di documenti nei quali il termine compare almeno una volta)
- b) Contiene una lookup table tra i termini e la posting list corrispondente

 term df cf → offset della posting list o puntatore alla posting list corrispondente (contenuta nel inverted index)

OSSERVAZIONI:

- a) Viene acceduta con accesso randomico
 - b) Viene implementata tramite: Dizionario / Hash table
 - c) Contiene il mapping term - termID
- 3) Document Index: Struttura dati contenente un record per ogni documento.

 docid len PR URL docno Per ogni documento salva info come: l' URL, titolo, lunghezza, etc --

- a) Contiene il mapping document - Doc ID (hash table)
- b) Dato un DocID dobbiamo essere capaci di recuperare l' URL
- c) I record sono ordinati tramite il DocID
- d) Viene implementato tramite un array (perché sappiamo il numero di documenti)
- e) DocNo = Nome (stringa) dato ad un documento usato per valutare la qualità del search engine (può essere usato anche come URL)

OSSERVAZIONI GENERALI:

- 1) le strutture viste vengono salvate su disco (un file per ogni struttura)
- 2) L'ordine di grandezza delle data structures è il seguente:
Inverted index > vocabolario > document index
- 3) Il vocabolario salva le term frequencies, il document index salva la doc. length.

4) Esistono due strutture dati ulteriori (salvate in file diversi su disco):

- Collection Statistics: Contiene info come n° totale di doc, n° totale di terms
- Direct Index: effettua il mapping documents - termID & frequencies

Indexing

Come fanno l'indexing dei documenti?

1) In-Memory Indexing

Crea un inverted index passando un doc.

per volta e mantenendo in memoria l'intero

inverted index e vocabolario.

Svantaggi:

- le posting list finali sono incomplete fino alla fine.
- Richiede molto spazio, 8 bytes per ogni (termID, docID)
- Non scala perché non è possibile salvare l'intera collezione in memoria, sono scritte nel disco.
- Non è eseguibile in parallelo (check sul lexicon e inserimento non lavorano bene in parallelo)

```
procedure BUILDINDEX(D)
    I ← HashTable() → Inverted Index
    n ← 0 → Doc IDs
    for all documents d ∈ D do
        n ← n + 1 → Doc ID
        T ← Parse(d) → Tutti i token in D
        Remove duplicates from T → Trova i termini distinti
        for all tokens t ∈ T do → Vi termini
            if  $I_t \notin I$  then
                 $I_t \leftarrow$  Array() → Does not work well in parallel
            end if
             $I_t.append(n)$  → Crea una posting list
        end for
    end for
    return I
end procedure
```

Se non esiste l'index per il termine t

append il doc alla posting list relativa a quel termine

2) Blocked Sort-Based Indexing

- Definiamo dei blocchi di record e li processiamo.
- Si creano le posting lists, le sortiamo e le salviamo su disco → molti sorting! costoso
- Alla fine usiamo il multi-way merge dove apriamo tutti i file su disco, e mengiamo le posting list per ogni termine.

BSBINDEXCONSTRUCTION()

- $n \leftarrow 0$
- while (all documents have not been processed)
- do $n \leftarrow n + 1$
- $block \leftarrow \text{PARSENEXTBLOCK}()$
- $\text{BSBI-INVERT}(block)$
- $\text{WRITEBLOCKTODISK}(block, f_n)$
- $\text{MERGEBLOCKS}(f_1, \dots, f_n; f_{\text{merged}})$

Index A aardvark | 2 | 3 | 4 | 5 | apple | 2 | 4 |

Index B aardvark | 6 | 9 | actor | 15 | 42 | 68 |

Index A aardvark | 2 | 3 | 4 | 5 |

apple | 2 | 4 |

Index B aardvark | 6 | 9 | actor | 15 | 42 | 68 |

Combined index aardvark | 2 | 3 | 4 | 5 | 6 | 9 | actor | 15 | 42 | 68 | apple | 2 | 4 |

Il problema di questo algoritmo è che dobbiamo mantenere l'intero dizionario in memoria.

Perché dobbiamo sapere il mapping termid-shinga per selezionare i termini e tutte le altre operazioni.

Due possibili idee:

- Creare un dizionario per ogni blocco e fare il merge
- Accumuliamo i posting senza sottrarli

Queste due idee vengono usate nell'algoritmo SPIMI.

3) Single-Pass In-Memory Indexing (SPIMI)

```
SPIMI-INVERT(token_stream)
1 output_file = NEWFILE()
2 dictionary = NEWHASH()
3 while (free memory available)
4 do token ← next(token_stream)
5   if term(token) ∉ dictionary
6     then postings_list = ADDTO DICTIONARY(dictionary, term(token))
7     else postings_list = GETPOSTINGSLIST(dictionary, term(token))
8   if full(postings_list)
9     then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10    ADDTOPOSTINGSLIST(postings_list, docID(token))
11 sorted_terms ← SORTTERMS(dictionary)
12 WRITETOBLOCKTODISK(sorted_terms, dictionary, output_file)
13 return output_file
```

a) Per ogni blocco genera il vocabolario e l'inverted index

↳ Quando la mem. è piena, il vocabolario viene sottratto (le posting lists no) e salva

sia il vocabolario che l'inverted index su disco.

b) Viene fatto il merge del vocabolario e degli inverted index.

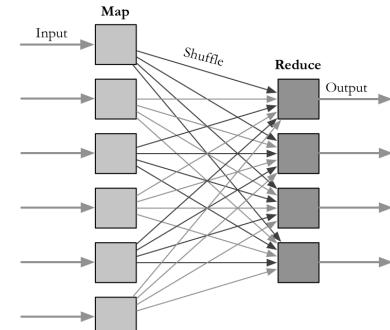
4) Map Reduce Indexing

Essendo la quantità di documenti molto grande abbiamo bisogno di un sistema

distribuito (fault tolerant) per effettuare l'indexing.

```
procedure MAPDOCUMENTSTOPOSTINGS(input) → Gruppo di docs
  while not input.done() do → Per ogni documento
    document ← input.next() → prendi doc
    number ← document.number → docNo
    position ← 0 → Posizione del termine nel doc.
    tokens ← Parse(document) → Tokens nel doc.
    for each word w in tokens do → Per ogni parola trovata
      Emit(w, document:position) ]→ emit (term, doc : position)
      position = position + 1
    end for
  end while
end procedure
```

```
procedure REDUCEPOSTINGSTOSETS(key, values) → Key = word
  word ← key
  WriteWord(word) → Nel vocabolario
  while not input.done() do
    EncodePosting(values.next()) → popola la posting list relativa alla word
  end while
end procedure
```



Index Update

La collezione dei documenti cresce e cambia nel tempo.

1) **Static collections**: Sono collezioni che hanno pochi cambiamenti e possono essere re-indexed ogni volta che avviene un cambiamento.

L'update del inverted index file non viene preso in considerazione perché richiede scrittura nel metto del file.

2) **Dynamic Collections**: Collezioni che cambiano spesso e il re-indexing non è un'opzione.

a) **Index Merging**: I nuovi documenti vengono indexati e l'index esistente viene unito con il nuovo (usato quando arrivano molti documenti ogni volta).

b) **Result Merging**: Viene mantenuto un index più piccolo e separato in memoria. Le queries vengono processate su entrambi gli index, unendo i risultati. (Quando arrivano pochi documenti per volta)

Attenzione: Quando uno o più documenti devono essere eliminati dalla collezione di documenti, essi vengono salvati in una deletion list (implementata come bloom filter) e i documenti vengono rimossi dai risultati (facendo un check nella deletion list) prima di essere restituiti.

→ Oss: Implementando la deletion list come bloom filter rischiamo al massimo di non restituire un documento che in realtà non è eliminato.

Query Processing

le query possono essere risolte in modo:

- 1) **Conjunctive**: Restituisce i documenti che hanno tutti i query terms
- 2) **Disjunctive**: Restituisce i documenti che hanno tutti i query terms o un sottinsieme di essi

Il retrieval dei documenti può avvenire:

- 1) **Boolean Retrieval**: Tutti i documenti che matchano la query
- 2) **Ranked Retrieval**: Un sottinsieme di K documenti più rilevanti secondo una scoring function

$$s_q(d) = \sum_{t \in q} s_t(q, d)$$

La scoring function assegna un numero reale (score) che misura quanto bene un documento e una query matchano.

Come accedere le posting list per il query processing?

- 1) **Term at a Time (TAAT)**

Computo lo score un termine per volta (scorrendo seq. la posting list) e lo salvo nell' accumulator.

PRO:

- a) Semplice

- b) Cache - Friendly (dovuto all'accesso sequenziale)

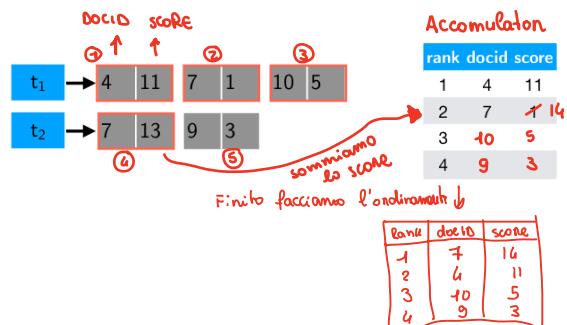
CONS:

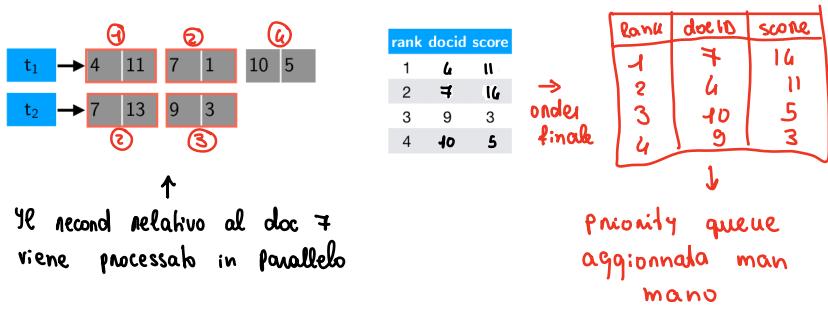
- a) Richiede molta memoria per contenere i partial scores di tutti i documenti
- b) Difficoltà nel fare boolean o phrasal queries (perché considera un term. per volta)

- 2) **Document at a Time (DAAT)**

Vengono aperte tutte le posting list, dei termini della query, contemporaneamente.

le posting list vengono attraversate in parallelo, document by document, computando direttamente la somma (non uso l'accumulazione)





PRO:

- a) Memory footprint più piccolo, perché non ci sono valori intermedi da salvare essendo che le somme vengono subito calcolate dallo scorrimento in parallelo delle posting lists
 - b) Supponendo per le boolean e phrasal queries (perché tutti i puntatori devono puntare allo stesso min docID)
- Contra: Meno cache-friendly

Query Processing Distribuito

Essendoci la necessità dell'indexing distribuito, anche il query processing lo deve essere.

- 1) Tutte le query vengono inviate ad una macchina centrale, chiamata Manager machine.
- 2) Il manager manda messaggi ai vari index servers.
- 3) Ogni index server fa una porzione del query processing.
- 4) Il manager riceve i risultati, gli organizza e li ritorna all'utente.

Come distribuire i documenti agli index servers:

- 1) Document Distribution
 - a) Ogni index server riceve una frazione della totale collection
 - b) Il manager invia una copia della query ad ogni index server, ognuno restituisce i top results basandosi sui documenti assegnati ad esso
 - c) I risultati vengono uniti in una singola linked list dal manager e restituisce i top K.

PROBLEMA: le collection statistics dovrebbero essere condivise per un ranking effettivo e non locale per quel dato server.

2) Term distribution

- Viene costituito un single index e assegnato a ogni index server.
 - Uno degli index server viene scelto per processare la query mentre gli altri index server inviano informazioni a quest'ultimo.
oss: Viene scelto quello che ha la più lunga inverted list
 - Il risultato finale viene inviato al manager
- PROBLEMA: Il load balancing dipende sulla distribuzione dei query terms e le loro co-occurrences.

Caching

Il caching può migliorare significativamente l'effectiveness del query processing.

- Cache popula query results → Migliore
- Cache common inverted lists → Utile con query uniche

oss: La cache deve essere refreshata per prevenire dati obsoleti

Scoring in Ranked Retrieval

Come calcolare la scoring function $s: Q \times D \rightarrow \mathbb{R}$ usata per il ranking dei documenti?

Consideriamo il modello bag of words, cioè non consideriamo l'ordinamento delle parole.

Luhn Assumption: Il peso di un termine è proporzionale alla sua raw (absolute) frequency.
→ Questa assunzione si basa sul fatto che uno scrittore solitamente ripete alcune parole importanti per l'argomento su cui scrive.

La raw term frequency non è ciò che vogliamo perché la rilevanza di una parola non cresce proporzionalmente con la term frequency (tf).

Usiamo quindi il logaritmo della term frequency, la quale è una funzione che non cresce proporzionalmente con la term frequency:

$$w_{t,d} = \begin{cases} 1 + \log(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

↑
Posso aprire solo le posting

Se il termine non appare nel documento \rightarrow list dei query terms

PROBLEMA: Non vengono presi in considerazione i **RARE TERMS** i quali sono più informativi dei frequent terms e devono avere un peso maggiore.

→ Termini che hanno una **document frequency (df_t) bassa**

Usiamo quindi la **inverse document frequency** del termine t:

$$idf_t = \log \frac{N}{df_t}$$

dove:

- df_t è Numero di documenti nei quali t compare
- N è il totale numero di documenti

l' **IDF** misura quanto un termine è comune o raro in un corpus. (più IDF è alto, più il termine è raro).

Usando quindi la **TF** e l' **IDF** insieme, otteniamo il **TF-IDF**, che riflette quanto è relevant un termine in un corpus

$$w_{t,d} = \begin{cases} (1 + \log(tf_{t,d})) \log \frac{N}{df_t} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

TF • IDF
Facciamo la moltiplicazione

Lo scone per una document-query pair è quindi data dalla somma sui termini t che compaiono sia in q che in d della TF-IDF measure:

$$s(q, d) = \sum_{t \in q \cap d} w_{t,d}$$

Vector space models

Se salvo i valori $w_{t,d}$ nel lexicon, abbiamo che un documento è un vettore di numeri reali nello spazio vettoriale di dimensione pari alla dimensione del lexicon $|V|$.

Posso quindi rappresentare anche le query nello stesso spazio e rankare i documenti in base alla loro prossimità alla query nello spazio.

Come computare la prossimità?

1) Euclidean distance: Pessima idea perché è grande tra vettori di diversa lunghezza,

Ese:



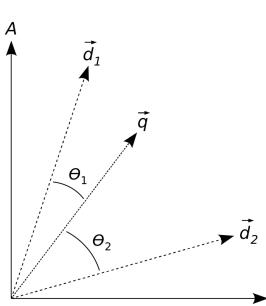
quindi la distanza tra la query e due documenti semanticamente identici ma di diversa lunghezza, è diversa. (Misura non invariante)

2) Cosine Similarity: Usa l'angolo tra due documenti il quale vale 0 quando la

similarità è massima.

l'angolo tra la query e due documenti semanticamente uguali è sempre uguale (Misura invariante).

$$s(q, d) = \cos(\theta(\vec{q}, \vec{d})) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \|\vec{d}\|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$



- Ye denotazione rappresenta la lunghezza della normalizzazione (rimozione delle info ridondanti) delle query e document vectors
- Nonnormalizzando, i documenti lunghi e corti hanno pesi comparabili
- Rispetto al TF-IDF fa anche la normalizzazione

Ranking Evaluation

Abbiamo bisogno di valutare la qualità del ranking per poterlo migliorare.

Ye ranking può essere valutato secondo due misure:

1) Efficienza

2) Effettività

- usando
- | | |
|--|---|
| a) Benchmark document collection (corpora) | { |
| b) Benchmark suite of queries (Topics) → users needs | |
| c) Una valutazione (assessment) di rilevante / non rilevante | |
- per ciascuna query e documento

Questi elementi vengono usati da uno standard tool chiamato TREC il quale fornisce un measure score secondo uno o più metiche.

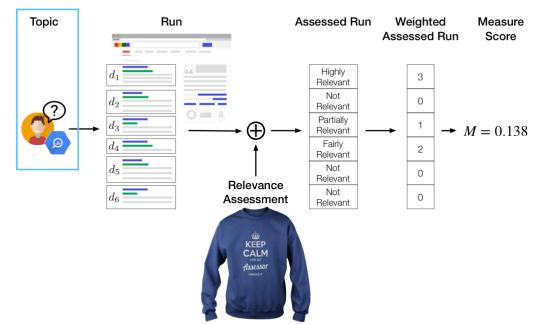
Questo valore può essere poi confrontato tra due IR systems per valutare quale si comporta meglio.

1) Ho un topic e l'IR genera una run, cioè una lista di documenti.

2) Usando il relevance assessment capiamo quali docs scelti sono rilevanti e quali no e generiamo una così detta assessed run.

3) Convertiamo i judgement nell'assessed run in dei valori numerici e successivamente generiamo una valutazione

4) Questo numero deve essere valutato rispetto ad un altro search engine sullo stesso topic per capire se è veramente buono o meno.



Topic

Un topic consiste in un file XML composto da:

- 1) Topic ID
- 2) Title: Breve descrizione dell' information need
- 3) Description: Descrizione più dettagliata dell' info. need
- 4) Narrative: Istruzioni per l'assessore su quando considerare un documento rilevante

Oss: Solitamente nelle experimental collections si usano ≈ 50 topics.

Relevance Assessment (judgement) - qrels

Ogni benchmark document per ogni benchmark query deve avere un relevance judgement.

- a) Binary Relevance: Rilevante = 1, non rilevante = 0 \Rightarrow Misura solo se è rilevante il documento per la query

- b) Graded Relevance : La scala di rilevanza va da 0 a 5 \Rightarrow Indica un grado di rilevanza per un documento rispetto alla query

Un relevant judgement viene dato dai relevant assessors, cioè persone che valutano quanto/se il documento è rilevante.

Esempio Relevant Assessment file

- Topic ID
- Fixed : Vale sempre 0
- DocNo : Non possiamo usare il docID perché è un ID interno allo specifico IR, non generale.
Viene quindi usato il DocNo
- Judgement : Binario o Ranked

OSS : • Field separati da tab/space
• 300-700 file per ogni topic

Topic ID	Fixed	Doc No	Judgement
41	0	LA050394-0237	0
41	0	LA112394-0177	0
41	0	LA091294-0164	1
41	0	LA040594-0187	0
41	0	LA041694-0248	1
			...
42	0	LA031694-0234	0
42	0	LA040494-0111	0
42	0	LA081794-0171	1

Pooling

Avere un judgement per tutti i documenti in una collezione è impossibile.

Viene quindi usata una pooling technique :

- 1) Viene data la stessa query a diversi IR systems e da ognuno ottieniamo i top K results
- 2) I risultati vengono uniti in una pool e vengono rimossi i duplicati
- 3) Vengono man mano estratti i documenti dalla pool in modo randomico e presentati ai relevance assessors.

OSS : I documenti che non sono stati giudicati dai relevant assessors vengono considerati non rilevanti!

Run Execution Example

Dall'esecuzione di una run otteniamo un file così composto:

Topic ID	Fixed	Doc No	Rank	Score	Run ID
41	Q0	LA050394-0237	1	0.6342	updrun
41	Q0	LA091294-0164	2	0.4278	updrun
41	Q0	LA040594-0187	3	0.4278	updrun
41	Q0	LA041694-0248	4	0.3197	updrun
41	Q0	LA102394-0113	5	0.3005	updrun
			...		
42	Q0	LA081794-0171	1	0.7687	updrun
42	Q0	LA031694-0235	2	0.7011	updrun
42	Q0	LA031694-0234	3	0.6950	updrun

nome dato al file

Metrics

le metriche usate in fase di valutazione possono essere di due tipi:

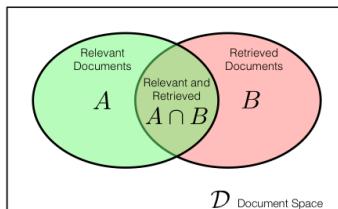
- 1) Set-based Metrics
- 2) Rank-based Metrics

Set-Based Metrics

Sono metriche che non prendono in considerazione l'ordine dei documenti.

- 1) **Precision**: Quanti documenti restituiti sono ^{davvero} rilevanti rispetto al totale di docs restituiti rilevanti
- 2) **Recall**: Quanti dei documenti rilevanti sono stati restituiti rispetto al totale di docs rilevanti.

Precision e Recall insieme misurano la retrieval effectiveness, cioè l'abilità del sistema di restituire documenti rilevanti e non restituire quelli non rilevanti.



$$\text{Precision } P = \frac{|A \cap B|}{|B|} = \frac{TP}{TP+FP}$$

$$\text{Recall } R = \frac{|A \cap B|}{|A|} = \frac{TP}{TP+FN}$$

3) **F-measure** (F_1 -score) : È una misura che combina precision e recall,

$$F = 2 \frac{P \cdot R}{P + R} = \frac{|A \cap B|}{\frac{|A| + |B|}{2}}$$

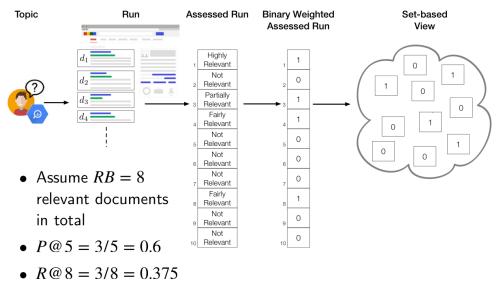
svolgendo l'harmonic mean tra esse.

Rank-based Metrics

Iniziamo con il definire due metriche importanti:

a) **Precision at cutoff k** : $P@k = \frac{1}{k} \sum_{i=1}^k r_i$

b) **Recall at cutoff k** : $R@k = \frac{1}{RB} \sum_{i=1}^k r_i$



Dove :

- $r_i \in \{0,1\}$ è il relevance score dell'i-esimo documento
- RB è la recall base, cioè il numero totale di relevant documents nella collection

Definiamo ora le rank-based metrics:

a) Average Precision

a) Considera la rank position di ogni relevant document ($k_1, \dots, k_5, \dots, k_{RB}$)

* b) Calcola $P@k$ per ogni k value

c) Fai la media di questi valori (Average Precision)

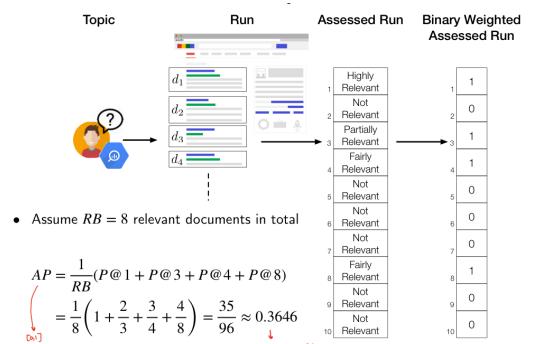
d) Calcola la media degli average precision ottenute

sull'insieme di topic (Mean Avg Precision - MAP)

OSS:

* 1) "Nella realtà" non viene considerata la rank position di ogni relevant document, ma solo fino al cutoff k ($k_1, \dots, k_5, \dots, k_k$).

2) L'area under the precision-recall curve (AUC), la quale assume il trade-off



fra precision e recall, equivale all' average precision (AP).

$$AUC = \sum_k P@k \Delta R_k$$

$\Delta R_k = R_k - R_{k-1}$

→ Dimostrazione :

a) sia AUC definita come $AUC = \sum_{n=1}^N P@n (R@n - R@(n-1))$

- b) Quando l' n -th document non è rilevante, $R@n$ è uguale a $R@(n-1)$ e la loro differenza è 0

Possiamo quindi sommare solo nell'insieme delle rank positions dei relevant

achieved documents : $AUC = \sum_{k \in \mathcal{R}} P@k (R@k - R@(k-1))$

↳ Considero solo le rank position dei relevant docs

- c) Due rank positions adiacenti differiscono di un solo relevant document

$$R@k - R@(k-1) = \frac{1}{RB}$$

Quindi :

$$AUC = \frac{1}{RB} \sum_{k \in \mathcal{R}} P@k = AP$$

- 3) AP prende in considerazione solo il binary score relevance (rilevante o non rilevante), non diversi gradi di essa da [0,5]

- 2) Normalised Discounted Cumulative Gain ($nDCG$)

Definiamo DCG, la quale è una metrica che prende in considerazione la patientia dell'utente nello scommettere la result list

$$DCG@k = \begin{cases} \sum_{i=1}^k r_i & \text{if } k < b \\ DCG@(k-1) + \frac{r_k}{\log_b k} & \text{if } k \geq b \end{cases} = \sum_{i=1}^k \frac{r_i}{\max(1, \log_b k)}$$

due : • $b = 2$ is an **impatient user**
• $b = 10$ is a **patient user**

L'idea è di andare a ridurre la rilevanza dopo la b -esima posizione per modellare

la perdita di patienta dell'utente.

Oss: DCG può essere usato con diversi gradi di rilevanza.

Problema: DCG non è un valore limitato
 ↓

Soluzione:

Per normalizzare DCG o K dobbiamo calcolare

la ideal run, cioè ordinare i relevant document

in ordine discendente basato sul rank k.

Questo ci permette di calcolare il massimo valore

di DCG@K (best possible retrieval) chiamato

i NDCG@K.

Il nonnormalised Discounted Cumulative Gain at cutoff k è dato da:

$$nDCG@k = \frac{DCG@k}{iDCG@k}$$

3) Rank-Biased Precision

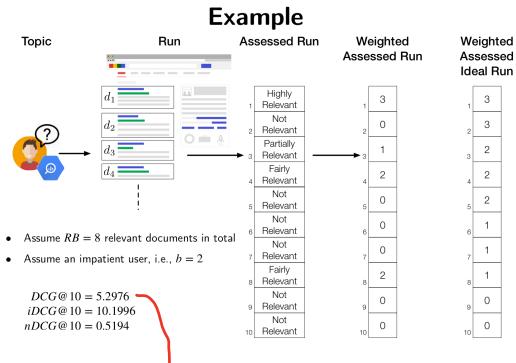
La rank-biased precision metric si basa sull'idea che un utente inizia a sconfinare i documenti dal primo ranklab e poi passa al successivo con probabilità p (chiamata persistence) o si ferma con probabilità 1-p.

$$RBP@k = (1-p) \sum_{i=1}^k p^{i-1} r_i = (1-p) \sum_{k \in \mathcal{R}} p^{k-1}$$

→ Avg precision accumulata
nella ranked exploration
dell'utente al cutoff k

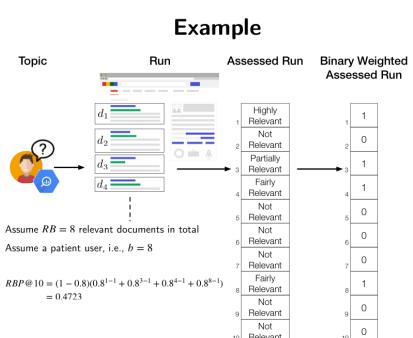
I valori tipici per p sono:

- 0.5 per gli impatient users
- 0.8 per i patient user
- 0.95 per gli extremely patient users



calcolato così:

$$DCG@10 = 3 + \frac{1}{\log_2 3} + \frac{2}{\log_2 4} + \frac{2}{\log_2 8} = 5.2976$$



4) Mean Reciprocal Rank

Supponiamo che esista un solo doc. rilevante e sia K la sua posizione.

Il Reciprocal Rank Score è calcolato come:

$$\text{Reciprocal Rank score } RR = \frac{1}{K} \quad \rightarrow \quad \begin{array}{l} \text{es: } K=1 \text{ (prima posizione)} \rightarrow RR=1 \\ \text{es: } K=2 \text{ (seconda posizione)} \rightarrow RR=\frac{1}{2} \\ \vdots \end{array}$$

Per queries multiple, faccio la media (Mean Reciprocal Rank)

User Models

le rank-based evaluation Measures usano, implicitamente o esplicitamente, uno user model che comprende:

- Browsing Model: Descrive come l'utente interagisce con i risultati.
- Model of document utility: Descrive quanto è utile un documento rilevante per un dato utente.
- Utility Accumulation Model: Descrive quanto un utente accumula utilità scrollando i risultati.

Nel caso di DCG:

- 1) Browsing Model: L'utente sconce i risultati uno per volta finché non raggiunge il K^* , il quale sarà scelto con prob. proporzionale al logaritmo del rank.
- 2) Model of document utility: L'utente ottiene "qualcosa" da ogni relevant document in proporzione al suo relevance degree.
- 3) Utility Accumulation Model: Un utente ottiene "qualcosa" da ogni relevant doc. $t \rightarrow K$

Significance testing

su un certo numero di query

Dati i risultati ottenuti da due differenti ranking algorithm (A e B), come posso essere sicuro che B (nuovo algo) sia meglio di A (baseline)?

uso un significance test, il quale è composto da:

- Null hypothesis: Non c'è differenza statistica tra A e B
- Alternative hypothesis: B è meglio di A

OSS: la potenza del test aumenta quanto più query vengono usate.

Procedimento:

- Calcola la effectiveness measure per ogni query, sia per A che per B
- Computa un test statistico basato sul confronto dell'effectiveness per ogni query, e ottieni come risultato il p-value (probabilità che un test statistic value almeno così estremo possa essere osservato se la null hypothesis fosse vera).
- La null-hypothesis è rifiutata se p-value < α (significance level).

I valori di α sono solitamente piccoli, tipicamente 0.05 o 0.01.

T-test

1) Il test statistico è $t = \frac{\bar{B} - \bar{A}}{\sigma_{B-A}} \sqrt{N}$

- Chechiamo la t-test table per ottenere il p-value

- Se il p-value < α , B è statisticamente migliore di A con un confidence level α .

Problema: I valori per ogni query non sono correlati!

query	A	B	$B - A$
1	25	35	10
2	43	84	41
3	39	15	-24
4	75	75	0
5	43	68	25
6	15	85	70
7	20	80	60
8	52	50	-2
9	49	58	9
10	50	75	25
mean	41.1	62.5	

• Significance level $\alpha = 0.05$
• Is system B better than system A?
• The test statistic is
$$t = \frac{\bar{B} - \bar{A}}{\sigma_{B-A}} \sqrt{N}$$

$$\bar{B} - \bar{A} = 21.4, \sigma_{B-A} = 29.1, N = 10, t = 2.33$$

• We check the t-test table and get a p-value of 0.02
• $0.02 < 0.05$: system B is statistically significantly better than system A at a significance level of .05

Wilcoxon signed-rank test

1) Calcola la differenza, in absolute value, dei valori senza zero a sinistra (non-zero diff.)

$$\text{es: } ||0.01 - 0.029|| = ||-1 - 29|| = 28$$

2) Ordina i valori ottenuti

3) Computa il connispettivo signed rank.

g tie (pareggi) vengono gestiti assegnando un avg value per entrambi.

$$\text{es: } \begin{array}{l} 2, 9, 10, \underline{24}, 25, 25, 41, 60, 70 \\ \text{hanno diff negativa } (B-A < 0) \end{array} \rightarrow \begin{array}{l} -1, +2, +3, \underline{-4}, +5.5, +5.5, +7, +8, +9 \\ \text{e' assegnato ad entrambi} \end{array}$$

4) Somma tutti i rank values (35) e checkla la Wilcoxon table per ottenere il p-value corrispondente.

5) Se il p-value < α , B è statisticamente migliore di A con un confidence level α .

query	A	B	B - A	
1	25	35	10	i) • Compute the non-zero differences in rank order of absolute value:
2	43	84	41	ii) • <u>2, 9, 10, 24, 25, 25, 41, 60, 70</u>
3	39	15	-24	iii) • Compute the corresponding signed ranks (check the tie-breaking):
4	75	75	0	iv) • <u>-1, +2, +3, -4, +5.5, +5.5, +7, +8, +9</u>
5	43	68	25	v) • The sum gives a vale of 35
6	15	85	70	vi) • We check the Wilcoxon singed-rank test table and get a p-value of 0.02
7	20	80	60	vii) • 0.02 < 0.05: system B is statistically significantly better than system A at a significance level of .05
8	52	50	-2	
9	49	58	9	
10	50	75	25	

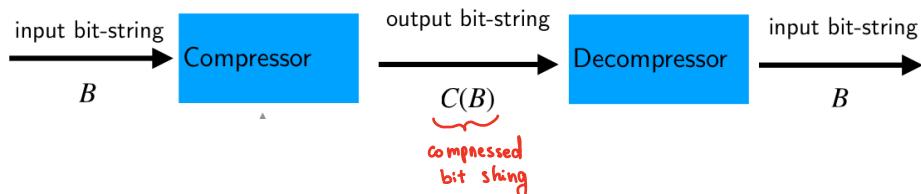
Index Compression

(reversible)

La data compression è un processo nel quale i dati vengono trasformati in un'altra rappresentazione

Per:

- Salvare spazio
- Salvare tempo di trasmissione



Il compression ratio (CR) indica che il compressed output è r volte più piccolo dell'input.

$$CR = |B| / |C(B)| = r$$

Proposizione: Nessun algoritmo può comprimere qualsiasi bit-string perché se esistesse allora potremmo utilizzarlo ricorsivamente fino ad ottenere 0 bit.

Perché siamo interessati alla compressione?

Vogliamo poter comprimere:

- a) **Vocabolario:** Per tenerlo in main memory e poter tenere anche alcune posting lists
- b) **Posting list**
 - Per ridurre il disk space necessario per salvare
 - Ridurre il tempo necessario per leggerla da disco
 - Salvare una porzione significativa di esse in memoria

Distribution Laws

Abbiamo studiato due leggi che ci permettono di predire in modo empirico la grandezza del vocabolario (Heap's law) necessaria per rapp. un dato corpus e la collection frequency di un termine (Zipf's Law).

Heap's Law

Siano M e T definite come segue:

- $M = |\text{v}_t| \rightarrow \text{Numero di token distinti nella collezione}$
- $T \rightarrow \text{Numero totale di token nella collezione}$

La Heap's law è definita come segue:

$$M = kT^b \quad \text{dove} \quad 30 \leq k \leq 100 \text{ and } b \approx 0.5$$

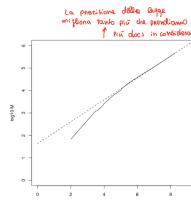
- For the RCV1 collection, the dashed line is the best least square fit:

$$\log_{10} M = 0.49 \log_{10} T + 1.64$$

Hence

$$M = 10^{1.64} T^{0.49}$$

- For the first 1,000,020 tokens, Heap's law predicts 38,323 terms and, actually, we have 38,365 terms



Possiamo trasformare la formula da moltiplicazione a somma (reducendo l'exec. time della computazione) usando i logaritmi (log-log space)

$$\log M = \log k + b \log T$$

Zipf's Law

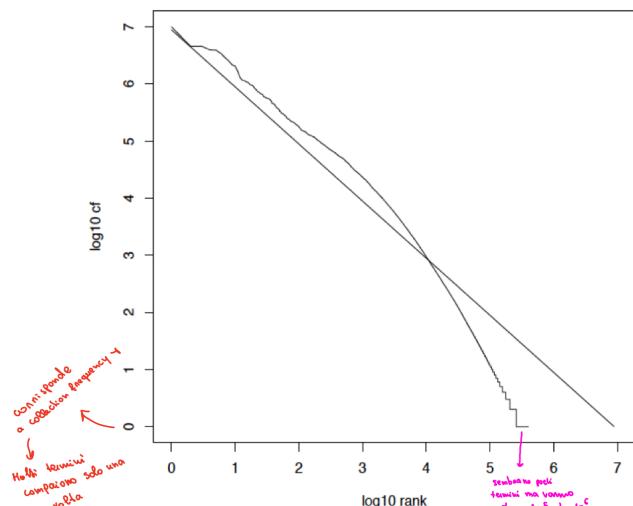
Siano i termini in una sequenza ordinata per absolute frequency (descending), la zipf's law ci dice che l'i-th termine più frequente ha una frequency proporzionale a $\frac{1}{i}$.

$$cf_i \propto \frac{1}{i} = \frac{K}{i}$$

dove:

- K è la normalising constant
- cf_i è la collection frequency, cioè il numero di occorrenze del termine t_i nella collezione

Nel log-log space ottieniamo: $\log cf_i = \log K - \log i$



OSS: Analizzando le due leggi, abbiamo come conseguenza che la dimensione del vocabolario continua ad aumentare con l'aumentare dei documenti nella collezione (no upper bound). Di conseguenza, il vocabolario necessita di essere compresso (ma non lo studieremo).

Integer compression

Il più dello spazio è occupato dalle posting list perché ci sono termini molto frequenti nella collezione che hanno posting lists molto lunghe.

Di conseguenza studieremo due integer compression techniques:

- 1) Static integer compression
- 2) Sorted integer list compression

Static Integer Compression

Dato un intero $x > 0$, vogliamo disegnare un algoritmo (code) che rappresenta x con il minor numero di bits.

- L'output dell'algoritmo dato x è chiamato codeword of x , e denotato con $C(x)$.
- Un messaggio $L = [x_1, x_2, \dots, x_n]$ viene codificato come la concatenazione delle codewords di x_1, \dots, x_n , cioè $C(x_1), \dots, C(x_n)$.
- Un code è chiamato statico se assegna sempre la stessa codeword $C(x)$ a x .

Binary code compression

Sia $\text{bin}(x)$ il numero minimo di bits necessari per rappresentare x , cioè $\lceil \log_2(x+1) \rceil$, essendo che assumiamo $x > 0$, abbiamo che $B(x) = \text{bin}(x-1)$ è il binary codeword assegnato a x dal binary code.

Attenzione: $B(x)$ rappresenta il numero minimo di bits per rappresentare x , cioè è un lower bound per il numero di bits di $C(x)$.

x	$B(x)$
1	0
2	1
3	10
4	11
5	100
6	101
7	110
8	111

Problema: Non possiamo usare la binary code compression perché non è possibile decodificare il messaggio, dovuto alle troppe ways possibili di decodifica (il code è ambiguo).



Solutions: Prefix-free codes

Y prefix-free codes sono codes dove nessuna codeword è un prefisso di un'altra codeword, in questo modo possiamo decodificare senza ambiguità.

Un code C è detto prefix-free quando non esistono $c(x)$ e $c(y)$, con $|c(y)| \geq |c(x)|$, t.c. $C(x) = C(y)[0 : |C(x)| - 1] \rightarrow$ i primi $|c(x)| - 1$ bits non sono uguali a $c(x)$, cioè $c(x)$ non è un prefisso di $c(y)$.

Unary Code

L' unary code è un prefix-free code, il quale rappresenta $x > 0$ come $U(x) = \underbrace{1 \dots 1}_{x-1} 0$, cioè una serie di $x-1$ uni e uno zero finale.

Quindi $|U(x)| = x$, di conseguenza è buono solo per interi piccoli.

Example: $U(234) =$

11
 11
 11
 11
 1110

\rightarrow 234 bit per rappresentarlo,
molte di più di 32 bit
usate per rapp. un intero.

x	$U(x)$
1	0
2	10
3	110
4	1110
5	11110
6	111110
7	1111110
8	11111110

Gamma e Delta Codes

Gamma e delta sono prefix-free codes.

1) Gamma: Scrive $b = |\text{bin}(x)|$ bits usando l'unary code

seguiti da $b-1$ bits meno significanti di $\text{bin}(x)$.

Oss: a) $|\gamma(x)| = 2|\text{bin}(x)| - 1$ bits \rightarrow factor of 2 dist. dall'ottimo.

b) $b-1$ bits perché il primo è sempre uguale a "1".

L' + viene aggiunto durante la decodifica.

c) y bit a destra del . possono essere al massimo 32 (dimensione di un intero), ma

per numeri interi che usano più di 32 bit conviene usare il delta codes perché

sarebbero +7 bit sulla sx + 16 sulla destra = 33 che è maggiore di 32 bit dell'intero.

x	$\gamma(x)$	$\delta(x)$
1	0.	0.
2	10.0	100.0
3	10.1	100.1
4	110.00	101.00
5	110.01	101.01
6	110.10	101.10
7	110.11	101.11
8	1110.000	11000.000

2) Delta: Scrive $b = |\text{bin}(x)|$ bits usando il gamma code $\gamma(b)$ seguiti da $b-1$ bits meno significativi di $\text{bin}(x)$, perché $V(b)$ può essere molto grande per interi grandi.

Abbiamo che $|\delta(x)| = |\gamma(|\text{bin}(x)|)| + |\text{bin}(x)| + 1$ bits, questo numero di bits è lontano dall'ottimo per un factor of $1 + o(1)$, essendo $\gamma(|\text{bin}(x)|)$ costante (massimo 32 bits).

Esempi:

a) Esempio Gamma, $x=5$:

$$\begin{aligned} \cdot \text{ bin}(5) &= \underbrace{\begin{array}{c} 1 \\ 0 \\ 1 \end{array}}_{|\text{bin}(x)|} &> \gamma(x) = \text{Unary}(|\text{bin}(x)|) \cdot \text{bin}(x) = 110.01 \\ \cdot \text{ unary}(3) &= 110 \end{aligned}$$

b) Esempio Delta, $x=5$:

$$\begin{aligned} \cdot \text{ bin}(5) &= \underbrace{\begin{array}{c} 1 \\ 0 \\ 1 \end{array}}_{|\text{bin}(x)|} &> \delta(x) = \gamma(|\text{bin}(x)|) \cdot \text{bin}(x) = 101.01 \\ \cdot \text{ gamma}(3) &= 101 \end{aligned}$$

Variable - byte codes

(byte-aligned)

Le variable - byte code è un algoritmo che lavora con i byte invece che i bit. Le byte-aligned codewords sono più semplici da implementare e favoriscono la decoding speed (svolgendo op. in parallelo, uso di registri SIMD)

La rappresentazione binaria di un intero x viene suddivisa in un numero di bytes, ognuno di essi composto da 7 bit usati per la rappresentazione di x e 1 bit di controllo che indica la presenza o meno di byte successivi.

Example for $x = 67822$, $\text{bin}(67822, 17) = 10000100011101110$

(1) 100 0010001 1101110
 (2) ~~xxxxx~~100 x0010001 x1101110
 (3) ~~00000100~~ 10010001 11101110
 Control bits

Come determinare se un encoding algorithm è migliore di un altro? \rightarrow Information Content

Dato un messaggio L , vogliamo stimare $P(x)$, cioè la probabilità che x occorra in L . For example, $P(1) = 0.9$, $P(2) = 0.09$, and $P(x > 2) = 1 - 0.9 - 0.09 = 0.01$

Golden rule: Se x è molto frequente in L ($P(x)$ è alto) dovrebbe ricevere una codeword corta.

L'information content $I(x)$ di x è definito come:

$$I(x) = \log_2 \frac{1}{P(x)} \rightarrow \text{è inversamente proporzionale alla frequenza di } x \text{ perché un evento che accade poche volte è più rilevante rispetto a uno frequente.}$$

Misurato in bits

Haggione è $P(x)$, minore è l'information content $I(x)$ di x e viceversa.

Di conseguenza, un codice C è un optimal code se $|C(x)| = I(x) = \log_2 1/P(x)$.

Possiamo invertire la relazione per trovare la distribuzione $P(x)$ per la quale il codice è ottimo:

$$P(x) = 2^{-|C(x)|}$$

Integer list Compression

posso trasformare una lista in strictly increasing sommando (e sottraendo in decodifica) l'indice della lista al corrispettivo elemento

Data una lista ordinata strictly increasing di n interi ($L[1...n]$) presi da un universo di dimensione U (cioè $U > L[n]$), vogliamo comprimerla nel minor numero di bits.

Vogliamo permettere due tipologie di operazioni sulla lista encoded senza doverla decodificare ogni volta:

a) next GEQ (x): Successore di x , cioè il più piccolo intero y nella lista t.c. $y \geq x$.

b) access (i): Ritorna l' i -esimo elemento di L per un random $i \in [1, n]$

+) GAPS

Gaps sfrutta la proprietà che i gaps tra gli interi della lista sono piccoli in media.

- $L = [1, 3, 13, 14, 15, 16, 20, 22, 23, 34, 35, 36, 40]$ GAPS
- $L' = [1, 2, 10, 1, 1, 1, 4, 2, 1, 11, 1, 1, 4]$

Possiamo poi usare un qualsiasi static integer compression sui gaps. (Perché la lista non è più ordinata)

Problema: Se vengono compresi solo i gaps, per svolgere le operazioni next GEQ e access dobbiamo decodificare tutta la lista, quindi le due op. hanno costo in tempo $O(n)$.

2) Blocking

Il metodo **blocking** consiste nel dividere la lista in blocchi (sublists) e fare la compressione blocco per blocco indipendentemente.

In questo modo le operazioni next GEQ e access richiedono la decodifica di uno specifico blocco (e anche il successivo se necessario) invece di tutta la lista.

next GEA (x) : Richiede la binary search per individuare il blocco dove x è contenuto e decodificare il blocco. $O(\log(n/B) + B)$

access (i) : Richiede la decodifica del $\lfloor i/B \rfloor$ -esimo blocco. $O(B)$

Esempio:

$$L = [1, 3, 13, 14, 15, 16, 20, 22, 23, 34, 35, 36, 40, 48, 51, 52, 53, 54]$$

upper = [15,34,51,54] (encoding of the last elements of the blocks)

$lower = [1,2,10,1][1,4,2,1][1,1,4,8][1,1]$ (gaps of the integers in the blocks) \Rightarrow L'ultimo elemento non viene salvato perché Salvava nell'upper block
 $widths = [4,3,4,1]$ (all gaps from block i can be represented in $widths[i]$ bits)

Problema: La presenza di un intero grande causa l'aumento del numero di bit per tutti gli interi all'interno del blocco.

3) P For Delta

L'idea di PFonDelta è di ignorare initialmente gli interi grandi (mettendoli da parte), per poi gestirli successivamente.

Viene scelta una base b (valore più piccolo della lista) e k (numero di bit con il quale posso rappresentare più del 90% dei valori nella lista).

a) Agli interi che non sono "eccezioni", cioè gli interi della lista che finiscono nel range $[b, b+2^k-1]$, viene sottratto b e rappresentati con k bits.

b) Agli interi che sono "eccezioni" ($> b+2^k$) viene assegnata la codeword 2^k-1 e codificati in una lista separata.

Oss: 2^k viene usato come marker per l'inizio delle eccezioni (intervi problematici).

Example for $L = [3, 4, 7, \underline{21}, 9, 12, 5, 16, 6, 2, 34]$

- We can choose $b = 2$ and $k = 4$, and model L as

[1, 2, 5, *, 7, 10, 3, *, 4, 0, *] - [21, 16, 34]

- The first part will be coded as

0001.0010.0101.
1111
 $\underbrace{\hspace{1cm}}$
 2^4-1
(eccezione)

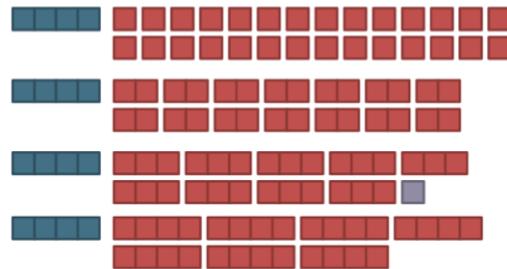
4) Simple 9

Il simple 9 stona diversi interi in un unico intero (32 bit).

Per farne ciò usa 4 bit per "l'encoding block" e i restanti per gli interi.

Con 28 bit ho 9 modi per salvare degli interi:

- 0: a single 28-bit number
- 1: two 14-bit numbers
- 2: three 9-bit numbers (and one spare bit)
- 3: four 7-bit numbers
- 4: five 5-bit numbers (and three spare bits)
- 5: seven 4-bit numbers
- 6: nine 3-bit numbers (and one spare bit)
- 7: fourteen two-bit numbers
- 8: twenty-eight one-bit numbers



I 4 bit più significativi (encoding block) servono per capire quale dei 9 modi viene usato.

Gli spare bit sono bits non utilizzati.

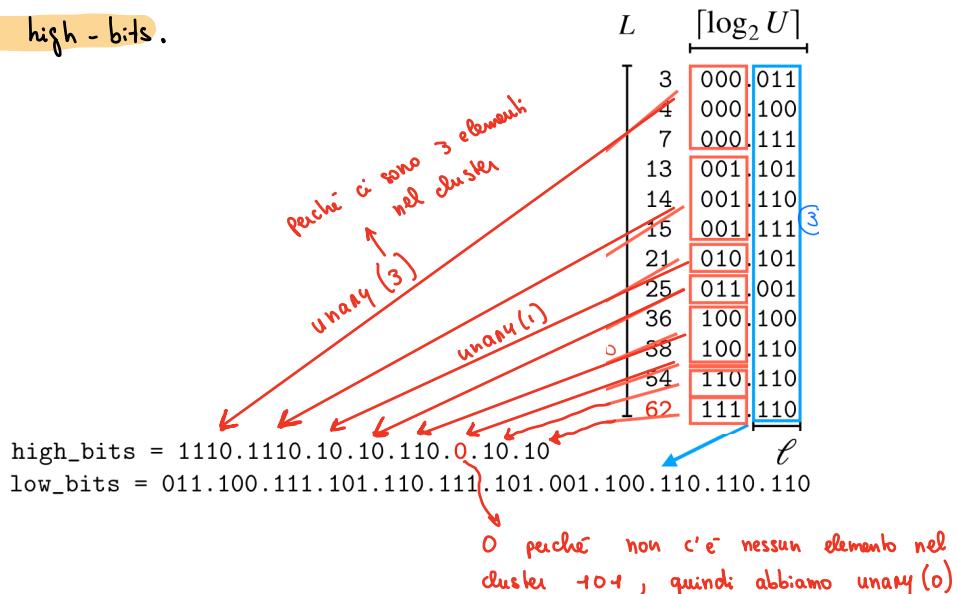
5) Elias - Fano

a) Calcola la rappn. binaria di ogni intero x della lista usando $\lceil \log_2 U \rceil$ bits, dove U è l'elemento più grande della lista (upper bound)

b) Dividi ogni rappn. binaria in due parti:

- $\lceil \log_2 (U/n) \rceil$ bits più significativi = low bits
- $\lceil \log_2 U \rceil - \lceil \log_2 (U/n) \rceil$ meno significativi = high bits

c) I low bits vengono scelti in un vettore di $n\ell$ bits chiamato low-bits.
Gli high bits vengono raggruppati e scelti in unario in un altro vettore, chiamato high-bits.



Osservazioni:

- Per il vettore low-bits abbiamo bisogno di $n \cdot \ell = n \lceil \log_2 (U/n) \rceil$ bits
- Per il vettore high-bits abbiamo $2n$ bits al più
- Lo spazio usato è al più $n \lceil \log_2 (U/n) \rceil + 2n$ bits il quale è approssimativamente 0.56 bit lontano dall'ottimo (lower bound della Shitzing approx)

$$n \log_2 \left(\frac{U}{n} \right) + 1.44n \text{ bits}$$

Attenzione:

- In tutti gli integer compression algo abbiamo ignorato le info sulla posizione dei termini (le quali sono molti interi). Gli schemi che si usano sono

sostanzialmente gli stessi.

- 2) Non abbiamo un compressione algo migliore di un altro, perché coloro che salvano più spazio sono più lenti a decomprimere, e viceversa (tradeoff).

Basic Bayes Rule: for events A and B such that $P(B) > 0$, we have:

$$\text{Posterior probability (of } A \text{)} \longrightarrow P(A | B) = \frac{P(B | A)P(A)}{P(B)} \longleftarrow \text{Prior probability (of } A \text{)}$$

Probabilistic Information Retrieval

Abbiamo il bisogno di introdurre la probabilità nell'IR perché abbiamo dei fattori non deterministici, come la user need o la rilevanza di un doc per una query. La probabilità ci permette di modellare l'incertezza, i.e. non determinismo.

Vogliamo quindi rankare i documents in una collection con la probability of relevance del documento, data una query.

$$P(\text{relevant} | \text{document, query})$$

Cioè, data una query, calcoliamo P per ogni documento della collezione e gli ordiniamo in modo decrescente.

Probability Ranking Principle

Il PRP funziona sotto certe assunzioni:

- 1) Un solo information need encoded come textual query
(un solo utente)
- 2) L'utente esprieme un relevance judgement per ogni documento della collezione, rispetto alla query.

↳ è un'assunzione molto forte perché stiamo assumendo che un utente possa dare un relevance judgement

PER OGNI documento della collezione, cioè senza tenere in considerazione, ad esempio, documenti scritti in linguaggi diversi o altro.

- 3) Un relevance judgement è una proprietà del documento che dipende solo dall'information need.
↳ Assunzione sull'indipendenza dei documenti e indipendenza dagli altri information need.

4) Un relevance judgement è binario (relevant / not-relevant)

Sotto queste assunzioni, il PRP attesta:

Se le risposte di un IR system per ogni query sono un ranking dei documenti nella collezione in ordine decrescente basato sulla probabilità di utilità per l'utente che ha fatto la richiesta, dove le probabilità sono stimate il più accuratamente possibile sulla base dei dati messi a disposizione al sistema, allora la effectiveness del sistema per l'utente sarà la migliore ottenibile su quei dati.

↓ Detto con la matematica

- L' utilità è misurata come la rilevanza di un documento a una query
 - La rilevanza (R) è una random binary variable con valori $R=1$ e $\bar{R}=0$.
 - La query (Q) è una variabile random con un certo valore q
 - Un documento (D) è una variabile random con un certo valore d

La probabilità che la coppia sia rilevante (query q e il documento d) è data dalla seguente formula:

$$P(R = r | Q = q, D = d) = P(r | q, d)$$

- **YR** sistema ritorna K documents (data una query) : d_1, \dots, d_K
 - **YR Recall** (R_K) è una random variable che denota il numero di documenti relevanti retrieved (per una given query) : $R_K \in \{0, 1, \dots, K\}$
 - l' effectiveness è misurata come l' expected number of relevant documents retrieved, cioè $E[R_K]$. \rightarrow Expected Recall

TEOREMA: Sotto le assunzioni, il PRP è valido. \rightarrow Ottieniamo il recall massimo possibile.

Dimostakione:

Dimostrazione:

$$1) \text{ Per ogni } k \text{ abbiamo: } E[R_k] = \sum_{r=1}^k P(r|d_i, q) + \bar{r} \sum_{\bar{r}=1}^k P(\bar{r}|d_i, q) = \sum_{r=1}^k P(r|d_i, q).$$

$= 1$ $= 0 \rightarrow$ possiamo rimuovere i nodi bloccati

- 2) Dobbiamo dimostrare che, per ogni k , $E[R_k]$ è massimale quando i documenti sono ordinati secondo la probability of relevance decrescente.
 Lo facciamo per contraddizione.
- 3) Assumiamo, per contraddizione, che esiste un valore \bar{k} t.c. $E[R_{\bar{k}}]$ è massimale quando i documenti non sono ordinati secondo la probability of relevance decrescente, quindi esiste almeno un documento d_{ℓ} con $P(r|d_{\ell}, q) < P(r|d_{\bar{k}}, q)$ t.e.:
- Il sorting decrescente dei docs è il seguente: $d_1, \dots, d_{\bar{k}}$
 - Il sorting della contraddizione è il seguente: $d_1, \dots, d_{\ell}, \dots, d_{\bar{k}-1}$
- 4) Abbiamo quindi:

$$E[R_{\bar{k}}(d_{\bar{k}})] = \sum_{i=1}^{\bar{k}-1} P(r|d_i, q) + P(r|d_{\bar{k}}, q) > \sum_{i=1}^{\bar{k}-1} P(r|d_i, q) + P(r|d_{\ell}, q) = E[R_{\bar{k}}(d_{\ell})].$$

Che contraddice l'assunzione che $E[R_{\bar{k}}(d_{\ell})]$ sia massimale.

Cooper's Example

È un counterexample usato per osservare che se un'assunzione del PRP non è valida, anche il PRP stesso non è valido.

Assumiamo:

- Una query q eseguita da un utente scelto a random da due user groups, U_1 e U_2 , tale che $|U_1| = 2 |U_2|$
- Document collection di 10 docs
- Documenti: d_1, \dots, d_9 rilevanti per gli utenti in U_1 e d_{10} rilevante per gli utenti in U_2 . Cioè: $P(r|d, q) = \begin{cases} \frac{2}{3} & \text{if } d = d_1, \dots, d_9 \\ \frac{1}{3} & \text{if } d = d_{10} \end{cases}$
- Il sistema è ottimale se fa il retrieve di almeno un relevant doc per ogni utente.

Consideriamo ora due ranking:

- 1) È un PRP ranking (d_1, \dots, d_{10}) per il quale un utente da U_1 è sempre soddisfatto dal primo documento, mentre un utente da U_2 deve aspettare sempre il 10° documento.
- 2) Non è un PRP ranking $(d_1, d_{10}, d_2, \dots, d_9)$ per il quale un utente da U_1 è sempre soddisfatto dal primo documento, mentre un utente da U_2 deve aspettare sempre il 2° documento.

Come possiamo osservare il NON-PRP ranking è migliore perché soddisfa entrambi gli utenti con un minor numero di docs da essere "scrollati".

Questo risultato è dovuto al fatto che l'assunzione sul "single user" non è valida.

Probabilistic Relevance Models

La probabilità che un documento d sia rilevante per la query q è data da:

$$P(r|d, q) = \frac{P(d|r, q)P(r)}{P(d)}$$

dove:

$P(r)$ = Prob di restituire un doc rilevante random

$P(d|r, q)$ = Prob. che se un doc rilevante viene restituito, sia d

Viceversa, la probabilità che d non sia rilevante per q : $P(\bar{r}|d, q) = \frac{P(d|\bar{r}, q)P(\bar{r})}{P(d)}$

la somma di queste due probabilità deve essere 1.

In generale, la relevance strategy basata su PRP è la seguente:

- 1) Stimare come ogni termine contribuisce nella rilevanza (assumiamo che i termini che non appaiono nel documento hanno una contribuzione pari a 0)
- 2) Combiniamo le contribuzioni dei termini per trovare la document relevance probability
→ Dobbiamo tenere in considerazione
 - Quante volte un term occorre
 - Lunghezza dei documenti
- 3) Ordiniamo i docs per probabilità decrescente e scegliamo i primi k

Potremo da modelli semplici (che non usano la term occurrence o doc length) ed aumenteremo la complessità man mano.

Binary Independence Model (BiM)

n documenti vengono rappresentati con vettori di n binary values $x = (x_1, \dots, x_n)$, dove n è la lunghezza del vocabolario.

Quindi abbiamo:

- term frequency costante ($= a$) perché binario
- document length costante ($= a n$)

x_i rappresenta se l' i -th termine è presente nel documento.

Data una query:

- 1) Per ogni documento calcoliamo $p(n | d, q)$ che diventa $p(n | x, q)$ perché d è un binary vector x
- 2) Siamo solo interessati all'ordine dei docs (usiamo rank-preserving functions, come la odds che è monotonically increasing, per semplificare il calcolo della probabilità)

Ottieniamo infine che:

$$w_i^{\text{BM1}} = \log \frac{N - n_i + 0.5}{n_i + 0.5} \approx \log \frac{N - n_i}{n_i} \approx \log \frac{N}{n_i} = IDF_i$$

BiM ha delle assunzioni troppo restrittive:

- Relevance Binaria
- Relevance di un documento indipendente dagli altri documenti
- Termini nello stesso documento sono indipendenti
- Termini della query non impattano il document ranking.

Abbiamo bisogno di qualcosa che sia capace di gestire la term occurrence e doc length.

BiM with eltness

N documenti possono essere generati da un generative model dal quale le parole vengono pescate indipendentemente usando una multinomial distribution.

La distribuzione del term frequency seguono una Poisson Distribution, la quale modella la probabilità di un certo insieme di eventi di avvenire dopo un certo numero di esperimenti (**document length fissata**).

1 - Poisson model : è un singolo poisson model ed è utile per modellare le generic words ma non le topic words.

Questo perché lo t-poisson model assume un constant rate delle occurrente tra i documenti.

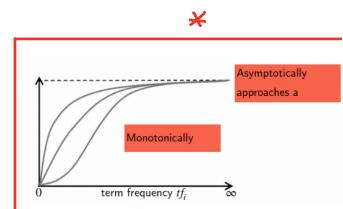
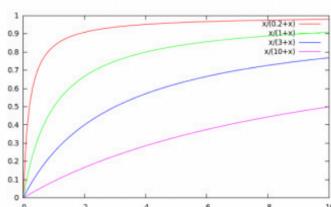
2 - Poisson model : è un mix di due Poisson distribution, ognuna con un differente occurrence rate.

Una viene usata per i general terms e una per gli elite terms.

Stimare i parametri del 2-poisson model non è facile, ma abbiamo visto che l'information content che forniamo ripetendo più volte un elite term è limitato da c_i^{BiM} , quindi, essendo che siamo interessati ad un risultato cometto in media su tutti i termini, approssimiamo con una curva che abbia più o meno la forma cometta.

Questa funzione è chiamata **saturating function** ed è così definita:

$$\frac{tf_i}{k_1 + tf_i}$$



Ponendo quindi da BiM, rappresentando i documenti con le term frequency $x = (tf_1, \dots, tf_n)$

otteniamo: $c_i^{\text{BM15}}(tf_i) \approx \log \frac{N}{n_i} \times \frac{tf_i}{k_1 + tf_i} \rightarrow \text{TFIDF normalizzato}$ (bounded scores) $\downarrow n = |V|$

La poisson distribution assume che i documenti siano della stessa lunghezza, ma nella realtà è un'assunzione che non regge.

La term frequency dovrebbe essere normalizzata secondo la document length.

Siano:

- Document length (dl_j): $dl_j = \sum_{i \in V} tf_i$

- Avg Document length (avdl): $avdl = \frac{1}{N} \sum_{j=1}^N dl_j$

Possiamo rappresentare la term frequency (tf_i) in due modi:

1) BH41

Definiamo tf'_i come un valore normalizzato rispetto all'avg doc length over doc length. $tf'_i(d_j) = tf_i(d_j) \frac{avdl}{dl_j}$

Ottieniamo quindi, dalla BMIS formula, i BMII weights:

$$c_i^{\text{BMII}}(tf_i, d_j) = c_i^{\text{BM1}} \frac{tf'_i(d_j)}{k_1 + tf'_i(d_j)} = c_i^{\text{BM1}} \frac{tf_i(d_j)}{k_1 \frac{dl_j}{avdl} + tf_i(d_j)}$$

2) Definiamo tf'_i come un valore normalizzato rispetto al fattore di normalizzazione B_j (per avere una via di mezzo invece che la full normalization)

$$tf'_i(d_j) = \frac{tf_i(d_j)}{B_j} \quad \text{dove:} \quad B_j = \left((1 - b) + b \frac{dl_j}{avdl} \right) \text{ with } 0 \leq b \leq 1$$

- ↗ BMII
- When $b = 1$, full document length normalisation
 - When $b = 0$, no document length normalisation
- ↘ BMIS

Ottieniamo quindi, dalla BMIS formula, i BM25 weights:

$$c_i^{\text{BM25}}(tf_i, d_j) = c_i^{\text{BM1}} \frac{tf'_i(d_j)}{k_1 + tf'_i(d_j)} = c_i^{\text{BM1}} \frac{tf_i(d_j)}{k_1 B_j + tf_i(d_j)}$$

e quindi il BM25 RSV score è:

$$RSV^{\text{BM25}}(q, d) = \sum_{i \in q} \frac{tf_i(d)}{k_1 \left((1 - b) + b \frac{dl(d)}{avdl} \right) + tf_i(d)} \log \frac{N}{n_i}$$

TF parzialmente normalizzata IDF

dove :

- ↑ Controlla influenza tf
- k_1 settato tra 1.2 e 2 tipicamente
 - b settato a ≈ 0.75 tipicamente
- ↓ Controlla influenza doc length

Ranking with fields

Finora abbiamo assunto che i documenti non avessero struttura (titoli, sezioni ...), queste info (fields) possono però essere sfruttate.

Come fare il ranking with field?

Idea 1)

- Applicare una ranking function (es: BM25) ad ogni field separatamente
- Fare una somma pesata degli score di ogni field, ad esempio:

$$s(q, d) = w_{\text{title}} \cdot s(q, d_{\text{title}}) + w_{\text{body}} \cdot s(q, d_{\text{body}})$$

Problema: Stiamo assumendo che l'elitness property di ogni field è indipendente dagli altri fields. Ottieniamo quindi poco information content perché una parola ^{elite} può comparire una singola volta in quel field (indicando che quel termine non è importante per quel documento).

Idea 2)

Vogliamo che l'elitness sia una proprietà dell'intero documento, non solo del field.

Per fare ciò, invece di aggregare gli score, aggiungo le term frequency per ogni field, andando a dare pesi diversi per ogni field (es: Parola "hell" nel titolo uguale a "hell" nel body)

Considerando il BM25 quindi:

- Per ogni termine calcola la somma pesata delle term frequency di ogni field.
Allo stesso modo possiamo considerare la document length totale come somma pesata della lunghezza dei fields.

- Per l'IDF e i parametri (b e k), solo b è stato mostrato essere while field-specific.

Ottieniamo quindi:

$$RSV^{\text{BM25F}}(q, d) = \sum_{i \in q} \frac{\tilde{tf}_i(d)}{k_1 + \tilde{tf}_i(d)} \log \frac{N}{n_i}$$

dove:

- $\tilde{tf}_i(d_j) = \sum_{f \in F} w_f \frac{tf_i^f(d_j)}{B_j^f}$

- $B_j^f = \left((1 - b^f) + b^f \frac{dl_j^f}{avdl^f} \right)$ with $0 \leq b^f \leq 1$

Language Models

Un language model è un modello che assegna le probabilità a sequenze di parole.

La probabilità di una sequenza di parole è data dalla probabilità concatenata di esse (chain rule):

$$P(w_1, \dots, w_n) = P(w_1)P(w_2 | w_1)P(w_3 | w_1, w_2) \cdots P(w_n | w_1, w_2, \dots, w_{n-1}) \Leftrightarrow P(w_1, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1})$$

Calcolare questa probabilità è impossibile per⁻, non vedremo mai abbastanza dati!

Usiamo una semplificazione che considera solo k words precedenti, ottenendo un k -gram language model:

$$P(w_1, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1}) \approx \prod_{i=1}^n P(w_i | w_{i-k+1}, \dots, w_{i-1})$$

Unigram : $P(w_1, \dots, w_n) \approx \prod_{i=1}^n P(w_i)$

Bigram : $P(w_1, \dots, w_n) \approx \prod_{i=1}^n P(w_i | w_{i-1})$

Language Models in IR

Un language model è una distribuzione di probabilità su sequenze di parole.

- 1) Vediamo ognì documento come un language model che genera la query.
- 2) Data una query, vogliamo calcolare la probabilità che il language model generi la query, cioè $P(d|q)$, e ordinare i docs di conseguenza.

Per calcolare $P(d|q)$ usiamo la bayes rule:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$
 dove $P(q)$ e $P(d)$ sono costanti e dobbiamo solo calcolare $P(q|d)$
costanti per ogni doc

- 3) Per calcolare $P(q|d)$ assumiamo di conoscere il language model (M_d) di d .

Sia t un termine (unico) di q : $P(q|M_d) \approx \prod_{\tau \in q} P(\tau|M_d) = \prod_{\tau \in q} P(t|M_d)^{tf(t,q)}$ dove: $tf(t,q)$ frequency

OSS: Non stiamo considerando l'ordine dei termini perché $(q!)$ è costante $\forall d$ di t in q

- 4) Quello che ci manca da calcolare è $P(t|M_d)$

Calcoliamo $P(t|M_d)$ con una maximum likelihood estimation (caso favorevole / total cases)

$$\hat{P}(t|M_d) = \frac{tf(t,f)}{|d|}$$
 dove:

- $tf(t,f) = \text{n° volte } t \text{ appare nel documento}$
- $|d| = \text{lunghezza in token di } d$

Problema: Se un singolo termine della query non compare in d , la probabilità

$$P(q|M_d) \approx \prod_{\tau \in q} P(\tau|M_d) \text{ sarà } 0.$$

Smoothing: L'idea è di usare il language model generale della collezione per fare smoothing dello specifico language model del documento

Quindi, siano:

- M_c = Language Model della intera collezione
- $cf(t) = \text{n° occorrenze del termine } t \text{ nella collezione}$
- $|c| = \text{n° totale di token nella collezione}$

$$\hat{P}(t|M_C) = \frac{cf(t)}{|C|}$$

Facciamo un mix della collection language Model e il document language model per fare lo smooth:

$$P(t|d) = \alpha P(t|M_d) + (1 - \alpha)P(t|M_C)$$

α grande: valori grandi (conjunctive-like)
 α piccolo: valori piccoli (disjunctive-like)

Tipi di smoothing:

alfa ci permette di avere una combinazione convessa

- Jelinek-Mercer Smoothing: $\alpha = \lambda$
- The parameter α is constant for all documents

$$P(q|d) = \prod_{\tau \in d} \left[\lambda \frac{tf(\tau, d)}{|d|} + (1 - \lambda) \frac{cf(\tau)}{|C|} \right]$$

→ posso usare i log per evitarmi le moltiplicazioni

- Dirichlet Smoothing: $\alpha = \frac{\mu}{|d| + \mu}$
- The parameter α depends on the length of the document

$$P(q|d) = \prod_{\tau \in d} \frac{tf(\tau, d) + \mu \frac{cf(\tau)}{|C|}}{|d| + \mu}$$

language Models vs Probabilistic Models

! [LM non modella la rilevanza, invece BM2S si basa su essa.

BM2S funziona meglio perché più robusto (performance migliori su diverse collezioni) mentre il language model potrebbe non performare in alcune collezioni.

Faster Query Processing

Abbiamo visto come tecnica di query processing il DAAT, il quale non è migliorabile sotto il punto di vista dell'effectiveness, ma lo è sotto il punto di vista dell'efficienza.

Dynamic Pruning

È una strategia con lo scopo di fare lo scoring dei documenti più velocemente, facendo lo scoring a solo un subset di essi (Non scommette le posting lists per intero).

Vogliamo quindi ritornare i k documenti migliori per l'utente scippando o terminando

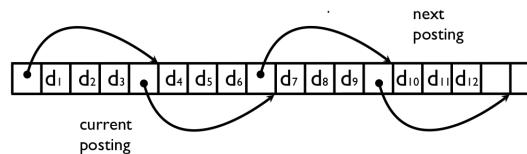
in anticipo lo scoring dei documenti.

Il dynamic pruning si basa quindi su:

1) Skipping

Consiste nell'utilizzo di alcuni skip pointers (che devono essere aggiunti alle posting lists) per poter saltare da un blocco all'altro delle posting lists.

Gli skip pointers sono strutture dati e contengono anche la info sul docID più grande di ogni blocco.



2) Early termination

Consiste nel non fare il fetch o lo score di tutte le posting list di un documento. (non tutti: posting vengono analizzati se non necessario).

Questo è fattibile usando due upper bounds:

a) Term upper bound

Venne calcolato eseguendo, per ogni termine nel vocabolario, una query con quel singolo termine.

Questo valore viene definito da $\sigma_t(q)$ ed è tale che per ogni documento nella posting list del termine t :

$$\sigma_t(q) \geq s_t(q, d)$$

Cioè è il massimo score ottenuto data la query con il singolo termine.

OSS: È un calcolo costoso ma viene fatto solo una volta (offline)

b) Document upper bound

Ad esempio alla fine dell'indexing

Venne calcolato sommando i query term upper bounds e gli actual scores.

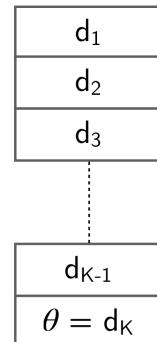
$$\sigma_q(d) = \sum_{t \in q} s_t(q, d) + \sum_{t \in q \setminus \hat{q}} \sigma_t(q, d) \rightarrow \text{Score processato fino a quel momento} + \text{term upper bounds dei termini restanti}$$

3) threshold

Durante il query processing, i top K documents (scores) vengono ordinati in una priority queue (heap data structure).

Il più piccolo valore di questi scores è la soglia θ (threshold)

OSS: Se non ci sono almeno K scores nella queue, il valore della soglia è 0.



Proprietà: Il threshold value non decresce mai.

Dynamic Pruning Condition

Per una query (q) e un documento (d), se il documento ha un upper bound $\sigma_d(q)$ minore o uguale alla soglia (θ), il processing del documento può essere terminato in anticipo.

Dynamic Pruning Strategies \rightarrow Usano le posting list ordinate per docID e lo skipping

1) Max Score

È una strategia basata sull'early termination e sulle posting list essential e non-essential.

- Tipi di posting list:

a) Non-essential postings: La somma dei term upper bound di queste posting list è minore della soglia corrente.

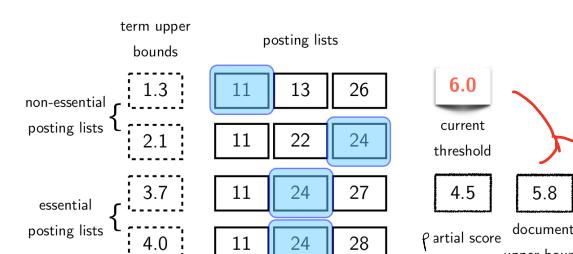
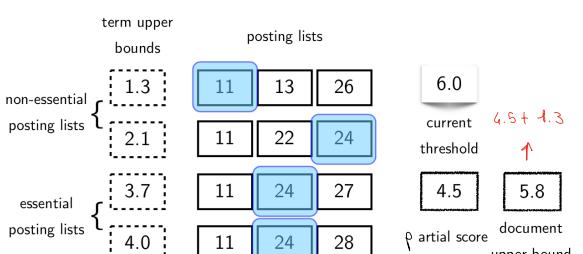
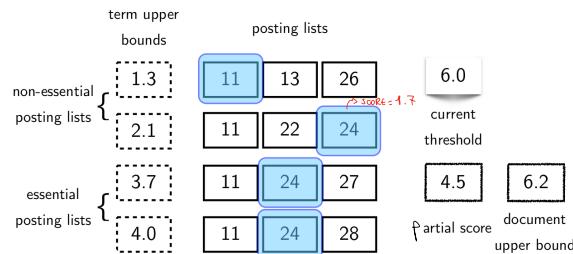
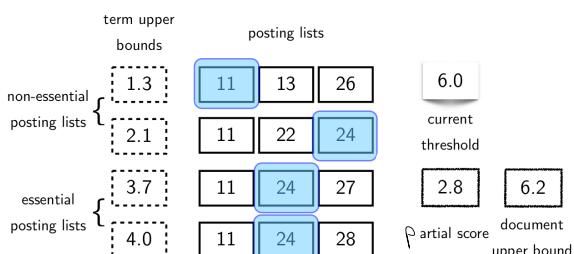
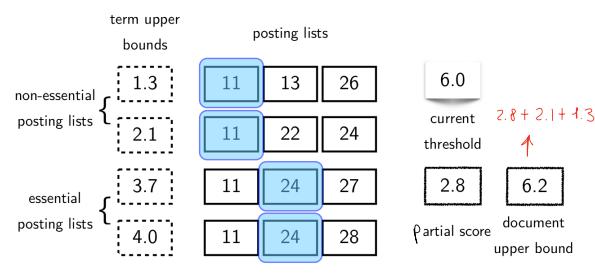
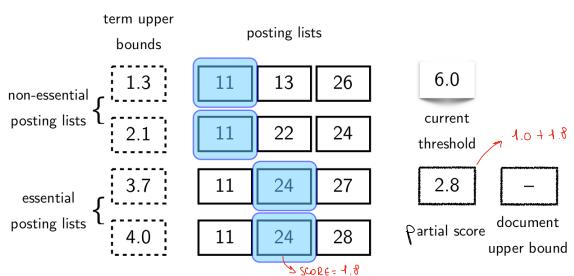
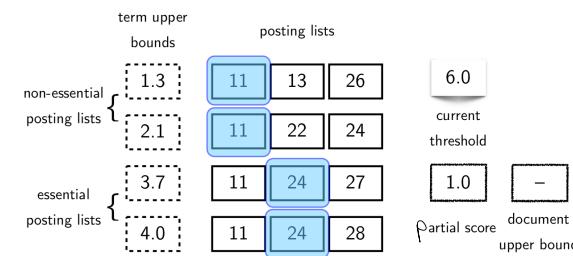
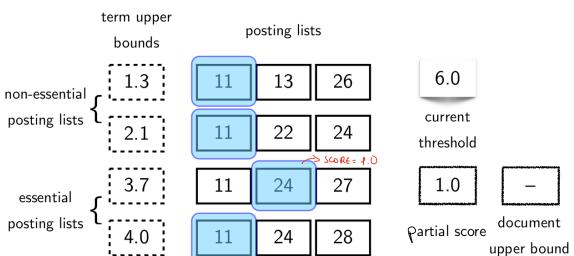
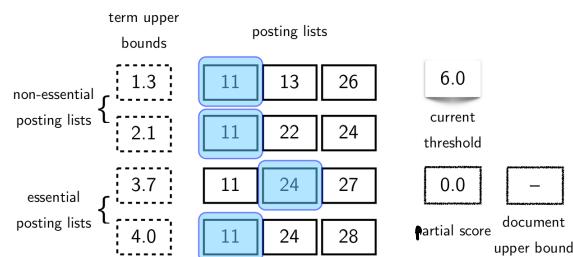
Questo vuol dire che un doc che compare in esse può avere al massimo uno score minore della soglia.

b) Essential posting lists: Viceversa.

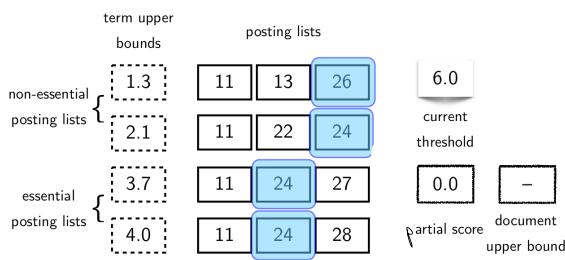
OSS: le essential e non-essential posting lists sono dinamiche, cambiano in base alla soglia.

Yolche, nelle non-essential posting lists viene usato lo skipping, nelle essential no.

- le posting list sono ordinate per term upperbound (ordine statico).



Doc upper bound minore della soglia,



posso fermare qua la computazione per il doc 24 perché non ottenga sicuramente un valore (score) maggiore di 6.

OSS:

- Miglioramento performance maggiore se numero termini nella query maggiore

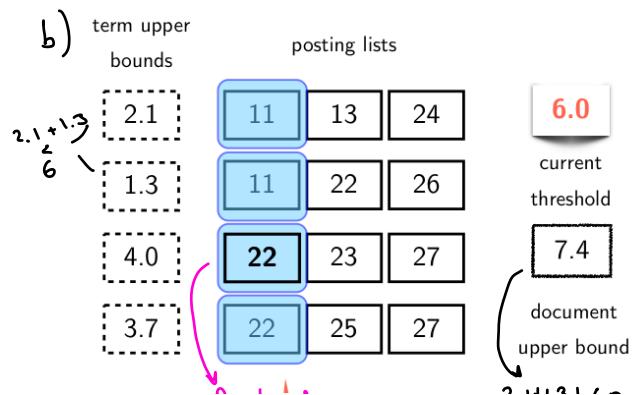
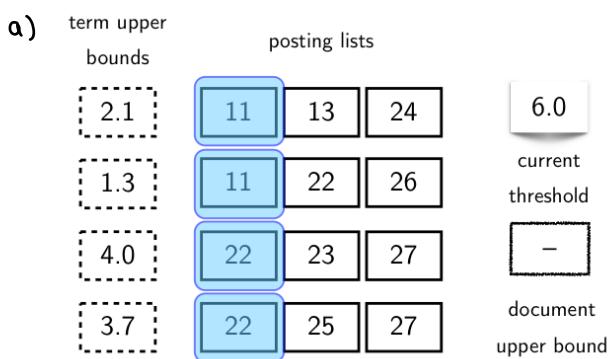
2) WAND

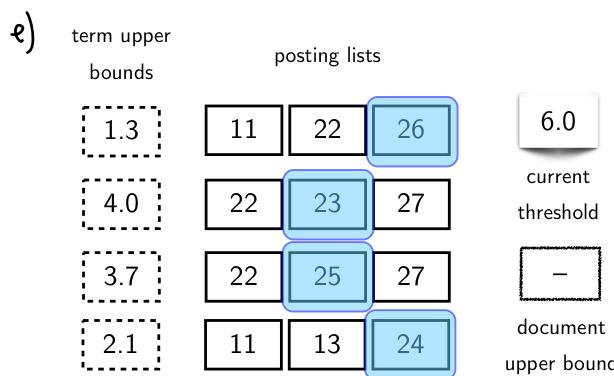
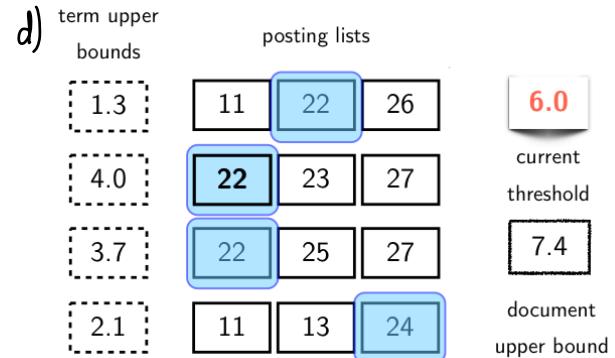
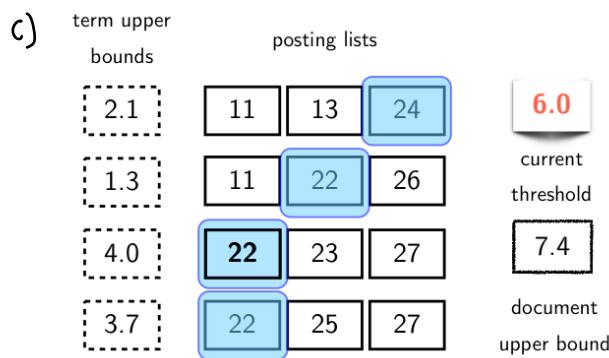
è una **strategy basata sull' approximate evaluation** (cioè che **non considera docs con score approssimati più piccoli della soglia**) e sui pivot term e pivot document.

- le **posting list** sono sempre ordinate per "current docID" (docID puntato dall'iterazione).
- Il **pivot docID** è il primo docID con una non-zero chance di finire nei top K results, cioè la somma dei term upper bounds è maggiore della soglia corrente.

- Ondinare le **posting list** per docID corrente (ordine crescente)
- Trovare il **pivot docID**
- Muovere l'iterazione delle **posting list** fino al **pivot docID** (o subito dopo) con **next GEA (d)**
- Ondinare le **posting list** per docID corrente (ordine crescente)
- Calcolare lo **score** relativo al **docID corrente** e muovere gli iteratori di 1. Ripetere dal punto a)

OSS: l'efficienza di **WAND** si basa sull'efficienza del sorting algorithm per le **posting lists**.





static pruning

Lo static pruning consiste nel rimuovere i dati meno importanti mantenendo (spesso) la qualità dei top K results.

I dati meno importanti sono dei docs che non vengono mai restituiti e termini che non danno un aiuto nel retrieval.

1) Document based pruning: Scarta tutti i termini da un documento che sono meno rappresentativi per il contenuto del doc.

Oss: Può essere fatto durante l'indexing se abbiamo una reasonable collection statistics.

2) Term-based pruning: Scarta i termini che hanno meno possibilità di influire sulla retrieval performance di uno specifico weighting model.

Oss: Viene fatto dopo che l'indexing è completato, in modo offline.

Caratteristiche:

- Static Pruning in se per sé è una lossy compression dell' inverted index.
Pero può essere applicato in una multi-tier architecture, cioè mantenendo sia l'index iniziale che quello con static pruning, e facendo gestire la 1° pagina di risultati all'index con static pruning e le pagine successive, con più tempo disponibile, all'index iniziale.
L'architettura multi-tier ci permette di non perdere info.
- L'index, dopo lo static pruning (fatto offline) è più piccolo e veloce.

Caching

Il caching è un'ottima strategia per risparmiare tempo nella risposta, while soprattutto per le query più popolari.

- 1) Search Result Caching: Salva la lista finale (rankata) dei docs per una specifica query.
- 2) Term Caching: Salva le posting lists di ogni query term in memoria in uncompressed form.

Relevance Feedback

Un **information need** è la causa della query che una persona submette al search engine ed è caratterizzato da:

es: web search $\rightarrow K=10/20$

- **Numero di docs rilevanti richiesti** (K) $\xrightarrow{\text{es: Sini}} K=1$
- **Tipo dell'informazione** (non solo docs ma anche hypopages tipo i numeri ad esempio)
- **Tipo di task**

Una query può rappresentare diversi information need e può essere una "poor representation" di essi (info need difficile da esprimere, query ambiguity dovuto al fatto che una stringa può rappresentare diversi info needs (es: python, apple...))

Il **query formulation problem** consiste nella difficoltà di generare well-formulated queries senza:

- **Conoscere la collectione**, cioè quali termini vengono usati e sono inclusi nell'index.
- **Conoscere il retrieval environment**, cioè le tecniche usate dall'IR.

La **prima query** è generalmente una "trial run" usata per estrarre info e generare le successive query conseguentemente \rightarrow **Iterative Process**

Explicit Interaction

L' **interaction** (fare query, browsing, fare query,...) tra l'utente e il sistema può essere **esplicita**, cioè il sistema aiuta l'utente nel perfezionare la query.

Abbiamo visto 3 tipi di explicit interaction:

- 1) **Query suggestion**: il search engine suggerisce query relative alla query inserita dall'utente ("Forse intendevi questo?")
- 2) **Query expansion**: il search engine suggerisce query terms aggiuntivi all'utente (l'autocompletamento suggerito)
Viene fatto usando i query logs.

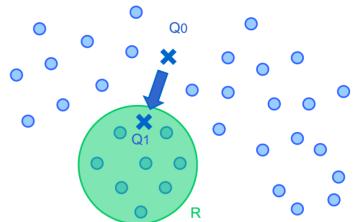
3) **Relevance Feedback**: le search engine permette all'utente di dire quali docs restituiti sono rilevanti o meno, e a formula la query sulla base di queste info.

a) Vengono restituiti all'utente dei risultati, permettendogli di fornire un feedback sulla relevance di uno o più docs.

b) L'IR system usa il feedback per a formulare la query, andando ad analizzare i termini più freq. nei docs segnati come rilevanti e aggiungendoli alla query con un peso basato sulla term frequency

c) Presentare i nuovi doc all'utente sulla base della nuova query e ripetere finché l'utente non è soddisfatto.

Rappresentando i docs e le query in un vector space, intuitivamente dobbiamo muovere la query verso un documento rilevante aggiungendo o rimuovendo termini ad essa o modificando i pesi dei termini già presenti.



→ **Clustering hypothesis**: Docs rilevanti sono vicini nello spazio

Intuitivamente vogliamo:

- **Dare** un **peso positivo** a termini che sono in docs **rilevanti** → **AVVICINARE**
- **Dare** un **peso negativo** a termini che sono in docs **non-rilevanti** → **ALLONTANARE**
- **Rimuovere** i termini che appaiono solo in doc **NON-RILEVANTI**

Scritto formalmente, la query ottimale massimizza la differenza fra l'avg

Top 10 documents for "tropical fish"

1. [Badmans Tropical Fish](#)
A freshwater aquarium page covering all aspects of the tropical fish hobby ... to Badmans Tropical Fish ... world of aquariony with Badmans Tropical Fish ...
2. [Tropical Fish](#)
Notes on exotic species and a gallery of photos of African cichlids
3. [The Tropical Tank Homepage - Tropical Fish and Aquariums](#)
Info on tropical fish and tropical aquaria, large fish species index with ... Here you will find lots of information on Tropical Fish and Aquariums ...
4. [Tropical Fish Centre](#)
Offers a range of aquarium products, advice on choosing species, feeding, and health care, and a discussion board
5. [Tropical Fish Information, the free encyclopedia!](#)
Tropical fish are popular aquarium fish ... due to their often bright coloration ... Practical Fishkeeping - Tropical Fish Hobbyist • Vol. Aquarium related companies ...
6. [Tropical Fish Find](#)
Home page for Tropical Fish Internet Directory ... stores, forums, clubs, fish facts, tropical fish compatibility and aquarium ...
7. [Breeding Tropical Fish](#)
Information on keeping and/or breeding Tropical, Marine, Pond and Coldwater fish ... Breeding Tropical Fish - breeding tropical, marine, coldwater & pond fish
8. [FishInfo](#)
Includes tropical freshwater aquarium how-to guides, FAQs, fish profiles, articles, and forums
9. [Cathy's Tropical Fish Keeping](#)
Information on setting up and maintaining a successful freshwater aquarium
10. [Tropical Fish Place](#)
Tropical Fish information for your freshwater fish tank ... great amount of information about a great hobby, a freshwater tropical fish tank ...

vector (centroid) che rappresenta i relevant docs e l'avg vector (centroid) che rappresenta i non-relevant docs:

$$\vec{q}' = \alpha \vec{q} + \beta \frac{1}{|D^+|} \sum_{d \in D^+} \vec{d} - \gamma \frac{1}{|D^-|} \sum_{d \in D^-} \vec{d}$$

dove : D^+ : Relevant docs per q
 D^- : Non-Relevant docs per q

OSS :

- γ query terms con pesi negativi vengono doppati
- Per rendere l'espansione della query, solo un certo numero di termini, con il più grande peso nei relevant docs, viene aggiunto.
- Relevance feedback migliora generalmente l'effectiveness a discapito del tempo (molte iterazioni, molti query terms...)
- Migliora principalmente il RECALL!
- È conosciuto anche come Rocchio Algorithm

α, β, γ hyperparametri

Pesi della query, doc rilevanti e non rilevanti

Non sempre:

Se la query è nel mezzo

tra due docs rilevanti o i

parametri non sono adatti per q.

Implicit Feedback

Tramite mouse/click tracking e user behaviour

No user Feedback

Usa i relevance feedback methods senza explicit user input.

- Ritorna i docs e assume che i top m siano rilevanti
- Usa gli m docs per riformulare la query, cioè aggiunge termini più rilevanti di quei docs alla query.

! Funziona perché supponiamo che i primi 10 docs fossero quello che cercavamo e costuiamo un language model su quella collezione (unigram collection model), calcoliamo le freq. e aggiungiamo i termini frequenti che non sono stopwords con l'assunzione è che non avremo mai negative information.

OSS: Generalmente non cambiamo i pesi ai termini della query.

Problema: Query drift, se nessuno (o pochi) dei docs restituiti sono rilevanti effettivamente, riformuliamo la query con termini "inutili" che allontanano la query da docs veramente rilevanti.

Nonostante ciò, può migliorare l'effectiveness del sistema (penalizzando naturalmente l'efficienza).

Query Expansion

Per fare query expansion possono essere usate diverse fonti:

- 1) Top Ranked docs: Basato sulla term co-occurrence con i query terms
- 2) Siti come wikipedia, usando a similar term co-occurrence analysis
- 3) Usando i query logs
- 4) Dizionario dei sinonimi

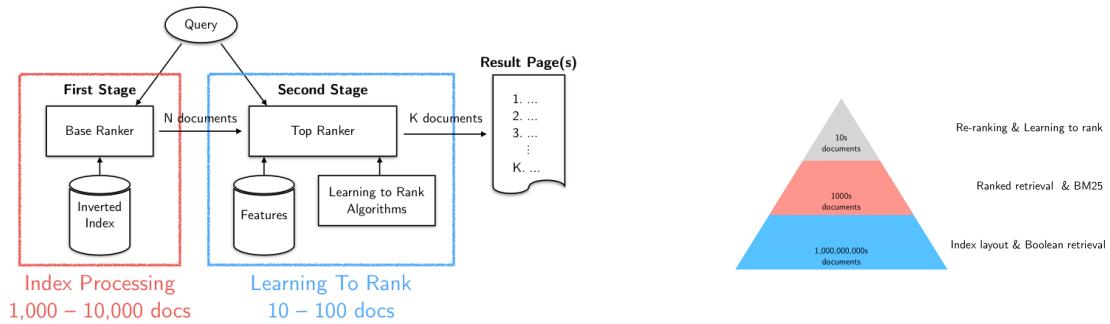
Per ogni termine t della query, espanderla con i sinonimi e related words dal dizionario, dando un peso minore ai termini aggiunti.

- oss:
- Aumenta generalmente il recall \rightarrow Vantaggio
 - Può diminuire la precisione dovuto ai termini ambigui (che non hanno solo un significato \rightarrow svantaggio)

Learning To Rank

Con **learning to rank** intendiamo un **machine learning model** il quale vuole mappare le coppie **(query, docs)** in un insieme ordinato di classi (**Ranking problem** o **ordinal regression**). → Non è né un **classification problem**, né un **regression problem**.

Con l'utilizzo del **learning to rank** model, il **ranking process** si svolge nel seguente modo (a cascata):



1) **First stage:** Viene fatta una **scenatura iniziale** tramite le **soring function "classiche"** (ad esempio **BM25**, essendo più veloci dei **ML algo**) per ottenere **N documents** (nell'ordine delle migliaia).

Di questi **N documents** non ci interessa **sono ordinali**, ma che **siano rilevanti** → Si usa quindi la **"Recall"** per misurare l'**effectiveness** di questo **processo** (Non vogliamo che ci restituiscano "spazzatura")

2) **Second stage:** Degli **N docs** restituiti dal primo stage, viene usato il **learning to rank model** per ottenere i **top K** (in ordine) documenti. Per misurare l'**effectiveness** di questo stage vengono usate le misure tipiche (**NDG, MRR ...**) perché il risultato finale del nostro sistema.

Model training

Come tutti i ML models vengono usati il training set, validation set, test set e una loss function (per aggiornare i pesi) per fare l'allenamento.

Training set

Il training set è composto da sample così formati:

- Query
- lista di (Document, relevance assessment) dove i documenti sono rappresentati da un insieme di real-valued features e i labels possono assumere 5 valori, da 0 (irrelevant) a 4 (perfectly relevant).

$$q_a \rightarrow [(d_4, y_{a4}), (d_2, y_{a2}), (d_3, y_{a3}), (d_8, y_{a8}), (d_{41}, y_{a41}), \dots]$$

$$q_b \rightarrow [(d_{99}, y_{b99}), (d_4, y_{b4}), (d_7, y_{b7}), (d_2, y_{b2}), (d_{11}, y_{b11}), \dots]$$

...

le features per descrivere i docs possono essere centinaia, ma possono essere categorizzate come segue:

1) Query document features: sono features che descrivono la relazione tra query e document
es: BM25, LM, TFIDF...

2) Query independent features: sono features che dipendono solo dal documento
es: doc length, pagerank, in-going/out-going links...

3) Query only features: sono features che dipendono solo dalla query
es: Query length, n° parole unique...

DSS: le features non includono come info la presenza di una specifica parola nel documento. Questo perché questa info viene già considerata nel primo stage.

Loss Function

Una loss function viene usata per capire quanto è buono un particolare modello sui training data.

Disegnare una loss function per ranking task non è semplice perché non è né un task di regressione, né di classificazione.

Abbiamo visto tre tipi di loss function families che sono state esplorate per i ranking tasks:

1) Point-wise loss functions

Sono praticamente dei regression approach, le quali valutano la loss function su un singolo documento (indipendentemente dagli altri).

Problemi:

- a) Ogni documento viene considerato indipendentemente, quindi non consideriamo l'ordine tra essi
- b) Queste loss functions ignorano il fatto che alcuni docs sono associati con la stessa query e altri no
- c) Se il numero di docs varia di molto tra varie query, la loss function sarà dominata da query che hanno un numero maggiore di documenti

2) Pair-wise loss functions

Prendono in considerazione due documenti, dove uno di essi è migliore dell'altro. Cerca di minimizzare la loss function basandosi sul ranking entro tra pairs di documenti.

Problemi:

- a) Cercare di minimizzare l'errore tra due docs non è detto approssimi l'ordine corretto

- b) Il numero di document pairs è quadratico nel numero di docs.
 Risulta quindi costoso, anche se non ho il rilevante essendo fatto offline.

3) List-wise loss Function

Considera l'intero ranking dei documenti ed incapsula una IR evaluation measure standard (es: NDCG, MRR...) nella loss function.
 Highion options!

Possible Models

Quali modelli possono essere usati per il learning to rank?

Linear Models

Sono modelli che generano una combinazione lineare dei features values.

$$s(q, d) = \sum_{f \in F} w_f f(d) \rightarrow \text{Somma pesata delle features}$$

Problemi:

- 1) Fanno assunzioni implizite
 - a) Assumono che le stesse features, con gli stessi pesi, siano usate da tutte le query
 - b) Assumono che il modello sia lineare
- 2) le IR measures sono non continue (non differentiabili) \rightarrow Non posso usare il gradient descent!

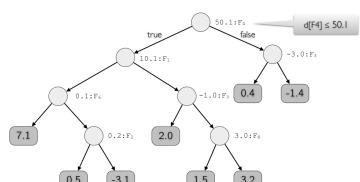
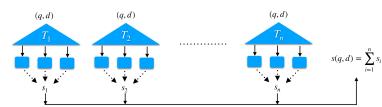
Tree Models

Vengono usate foreste di regression tree models per calcolare lo score finale di un doc rispetto a una query.

Gli score ottenuti dai vari regression tree vengono sommati

Ogni nodo intermedio dell'albero è caratterizzato da una soglia e un feature ID.

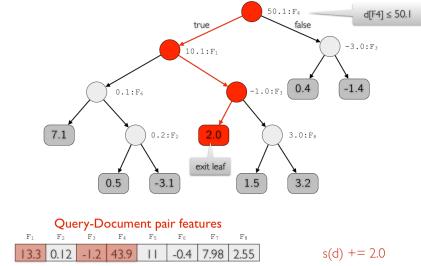
Ogni nodo foglia corrisponde a uno score.



Come vengono usati?

Dato un array di document - Query features F_1, \dots, F_n , partendo dalla radice, l'esplorazione dell'albero avviene in questo modo:

- Vai a sinistra se il valore della feature, corrispondente al featureID del nodo, è minore o uguale alla soglia
- Vai a destra altrimenti



le seguenti caratteristiche impattano sullo scoring Time:

- Numero di alberi: 1000 - 20.000
- Numero di docs: 3000 - 10.000
- Numero di foglie: 4 - 64
- Numero di features: 400 - 1000

1) Struct + - Implementation scoring

Ogni nodo è rappresentato da un oggetto C (struct) contenente:

- Feature ID
- threshold
- Puntatori sx e dx

```
Node* getLeaf(Node* np, float* fv) {
    if (!np->left && !np->right) {
        return np;
    }
    if (fv[np->fid] <= np->threshold) {
        return getLeaf(np->left, fv);
    } else {
        return getLeaf(np->right, fv);
    }
}
```

Vantaggi:

- Riconoscere → Mai lavora!
- No space locality (cache problem): gli elementi dell'albero sono salvati in spazi "random" della memoria
- High branch misprediction rate (essendoci molti if-else, il branch predictor può fare molti sbagli).

2) Code Gen - Implementation scoring

Code Gen consiste nello scrivere staticamente gli if-else block all'interno del programma.

Vantaggi:

- il compilatore può ottimizzare sul codice

feature ID ↗ soglia

```

if (x[4] <= 50.1) {
    // recurses on the left subtree
    ...
} else {
    // recurses on the right subtree
    if(x[3] <= -3.0) {
        result = 0.4;
    } else {
        result = -1.4;
    }
}

```

- Il codice è compatto da enfare nella instruction cache (high cache hit ratio)

Svantaggi:

- Richiede di essere riconciliato se il modello (tree) cambia
- High branch misprediction rate dovuto ai molteplici if-else
- Il compilatore deve attraversare tutti gli if-else del codice (che crescono esponenzialmente con il numero di foglie) per effettuare l'ottimizzazione → la compilazione può richiedere giorni o addirittura non terminare!

3) Vpred - Implementation scoring

Rappresenta i nodi in array (un array per tutti gli alberi).

Per fare ciò utilizza due strutture dati aggiuntive, una usata per localizzare un albero nell'array, una per sapere l'altezza degli alberi.

Nella pratica usa la tecnica del loop unrolling, cioè

scrive le iterazioni esplicitamente (sapendo appieno il numero).

Vantaggi:

- Il compilatore può ottimizzare sul codice
- Il codice può essere parallelizzato (usato su SIMD processor), cioè stessa istruzione su diversi dati in parallelo
- Non abbiamo branch misspredictions (non ci sono if, else, for, while ...)

Svantaggio: Usando l'unrolling ho più istruzioni rispetto ad avere un loop → high compilation times

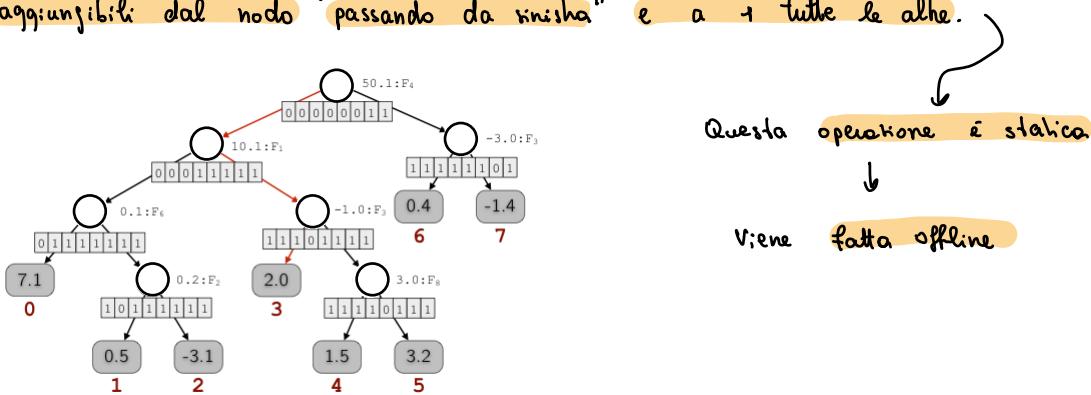
Theory	Practice
<pre>VPRED(x,T): score <- 0 for node <- T do pos <- postree[h] height <- heights[h] while pos < height do pos <- nodes[pos].index[x[nodes[pos].fid] >= nodes[nodeid].theta]; height <- height - 1 score <- score + nodes[pos].theta return score</pre>	<pre>double depth4(float* x, Node* nodes) { int nodeid = 0; nodeid = nodes[0].fid > children[0]; >nodes[nodeid].fid > nodes[nodeid].theta; nodeid = nodes[1].fid > children[1]; >nodes[nodeid].fid > nodes[nodeid].theta; nodeid = nodes[2].fid > children[2]; >nodes[nodeid].fid > nodes[nodeid].theta; nodeid = nodes[3].fid > children[3]; >nodes[nodeid].fid > nodes[nodeid].theta; return score[nodeid].theta; }</pre>

4) Quicksconen - Implementation scoring

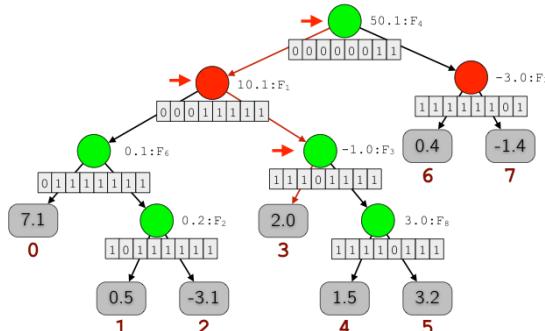
Quicksconen permette di processare più di un albero per volta facendo l'encoding degli alberi come bitvectors e attraversandoli usando logical bit-wise operation (and) che sono commutative.

Esempio su un singolo albero (Intuizione)

- 1) Assegnare un incremental ID ad ogni foglia di un albero
- 2) Sulla base degli ID, calcolare una bitmask (di m bit quante sono le foglie) per ogni nodo intermedio dell'albero, settando a 0 la posizione delle foglie raggiungibili dal nodo "passando da sinistra" e a + tutte le altre.



- 3) Dato un vettore di query-doc features, coloro di verde i nodi dove "vado a sinistra" (cioè dove il valore della features è minore o uguale della soglia) e di rosso gli altri

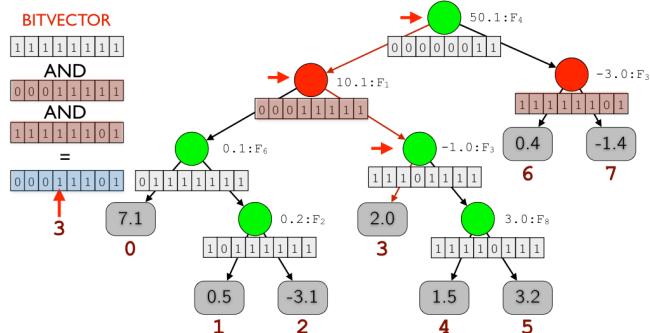


F ₁	F ₂	F ₃	F ₄	F ₅	F ₆	F ₇	F ₈
13.3	0.12	-1.2	43.9	11	-0.4	7.98	2.55

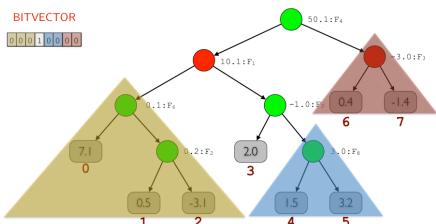
4) Per ottenere la exit leaf devo fare una and operation tra una bitmask

iniziale (settata tutta a 1) e le bitmask dei "nodi rossi".

Nella bitmask risultante, la posizione del bit a 1 più a sinistra corrisponde alla exit leaf che mi dà lo score per quel determinato albero.



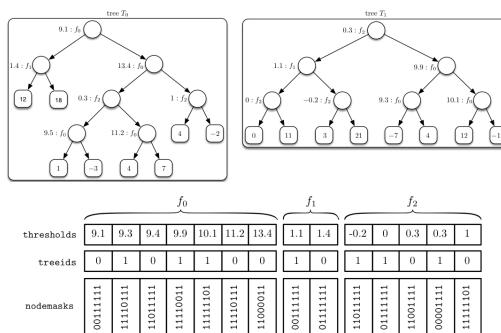
OSS: Quando attraverso i nodi falso rimuovo le foglie (settate a 0 nel bitvector) da sinistra verso destra, quando attraverso nodi veri succede il contrario.



Processare più alberi in parallelo (Algoritmo vero e proprio)

- 1) Calcolo le node mask per ogni nodo degli alberi, le raggruppo per feature ID e le ordino per threshold (TUTTO FATTO OFFLINE)

Ps: uso anche un treeID per riconoscere a quale albero corrisponde la bitmask.



2) Dato un vettore di query-doc features, prendo una feature per volta e verifico quali nodi sono verdi e quali rossi.

- Nodi con soglie miniori del valore → Rosse
- Nodi con soglie maggiori del valore → Verdi

3) Colonati tutti i nodi, per ogni albero prendo i corrispondenti nodi rossi e faccio l' AND operation tra una bitmask tutta a 1 e le bitmask dei nodi.

la posizione del bit settato a 1 più a sx della bitmask risultante corrisponde all' exit leaf (e al corrispondente score) per quel dato albero.

4) Faccio la somma degli score

Vantaggi:

- Spatial locality dovuto all' accesso sequenziale da sx a dx delle bitmask
- Pochi branch mismatch

Quickscore Algorithm



```

Input :
— x: input feature vector
— T: ensemble of binary decision trees, with
    — w: weights, one per tree
    — thresholds: sorted sublists of thresholds, one sublist per feature
    — treeids: tree's ids, one per node/threshold
    — nodemasks: node bitmasks, one per node/threshold
    — offsets: offsets for the bitmasks
    — leafindexes: result bitvectors, one per each tree leaf
    — leafvalues: score contributions, one per each tree leaf
Output:
— Final score of x
Quickscore(x,T):
1   foreach h ∈ 0,1,...,|T| − 1 do // Step ①
2     | leafindexes[0] ← 11...11
3     | foreach k ∈ 0,1,...,|F| − 1 do
4       |   i ← offsets[k]
5       |   offset ← offsets[k + 1]
6       |   while i < offset do
7         |     h ← treeids[i]
8         |     leafindexes[h] ← leafindexes[h] ∧ nodemasks[i]
9         |     i ← i + 1
10        |   if i == offset then
11          |     break
12
13   score ← 0 // Step ②
14   foreach h ∈ 0,1,...,|T| − 1 do
15     |   j ← index of leftmost bit set to 1 of leafindexes[h]
16     |   l ← h · |L_h| + j
17     |   score ← score + w_h · leafvalues[l]
return score

```

Web Search

Nel web search ci sono tre classi principali per le query:

1) Navigational queries: Sono query usate per cercare un risultato specifico.
 ↳ Esempio: "google", "amazon", ...

2) Informational queries: Sono query usate per ottenere informazioni su qualcosa (+ o più risultati)
 ↳ Esempio: "come cucinare la carbonara"

3) Transactional queries: Sono query usate da parte dell'utente per raggiungere un sito dove vuole svolgere uno specifico task
 ↳ Esempio: "dove comprare un ticket per il concerto dei PTN?"

Ranking in web search

Data una query e degli user context data (temporal moment, location, user history...), vogliamo rankare le webpages in modo che quelle rilevanti siano prima delle non relevant.

Quindi il nostro obiettivo diventa stimare $P(n | d, x)$

↓ ↓ ↓
relevance doc query + context

Come abbiamo visto nell' IR, questa probabilità è direttamente proporzionale a $s_M(f)$,

- dove:
- f = insieme di features che dipendono dal documento o contesto
 - M = model parameters (learning to rank model) → Perché abbiamo molte features
↓
BIR5 ne usa solo tre: tf, idf e document length
 - s = scoring function

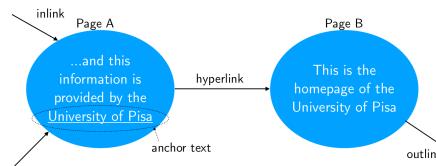
Link Analysis Ranking

Nel caso dei web search, per fare il ranking delle web pages viene usata una combinazione di:

- Content-based evidence: Modelli visti finora, es BM25.
- Authoritativeness: Page rank (instead of link-based ranking)

Link-based ranking

Siamo i link così definiti:



- Un hyperlink indica "autorità" data dalla pagina A alla pagina B (recommendation)
- L'anchor di un hyperlink descrive la "target page" (in questo caso B)

Il numero di in-links di una pagina indicano la sua popolarità/autorità! → SCORE

Problema: È "susceptible" ai link spam (pagine composte da solo un contenuto, cioè un link verso una pagina x)

Page Rank

Page Rank è meno suscettibile ai link spam e consiste in:

- 1) Simulare una random walk di un utente, cioè iniziare da una pagina random e ad ogni step andare in una pagina linkata da essa scelta randomicamente (equiprobabile)

- 2) Osservare quanto spesso le pagine vengono visitate
- 3) L'autoritativess di una pagina è indicata dalla sua popolarità nella simulazione

Markov Chains

Una markov chain consiste in n° states ed una matrice M $n \times n$ (transition probability matrix).

Ad ogni step, siamo in uno di questi states, ad esempio "i", e il numero M_{ij} ci dice la probabilità che j sia il prossimo stato.

Un markov chain è ergodic se non ci sono periodic visit patterns (cicli)

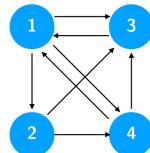


Per una qualsiasi markov chain ergodic, esiste un unico "long-term visit rate" per ogni stato, cioè partendo da un nodo a caso e fermandomi a caso, ho una distribuzione di prob. costante di trovarmi su un certo nodo (non dipende dalla partenza o numero di passi)

Page Rank Computation

Sia il web semplificato composto da 4 webpages connesse così: \Rightarrow

$$\text{Adjacency matrix} \Rightarrow A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \begin{aligned} A_{ij} = 1 &\text{ se c'è un outgoing link da } i \text{ a } j \\ A_{ij} = 0 &\text{ altrimenti} \end{aligned}$$



$$\text{Transition matrix} \Rightarrow T = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/2 & 1/2 \\ 1 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \end{bmatrix} \quad \text{Divido gli } \frac{1}{\cdot} \text{ nelle righe di } A \text{ per il numero di } \cdot \text{ nella riga}$$

$$\text{Markov chain matrix} \Rightarrow M = T^T = \begin{bmatrix} 0 & 0 & 1 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \quad \begin{aligned} M_{ij} = 0 &\Rightarrow \text{no link tra } i \text{ e } j \\ M_{ij} = \frac{1}{n_j} &\Rightarrow n_j \text{ uguale al numero di outgoing link della pagina } j \end{aligned}$$

$$x_1 = x_1 + \frac{1}{2}x_4$$

$$x_2 = \frac{1}{3}x_1$$

$$Mx = x \Rightarrow x_3 = \frac{1}{3}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_4$$

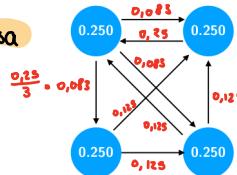
$$x_4 = \frac{1}{3}x_1 + \frac{1}{2}x_2$$

Vogliamo risolvere la seguente eq. la quale

ha un'unica soluzione se M è ergodic

$k=0$) Stato iniziale, tutte le pagine hanno la stessa

authoritativeness ($1/n$ pages)



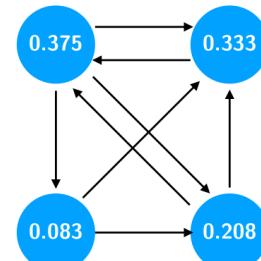
$k=1$) Ogni nodo riceve dell'authoritativeness dagli incoming links, li somma e questa è la nuova authoritativeness del nodo.

Il nodo in basso a sx ha minore authoritativeness perché

ha il minor numero di incoming nodes

Ripetiamo all'infinito ...

$k=\infty$) Final authoritativeness sconosciute, chiamati page ranks

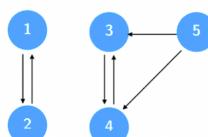


Problema: Essendoci molti nodi (webpages) nel web, è impossibile avere una ergodica matrice, dobbiamo inventarci qualcosa.

Permettiamo la Teleportation.

La Teleportation indica il fatto che in ogni nodo del grafico c'è una probabilità (non-zero) di teleportarsi random invece che seguire un outgoing link, descritta dal parametro lambda.

Sia il grafico e la Markov matrix così composte:



$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1/2 \\ 0 & 0 & 1 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

λ ci assicura che la matrice sia ergodica perché prevede che l'algoritmo sia stucked in un ciclo. (assurda convergenza)

soltanente $\lambda = 0.45$

Ottieniamo la google matrix facendo questa combinazione lineare rispetto al peso lambda

$$G = (1 - \lambda) M + \lambda S \text{ where } S \in \mathbb{R}^{n \times n} \text{ and } S_{ij} = 1/n$$

$$G = (1 - \lambda)M + \lambda S$$

Random Jump

La Markov chain rappresentata dalla Google Matrix è ergodica e quindi posso calcolare il PageRank score.

λ alto → convergenza lenta → G muoviamo random più spesso

Hits (Hyperlink-induced Topic Search)

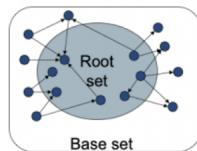
È un ranking algorithm nel quale la link analysis viene fatta su un query-induced graph.

In risposta a una query non viene restituita una ranked list of webpages, ma due inter-related scores:

- 1) Hub scores: Allo per pagine che forniscono tanti link utili per il contenuto delle pagine (topic authorities)
- 2) Authority scores: Allo per pagine che hanno molti in-links da hubs buone per il subject

Come funziona l'algoritmo

- 1) Esegui una query su un IR system per ottenere il root set (top K pages)
- 2) Aggiungi al root set, per ottenere il base set:
 - Pagine con outgoing link a pagine del root set
 - Pagine con incoming links da pagine del root set
 - Link che collegano le varie pagine (ottenuti da un connectivity server)
- 3) Per ogni pagina nel base set calcoliamo hub score e authority score tale che la somma degli hub score sia 1 e la somma degli authority score sia 1.



Per fare ciò usiamo una procedura iterativa:

- a) Inizializziamo, per ogni nodo del base set, $a(x)$ e $h(x)$ uguali a 1
- b) Ad ogni iterazione (K iterazioni):
 - Per ogni nodo x , $h(x) = \sum_{y \rightarrow x} a(y)$, cioè aggiorno hub score con la somma degli authority score dei nodi "puntati" da x
 - Per ogni nodo x , $a(x) = \sum_{y \rightarrow x} h(y)$, cioè aggiorno authority score con la somma degli hub scores dei nodi "che puntano x "
 - Per ogni nodo x , $a(x) = \frac{a(x)}{C}$ dove $\sum_{x \in S} \left(\frac{a(x)}{C}\right)^2 = 1$, cioè li normalizzo $a(x)$
 - Per ogni nodo x , $h(x) = \frac{h(x)}{C}$ dove $\sum_{x \in S} \left(\frac{h(x)}{C}\right)^2 = 1$, cioè li normalizzo $h(x)$
- c) Dopo poche iterazioni ($K=3-5$), gli score convergeranno

Prova di convergenza dell'algo:

- 1) Sia A la adj matrix del base set S
- 2) • Hub scores aggiornati diventano $h = A \cdot a$
• Auth scores aggiornati diventano $a = A^T \cdot h$
- 3) Sostituendo:
 - $h = A A^T h$ (cioè h è un autovettore di $A \cdot A^T$)
 - $a = A^T A \cdot a$ (cioè a è un autovettore di $A^T \cdot A$)

Entrambi a e h possono essere calcolati usando il power iteration method che è un algo conosciuto per calcolare gli autovettori ed è garantita la sua convergenza.

web search Architecture

Come costruiamo l'inverted index? → Usiamo un web crawler

Web Crawling

Il web crawling è il processo di localizzare, recuperare, stoccare e mantenere le pagine disponibili nel web. Questo processo viene svolto da programmi chiamati "crawlers", "spiders", "harvesters".

Un'assunzione importante per i web crawler è che possiamo raggiungere ogni web page da un certo numero di entry points (no cluster indipendenti) perché essi usano la shuttling ad hyperlinks.

Le informazioni "eshalte" dal web crawler vengono salvate nella web crawler repository.

Basic Web Crawling Process:

- 1) Initializzare una URL download queue (chiamata URL frontier) con alcuni URL iniziali (chiamati seed URL)
- 2) Ripetere i seguenti passi finché la download queue non è vuota:
 - a) Selezionare un URL dalla download queue e recuperare il suo contenuto
 - b) Salvare il contenuto recuperato nella web crawler repository
 - c) Estrarre gli hyperlinks nel contenuto recuperato (fetched)

d) Aggiungere i link alla download queue

Problema: Richiede molto tempo e potrebbero esserci duplicati, pagine non raggiungibili, spam...

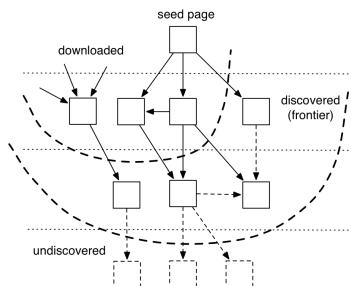
→ Dobbiamo gestire la scalabilità, la latenza, i duplicati e capire ogni quanto fane nuovamente i crawl dei siti (refresh).

Incremental web crawling

Il processo di crawling divide il web in tre subsets:

- Downloaded
- Discovered
- Undiscovered

Le pagine si muovono tra questi tre subset nel tempo.



Web crawling components

URL frontier: Coda contenente gli URL da essere crawled (possono essere ordinati per priorità)

Seen URL:insieme di URL che sono già stati crawled (bloomfilter)

Fetcher: Scava un unseen URL e lo salva nella data repository

Parsen: Estrae gli hyperlinks dalla pagina

URL Filtering: Elimina gli URLs che risultano essere immagini o che non sono permessi dai robots.txt files.

Content-seen filter: Elimina le pagine duplicate

Cosa DEVE fare un crawler?

1) Be Robust: Essere immune alle spider traps (pagine che fanno fare un infinito numero di richieste e fanno crashare il crawler) o altri comportamenti "malintentionati" da parte dei web servers.

→ Possibili soluzioni alle spider traps:

a) Checkare la lunghezza degli URL

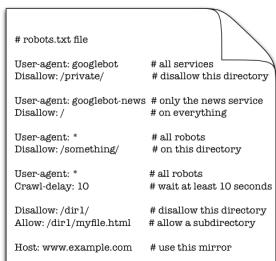
b) Implementare le "trap guards": Controllano cosa il crawler sta crawlando

2) Be polite: crawl solo pagine permesse (e non troppo velocemente) e rispettare i robots.txt

a) Explicit politeness: Specifiche da parte dei web master su quali portioni del sito possono essere crawled.

• Robots.txt file: È un file di testo (txt) che contiene una guida per i

web crawlers su quali parti del sito non dovrebbero essere crawled.
Spesso vengono cached da parte dei crawlers per efficienza



```
# robots.txt file
User-agent: googlebot      # all services
Disallow: /private/        # disallow this directory
User-agent: googlebot-news # only the news service
Disallow: /                # on everything
User-agent: *               # all robots
Disallow: /something/     # on this directory
User-agent: *               # all robots
Crawl-delay: 10            # wait at least 10 seconds
Disallow: /dir/             # disallow this directory
Allow: /dir/myfile.html    # allow a subdirectory
Host: www.example.com      # use this mirror
```

b) Implicit politeness: Evitare di fare troppe richieste in breve tempo

Duplicate Detection

Vogliamo evitare di fare il crawling di documenti duplicati.

La duplicazione può essere di due tipologie:

- Exact duplicates: Ad esempio un'immagine → Facili da eliminare (fingerprint)
- Near duplicates: Ad esempio documenti testuali → Difficili da eliminare (anche uno spazio fa differenza)
Possiamo usare la LSH (locality sensitive hashing) insieme allo shingling.

Dati due docs, usiamo un algo per trovare gli shingling, dove uno shingle è un n-gram nel documento.

Vengono poi comparati gli shingle, contati quanti sono in comune e usata una soglia per capire se i due docs sono duplicati → Questa tecnica viene usata insieme a LSH

Come valutare l'effectiveness del crawler?

- Quality: Valutare la qualità delle pagine (utili/fresh, high score/low score)
- Freshness: Contenuto aggiornato / non aggiornato
- Extensibility: Capacità di cogliere anche nuovi formati

- Coverage: Quanta parte del web stiamo coprendo

URL management

Un crawler mantiene due code:

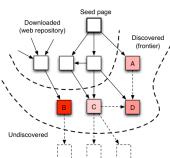
1) Discovery queue (URL frontier)

Sono URL che devono essere ancora scaricati per la prima volta.

Il crawler deve decidere l'ordine di crawling, serve della priorità che cerca di massimizzare le metriche visto prima (quality, coverage ...).

URL Prioritisation

- Random (A, B, C, D)
- Breadth-first (A)
- In-degree (C)
- PageRank (B)



2) Refreshing queue

URL che corrispondono a pagine già scaricate e che devono essere riscritte.

Abbiamo bisogno di un expire time che ci dica quando rinfrescarle e che sia adattivo alla frequenza di cambiamento di quella determinata pagina.

- a) Initializziamo tutti i refresh time ad un certo n° di giorni (es: 30 gg)
- b) Scavchiamo una determinata pagina dopo 30 gg, se essa non è cambiata la rimbustiamo nella coda con un expire time di 30 gg, altrimenti lo dimettiamo.
- c) Continuiamo così iterativamente, se la pagina è cambiata dimettiamo, altrimenti no doppiamo.

Refreshing

	A	B	C	D
PageRank	0.0003	0.0007	0.0002	0.0001
Average daily click count	47	332	2	1974
Last download time	2 hours ago	1 day ago	8 days ago	6 hours ago
Estimated update frequency	daily	never	minutely	yearly

- Random (A, B, C, D)
- Age (C)
- Link quality (B)
- Search impact (D)
- Longevity (A)

Crawling architectures

- 1) Single node crawlers
- 2) Multiple node crawlers: Crawlers paralleli in un singolo datacenter che lavorano sulla stessa coda \rightarrow MapReduce
- 3) Geographically distributed crawlers: Un Multiple node crawler per ogni continente

web evaluation

Per valutare le modifiche implementate a un search engine possiamo usare 3 metodi:

1) Offline evaluation: Usare test collections

- a) Collectionare un insieme di queries
- b) Per ogni query descrivere le info ricercate
- c) Avevi assessori che determinano quali docs sono rilevanti
- d) Valutare il sistema sulla base della qualità del ranking

- Pros
 - The experimental condition is fixed; same queries, and same relevance judgements
 - Evaluations are reproducible; keeps us "honest"
 - By experimenting on the same set of queries and judgements, we can better understand how one system is better than another
- Cons
 - Human assessors that judge documents relevant/non-relevant are expensive
 - Human assessors are not the user; judgements are made out of context
 - Assumes that relevance is the same for every user

È la prima valutazione che va fatta, se non funziona questa non funzionerà mai

Pro:

- Riproducibile
- Queries e qrels fissati
- Facile da confrontare con altre soluzioni

Contra:

- Assessors costosi e impegnativi
- Gli assessori non sono gli utenti, si assume la stessa rilevanza per tutti gli utenti.

2) User study Evaluation

- a) Fornire a un piccolo insieme di utenti diversi IR systems
- b) Far fare agli utenti diversi search tasks
- c) Valutare le system performance sulla base di:
 - Osservare cosa fanno
 - Chiedere perché l'hanno fatto

Pro:

- Dati dettagliati e vicini agli utenti

Contra:

- Metodo costoso (pagare utenti, scienziati...)
- Difficile generalizzare (pochi utenti)

3) Online Evaluation

- Vedere come gli utenti interagiscono con il sistema
- Fare usare ad alcuni utenti la versione cambiata
- Usare gli implicit feedback per capire se preferiscono la nuova versione del sistema (click, skips, saves, bookmarks...)

- Pros
 - System usage in a **natural environment**: users are situated in their natural context and often don't know that a test is being conducted
 - Evaluation can include a **lot of users** → **better samples** of the users population
- Cons
 - Requires a **service with lots of users**
 - Some users might experience a **sub-standard system performance**
 - Requires a **good understanding** of how implicit feedback signals predict a positive & negative user experience

problema da considerare: Biases

- Position Biases**: gli utenti sono più inclinati a esaminare e cliccare gli **higher ranked results**
- Contextual Biases**: le click di un utente dipende dai **nearby results**
- Attention Bias**: gli utenti cliccano di più sui risultati che catturano la loro attenzione
- Click accidentali, malintesi etc...**

Online Evaluation Approches:

1) A/B testing

- Splittare il traffico randomicamente tra due o più

Versioni



Ranking A



Ranking B

- Collezionare le metriche e analizzarle con test statistici per confermare che le differenze non sono "a caso".

Examples of online metrics used:

- Abandonment Rate** (% of queries with no click)
- Mean Reciprocal Rank** (mean of 1/rank for all clicks)
- User Engagement** (e.g. clicks per Query; Time to First Click, Time to Last Click, etc)

• Sessions per User

• Probability of switching to another search engine

2) Interleaving

- a) Generare una result list che combini i risultati di entrambi i sistemi



- b) Observare i click che avvengono sui risultati (se sono del sistema A o B)
c) Il sistema che ottiene più click vince.

- Allows to directly compare two rankings A & B
- Deals with issues of position bias and user calibration
- Usability issue:** Can only elicit pairwise preferences for specific pairs of ranking functions
- Interpretation issue:** Doesn't tell us much about document-level assessments and user behavior

Distributed Search

Come partitionare gli index in shards da essere distribuite sui vari nodi?

1) Term partitioning

- Different nodes are associated with a different subset of the vocabulary
- The queries are divided among the various query servers
- The broker collects all the results from each query server (all because low scores on some terms can have high scores on others)

2) Document partitioning

- Different docs are positioned in P different nodes
- Each server executes the query on N/P docs (load balanced)
- The broker collects the top K docs from each server

Image Basics

In questa "sezione" saranno descritte le caratteristiche base dell'immagini.

Tipi di immagini → Matrici di pixel

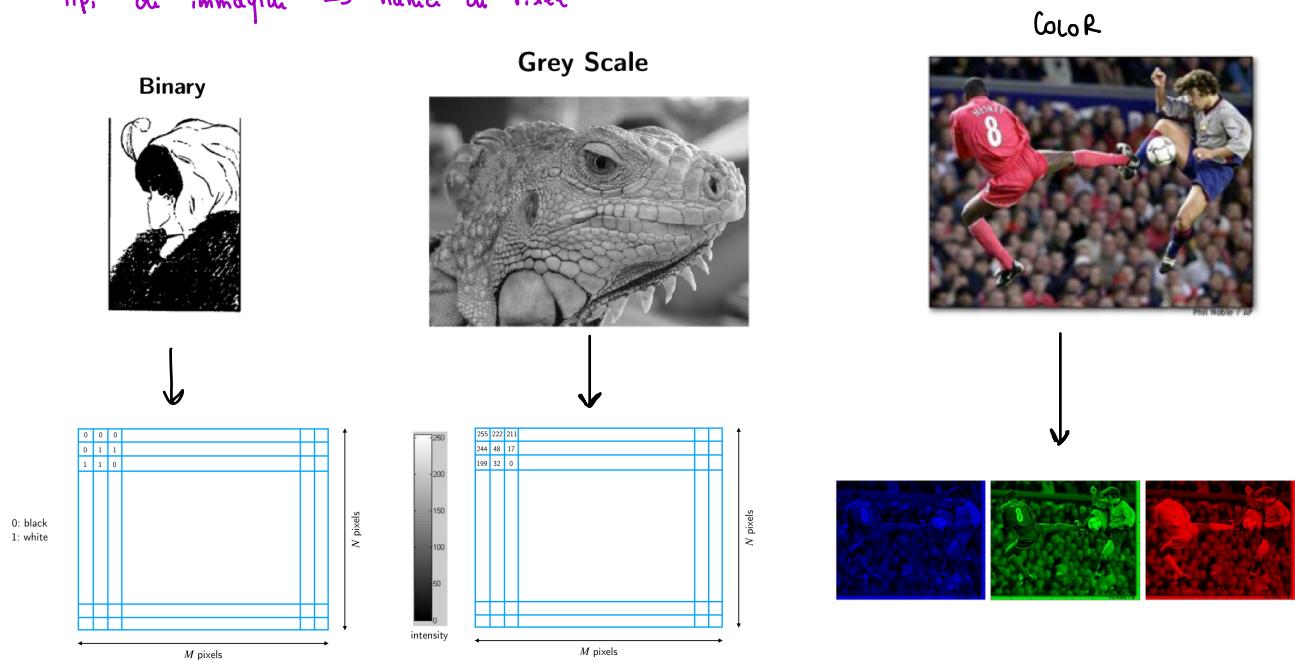


Image Resolution

La risoluzione è un parametro definito come "dots per inch" (DPI), il quale misura la densità dei pixel nello spazio.



Image Representation

Le immagini possono essere rappresentate come funzioni:

- Analog image: Il valore $f(x,y)$ da l'intensità nella posizione (x,y)

$$f: \mathbb{R}^2 \rightarrow \mathbb{R}^{\Delta}$$

- Digital image: Ha un dominio discreto e un intensity range

$$I: [a,b] \times [c,f] \rightarrow [0,255]$$

- Colour image: Ha una image function per canale.

$$I[x,y] = \begin{bmatrix} I_R(x,y) \\ I_G(x,y) \\ I_B(x,y) \end{bmatrix}$$

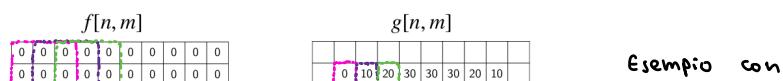
Systems e Filter

Filter: è una funzione la quale forma una nuova immagine trasformando i pixel della image originale.

Lo scopo è escludere info utili dall'img o trasformarle.

- features (edges, corners, blobs, ...)
 - effects (super-resolution, in-painting, de-noising, ...)

→ Moving avg Filter : è un 2D filter il quale fa la media dei valori in (blun) una finestra $k \times k$.



Esempio con
 $k = 3$



Systems: è un'unità che converte una funzione in input ($f[n,m]$) in un output

function ($g[n,m]$) , dove n, m sono variabili indipendenti.

Per fare ciò usa un system operator S , definito come un mapping da un membro dei possibili output di $g[n,m]$ ad ogni membro dei possibili input di $f[n,m]$.



Proprietà di un sistema:

- **Amplitude** properties
 - **Additivity:** $\delta(f[n], m) + g[m, n] = \delta(f[n, m]) + \delta(g[m, n])$
 - **Homogeneity:** $\delta(a f[n], m) = a \delta(f[n, m])$
 - **Superposition:** $\delta(a f[n], m) + \beta g[m, n] = a \delta(f[n, m]) + \beta \delta(g[m, n])$
 - **Stability:** $|f[m, n]| \leq k \Rightarrow |g[m, n]| \leq ck$
 - **Invertibility:** $\delta^{-1}(\delta(f[n], m)) = f[n, m]$
 - **Spatial** properties
 - **Causality:** for $n < n_0$ and $m < m_0$, $f[n, m] = 0 \Rightarrow g[n, m] = 0$
 - **Shift invariance:** $f[n - n_0, m - m_0] \xrightarrow{\delta} g[n - n_0, m - m_0]$

↓

Un sistema è lineare se soddisfa la superposizione.

Yel nosho visual system ē un "linear shift invariant" (LSI) system.

Passando una impulso function nel LSI system otteniamo una impulso response , questo

Serve per ottenere un filto $h[n,m]$ che ci dice cosa fa il sistema

Per calcolare $g[n,m]$ e $f[n,m]$ dal filo $(h[n,m])$ ottenuto possiamo:

+) Riscrivere $f[n,m]$ come somma di 2D impulse functions (scaled bidimensional signals)

$$f[n, m] = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} f[k, l] \times \delta_2[n - k, m - l]$$

2) Usiamo adesso la definizione di impulsive response, shift invariant principle e superposition

Principle :

$$\begin{aligned}\mathcal{S}(f[n, m]) &= \mathcal{S} \left[\sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} f[k, l] \times \delta_2[n - k, m - l] \right] \\ &= \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} f[k, l] \times \mathcal{S}[\delta_2[n - k, m - l]] \\ &= \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} f[k, l] \times h[n - k, m - l]\end{aligned}$$

3) Sia un Discrete Convolution operator così definito:

$$\mathcal{S}[f] = f[n, m] \star h[n, m]$$

Essendo che abbiamo usato una 2D impulse function, il dischetto convolution operation ottenuto è if 2D convolution il quale permette di iterare su 2 dimensioni invece che 1.

$$f[n, m] \star h[n, m] = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} f[k, l] h[n - k, m - l]$$

Dot Product

$$\text{Per calcolare ad esempio } (f \star h)[0,0] = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} f(k, l)h[-k, -l]$$

if you don't flip is a convolution, not a correlation

Border Effects

Il border effect, cioè ad esempio bordi spigolosi o perdita di dettaglio nei bordi dell'immagine, può verificarsi con l'uso dei linear filters perché generano risultati più piccoli.

Per gestire questo problema possiamo usare differenti "padding modes" per aggiungere extra pixels all'immagine prima di applicare il filtro e assicurarsi che il ricalco usato da ogni pixel sia all'interno dei limiti dell'immagine (e ridurre così il border effect).

1) Full padding: Aggiunge all'immagine abbastanza pixels ai bordi dell'immagine per far sì che l'output image abbia la stessa dim. dell'input image.

- I pixel aggiunti (padding) hanno valore 0.
- La dimensione dell'output sarà $(H+K-1) \times (H+K-1)$

2) Same padding: Aggiunge all'immagine abbastanza pixels ai bordi dell'immagine per far sì che l'output image abbia la stessa dim. dell'input image, ma solo su una dimensione.

- I pixel aggiunti (padding) hanno valore 0.
- La dimensione dell'output sarà $(H+K-1) \times H$ oppure $H \times (H+K-1)$

3) Valid padding: Non aggiunge nessun pixel ed applica il kernel solo se esso è interamente nell'immagine.

- No padding
- La dimensione dell'output sarà $(H-K+1) \times (H-K+1)$

OSS: • H = Dimensione immagine ($H \times H$)

• K = Dimensione kernel ($K \times K$)

Image Derivative

le image derivatives sono usate per calcolare il gradiente di un' immagine il quale misura come l'intensità o il colore dell' immagine cambia nello spazio.

One-dimensional discrete derivative

Nelle one-dimensional discrete derivative si considera come l'intensità o il colore dell' immagine cambia nello spazio di una one-dimensional image.

Vengono calcolate considerando la differenza tra pixel adiacenti:

- **Forward:** $f_x(x) = f(x+1) - f(x)$ → Pixel successivo
- **Backward:** $f_x(x) = f(x) - f(x-1)$ → Pixel precedente
- **Central:** $f_x(x) = \frac{1}{2}(f(x+1) - f(x-1))$ → Pixel Preced. e succ.
└→ più accurato

We can write them as filters

- **Forward:** $[1 \ -1 \ 0]$
- **Backward:** $[0 \ 1 \ -1]$
- **Central:** $\frac{1}{2}[1 \ 0 \ -1]$

Gradient of an image

Per calcolare l'intensità o il colore dell' immagine cambia nello spazio di una immagine, andiamo ad applicare il gradiente considerando le due dimensioni:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix} = [f_x \ f_y]$$

dim 1 dim 2

Il gradiente punterà nella direzione dove l'intensità cresce più rapidamente.

- **Gradient Magnitude:** è una quantità scalare che rappresenta la forza del gradiente di un' immagine e viene usata solitamente per rilevare angoli o altre caratteristiche.

$$\|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

- **Gradient Direction:** indica la direzione del gradiente calcolata come angolo.

$$\theta = \arctan\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

Gaussian Filter

Il Gaussian Filter (Kernel) è una matrice $N \times N$ con N dispari definito come segue:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \Rightarrow \text{su una singola dimensione invece: } g_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

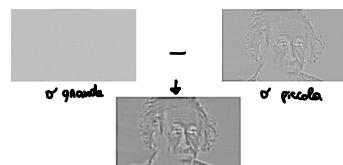
- x e y sono le coordinate all'interno del Kernel
- σ è il parametro che controlla la dimensione della Gaussian Function, differenti σ catturano strutture in scale differenti.

(DOG)

Abbiamo visto che facendo la differenza tra Gaussian con diverse standard deviations, il filto DOG può identificare regioni dove c'è un importante cambio d'intensità (angoli o altre features)

Image Matching

Il processo di image matching consiste in:



1) Feature detection: Trovare un insieme di Key-points

2) Feature description: Per ogni Key-point

- Definire una regione intorno ad ogni Key-point
- Estrarre e normalizzare il contenuto della regione
- Calcolare dalla regione normalizzata un local Feature Descriptor

3) Feature Matching: Trovare il miglior matching tra i descriptor di due immagini

4) Usare il feature matching per fare un altro task (downstream task), ad esempio "pose estimation".

↓
Task che viene costruito sull'output di un altro task

Feature detection

Il processo di feature detection consiste nel trovare feature points che possano essere riconosciuti in più immagini, anche in presenza di cambiamenti (cambi di luce, colore, intensità...), e che non siano punti comuni (ben distinti dal vicinato)

1) **Invariance**: Un feature point può essere rilevato anche dopo cambiamenti geometrici o

geometrici, ad esempio translation, rotation, scaling, color changes ---

2) **Distinguishability**: Un feature point deve poter essere usato per distinguere un'immagine

da un'altra. (ad esempio gli angoli di un quadrato, picchi di una montagna)

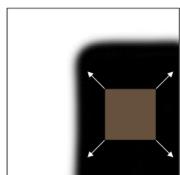
3) **locality**: Il feature point dovrebbe essere capace di essere riconosciuto in un local neighborhood senza considerare l'intera immagine

4) **Quantity**: Abbiamo bisogno di abbastanza feature points per coprire l'immagine.

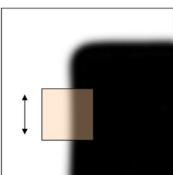
Harris Corner Detection

Per rilevare un corner usiamo una piccola finestra.

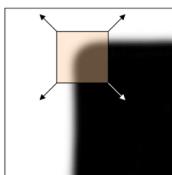
Muovendo la finestra in qualsiasi direzione, se siamo in un corner dovranno avere un grande cambiamento di intensità.



Flat region: no change in all directions



Edge: no change along the edge direction



Corner: significant change in all directions

Formalmente:

Shiftiamo la finestra $W (n \times n)$ di (u, v) pixel e sommiamo ogni pixel prima e dopo lo shift sommando le "squared differences".

$$E(u, v) = \sum_{(x,y) \in W} (I[x+u, y+v] - I[x, y])^2 \Rightarrow \text{Vogliamo che } E(u, v) \text{ sia il più grande possibile per ogni } (u, v).$$

Come calcoliamo $E(u, v)$?

1) La Taylor series Expansion di I è

$$I[x+u, y+v] = I[x, y] + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

Essendo lo shift (u, v) piccolo, allora

$$I[x+u, y+v] \approx I[x, y] + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v \approx I[x, y] + [I_x \quad I_y] \begin{bmatrix} u \\ v \end{bmatrix}$$

e quindi otteniamo:

$$\begin{aligned} E(u, v) &= \sum_{(x,y) \in W} (I[x+u, y+v] - I[x, y])^2 \\ &\approx \sum_{(x,y) \in W} (I[x, y] + I_x u + I_y v - I[x, y])^2 \\ &\approx \sum_{(x,y) \in W} (I_x u + I_y v)^2 \end{aligned}$$

2) $E(u, v)$ lo possiamo risolvere come:

$$E(u, v) \approx \sum_{(x,y) \in W} (I_x u + I_y v)^2 \approx au^2 + 2buv + cv^2 \quad \text{dove: } a = \sum_{(x,y) \in W} I_x^2 \quad b = \sum_{(x,y) \in W} I_x I_y \quad c = \sum_{(x,y) \in W} I_y^2$$

3) Scriviamo $E(u, v)$ in forma matriciale:

$$E(u, v) \approx [u \quad v] \begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \approx [u \quad v] H \begin{bmatrix} u \\ v \end{bmatrix}$$

second moment matrix = Massim Matrix = Harris Matrix

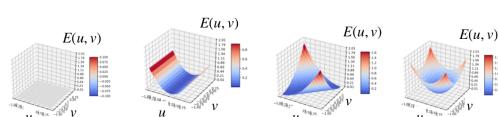
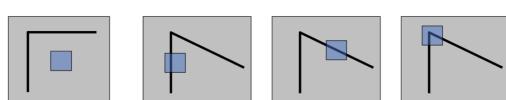
4) Vogliamo che $E(u, v)$ sia il più grande possibile per tutti gli u, v .

In termini di second moment matrix, cioè la matrice che descrive le proprietà statistiche di un'immagine in un piccolo vicinato di un punto, non vogliamo ci siano soluzioni

per $H \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, cioè non vogliamo che ci siano dipendenze dove E sia 0 perché

significherebbe che la finestra può scorrere in quella direzione senza cambi di

intensità.



$$\begin{array}{ccc} H = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & H = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & H = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \downarrow & \downarrow & \downarrow \\ E(u, v) = 0 \quad \forall u, v & E(u, v) = 0 \quad \forall v & E(u, v) = 0 \quad \forall u \end{array}$$

per identificare gli angoli useremo quindi gli autovetori di H : quali ci permettono di identificare punti dove l'intensità cambia bruscamente in due direzioni perpendicolari.

5) H è una matrice 2×2 quindi ha 2 autovetori (λ_{\min} e λ_{\max}) e due autovettori associati (x_{\min} e x_{\max}).

- Gli autovetori rappresentano il cambio di intensità nel punto interessato
 - $\lambda_{\max} = \text{amount of increase in direction } x_{\max}$
 - $\lambda_{\min} = \text{amount of increase in direction } x_{\min}$
- Gli autovettori rappresentano la direzione dal punto interessato che porta un maggior cambio di intensità.
 - $x_{\max} = \text{direction of the largest increase of } E$
 - $x_{\min} = \text{direction of the smallest increase of } E$

Cerchiamo punti con autovetori grandi e autovettori perpendicolari con ugual magnitudo.

Riassumendo gli steps:

- 1) Calcolo del gradiente per ogni punto dell'immagine
- 2) Crea una H matrix dalle entrie del gradiente (\forall punto)
- 3) Calcola gli autovetori di H e trova i punti con "large response" cioè $\lambda_{\min} > \text{soglia}$
- 4) Scegli i punti dove λ_{\min} è un massimo locale (nel suo vicinato) come features
Essendo λ_{\min} costosa da calcolare per ogni finestra, sfruttiamo le proprietà degli autovetori
e definiamo la misura di Horns la quale combina determinante e traccia.

$$R = \det(H) - \alpha \text{tr}(H)^2 = \lambda_{\min} \lambda_{\max} - \alpha(\lambda_{\min} + \lambda_{\max})^2$$

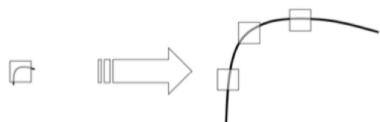
- $R > 0 \Rightarrow \text{Corner}$
- $R < 0 \Rightarrow \text{Edge}$
- $R \approx 0 \Rightarrow \text{Flat patch}$

oss: Dal parametro α dipende la sensitivity.

Questa tecnica è invariante alla rotazione e translation (perché i corner suotati avranno gli stessi autovalori).

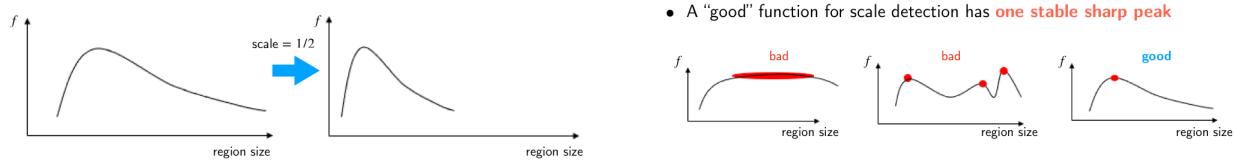
Questa tecnica è parzialmente invariante ai cambiamenti fotomehici perché ad esempio se diminuiamo il contrasto, diminuiamo anche R e quindi un punto che era un angolo scende sotto la soglia e non lo è più.

Questa tecnica non è invariante allo scaling, perché la window size è fissata, quindi cambiando lo scaling i corner potrebbero cambiare.

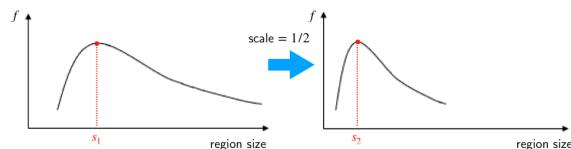


Questo problema può essere risolto cambiando la scala della finestra.

Per gestire la differenza di scale tra due immagini e poter fare matching viene disegnata una funzione della regione che è scale invariant, cioè è la stessa per regioni corrispondenti anche se sono in scala diversa.



L'approccio comune è di prendere il massimo locale di questa funzione perché non dovrebbe cambiare in base allo scaling.



L'idea è quindi di usare questa funzione per trovare il massimo locale (in intensità) e successivamente usare Harris Algo per rilevare in quella regione gli angoli.

La funzione che abbiamo visto è la Laplacian.

Feature description

Rilevati i key points, vogliamo trasformarli in local features coordinates che sono invarianti rispetto a translation, rotation, scale (per descrivere cosa succede intorno a essi)

Rotation invariance

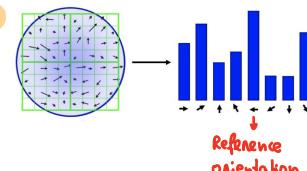
Dato un keypoint e la sua scale ottenuta da DoG (difference of Gaussians), vogliamo renderlo rotation invariant:

- 1) Ruotiamo le immagini sulla base del gradiente più grande in modo che esso rappresenti l'asse delle x (reference orientation)
- 2) Tutte le altre caratteristiche vengono descritte in relazione alla reference orientation in modo da renderle rotation invariant.

Per calcolare la reference orientation:

Dato il Keypoint e la sua scale (x, y, σ)

- 1) Calcoliamo gli image gradients sui vicini del Keypoint (le regioni dell'immagine)
- 2) Calcoliamo l'orientation histogram (composto dai vari gradienti) e selezioniamo il più grande (θ) il quale sarà la nostra reference orientation.



- 3) Per diventare rotation invariant, data la reference orientation Θ ruotiamo le gradient direction e le location di $-\Theta$.

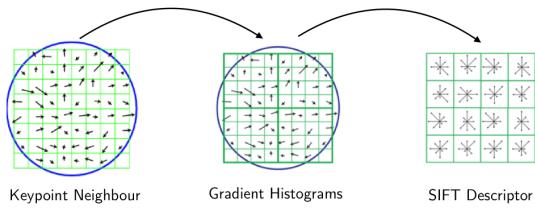
OSS: Avremmo potuto ruotare l'intera immagine di $-\Theta$ ma sarebbe stato più lento.

- Se abbiamo più di un picco, calcoliamo (al più) due descriptions per ogni Keypoint

SIFT Description

Data una regione di un key point (solitamente 8×8 o 16×16)

- 1) Viene creato un array di orientation histograms
- 2) I notated gradients vengono inseriti all'interno del loro local orientation histogram.
 - a) La gradient's contribution è divisa tra gli histograms vicini
 - b) Se è nel mezzo tra due histograms locations, da metà contributo a entrambi.
 - c) Le gradient contribution per gradienti lontani dal centro vengono ridotte.



Gli autori di SIFT hanno trovato che i migliori risultati erano con 8 orientation bins per histogram e un 4×4 array, cioè $8 \times 4 \times 4 = 128$ numeri.

Quindi un SIFT descriptor è un vettore di 128 dimensioni, invariant alla rotazione (perché abbiamo ruotato il descriptor) e invarianta alla scala (perché abbiamo lavorato su scaled images ottenute da DoG).

Per comparare due vettori (di due immagini diverse) possiamo fare il matching dei keypoints usando la distanza euclidiana.

- Remember that the descriptor is made of gradients (differences between pixels), so it's already invariant to changes in brightness (e.g. adding 10 to all image pixels yields the exact same descriptor)
- A higher-contrast photo will increase the magnitude of gradients linearly. So, to correct for contrast changes, normalise the vector (scale to length 1.0)
- Very large image gradients are usually from unreliable 3D illumination effects (glare, etc)
 - To reduce their effect, clamp all values in the vector to be ≤ 0.2 (an experimentally tuned value), then normalise the vector again

Matching SIFT Descriptors

- Input:
 - D_1 = array of SIFT descriptors from image 1
 - D_2 = array of SIFT descriptors from image 2
 - θ = similarity threshold, e.g., 0.7
- Output:
 - Map M matching image 1 to image 2 descriptors
- The euclidean distance is used to measure the similarity between two descriptors
- Algorithm:


```

 $M \leftarrow$  an empty array of  $|D_1|$  elements
for  $i=0$  to  $|D_1|$ :
   $j =$  index of the nearest descriptor to  $D_1[i]$  from  $D_2$ 
   $M[i] = j$ 
return  $M$ 
```

Audio Retrieval

In questa sezione analizzeremo il necessario per fare audio retrieval.

Fourier Transform

Dato un segnale analogico integrabile $x: \mathbb{R} \rightarrow \mathbb{R}$, la Fourier transform $X(f)$ del segnale $x(t)$

è definita come:
$$X(f) = \int_{-\infty}^{+\infty} x(\tau) e^{-2\pi i f \tau} d\tau \quad \text{for all } f \in \mathbb{R}$$
 dove f è la frequenza in Hz

Il valore $|X(f)|$ è il Fourier coefficient alla frequenza f e il suo valore assoluto è chiamato **magnitude alla freq. f** ($|X(f)|$).

Problema: La Fourier transform fornisce frequency info che è averaged su ll'intero time domain, non sappiamo cosa succede nei singoli time-slots.

Soluzione: Short Time Fourier transform (STFT)

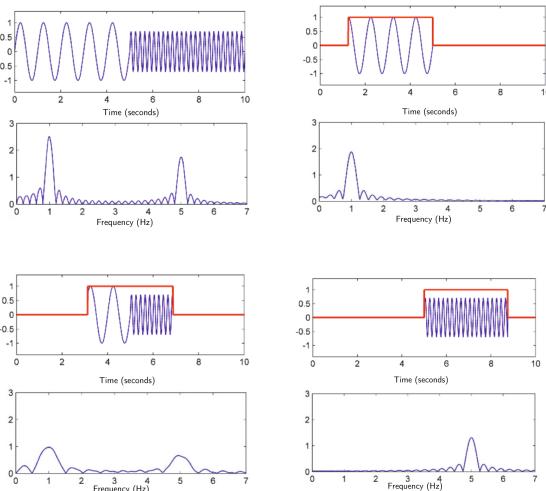
Short Time Fourier transform (STFT)

La STFT considera solo una piccola sezione del segnale (invece che l'intero segnale) usando una

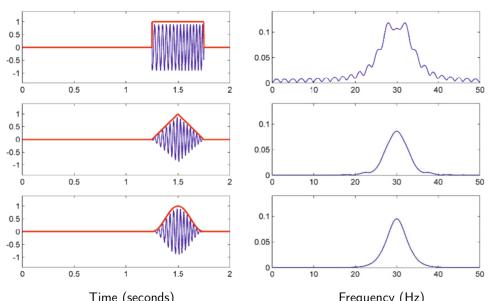
window function (funzione che non è zero solo per l'intervalle preso in considerazione)

Il segnale originale viene moltiplicato per la window function per produrre un **windowed signal**

e per ogni windowed signal viene calcolata la Fourier transform (per ottenere la freq. info a differenti time instances).



Diverse window functions producono diversi risultati.



Dato un segnale analogico $x(t)$ e una window function $g(t)$, la STFT $X(t, f)$

è definita come segue:

$$X(t, f) = \int_{-\infty}^{+\infty} x(\tau)g(\tau - t)e^{-2\pi if\tau}d\tau$$

Oss: $X(t, f)$ coincide con la Fourier transform del segnale localizzato $x(\tau) \cdot g(\tau - t)$ per una fixed time instance $t \in \mathbb{R}$.

Lo STFT di un segnale $x(t)$ può essere visualizzato con l'uso dello spectrogram, il quale è una rapp. due dimensionale di uno squared magnitude $|X(t, f)|^2$

Audio Retrieval Strategies

Content Based retrieval strategies (valgono anche per l'audio) possono essere classificate in base alla loro specificity:

1) Audio identification (High specificity)

Dato un piccolo frammento audio come query, identificare l'audio che è la sorgente della query.

2) Audio Matching (Medium specificity)

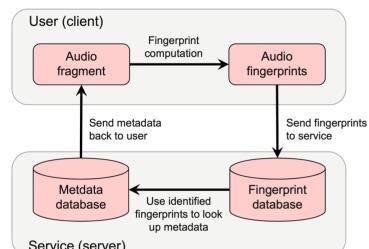
Dato un piccolo frammento di audio come query, restituire tutti gli audio che musicalmente corrispondono alla query (arrangiamenti o performance diverse)

3) Version Identification (Low specificity)

Dato un piccolo frammento di audio come query, restituire tutti gli audio che musicalmente corrispondono VAGAMENTE alla query (remix, cover...)

Audio Identification

L'Audio Identification utilizza gli audio fingerprints, i quali sono caratteristiche audio compatte e descrittive di un audio.



Audio Fingerprints

Sono caratteristiche audio compatte e descrittive di un audio e devono essere:

- High specificity: Devono saper identificare l'audio e distinguere tra tutti gli altri.
- Robustness: Devono essere robust al background noise e signal distortion.
(bandwidth)
- Compactness: Devono essere salvate insieme a milioni altre e trasmesse su canali limitati
- Scalability: La computazione deve essere possibile insieme a milioni di altre e essere semplice e efficiente

Fingerprints Design

Dato un audio nella forma di un sampled waveform:

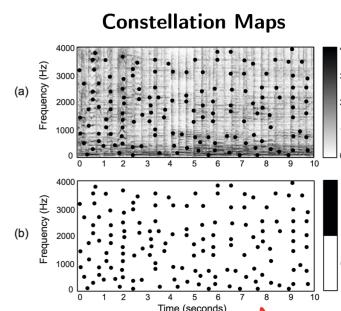
1) Calcolare la STFT $X(n, k)$ dove:

- n è un timestamp
- $k \in \{0, \dots, K\}$ è un "frequency stamp"

2) Reshingiamo l'intera STFT representation in pochi punti chiamati **peaks**

- τ e K sono parametri che definiscono il "neighborhood box" nel time-freq. plane

- I picchi sono punti dove fluisce l'energia del segnale (dove sta il maggior volume)



OSS:

- τ e K gestiscono la densità dei picchi selezionati
- I picchi rispettano la temporal locality perché usano un piccolo vicinato
- I picchi, con piccoli "time window shifts" nella STFT computation, sono invarianti a signal translations, cioè sono riproducibili indip. dalla posizione dell'audio fragment.

FingerPaint Matching

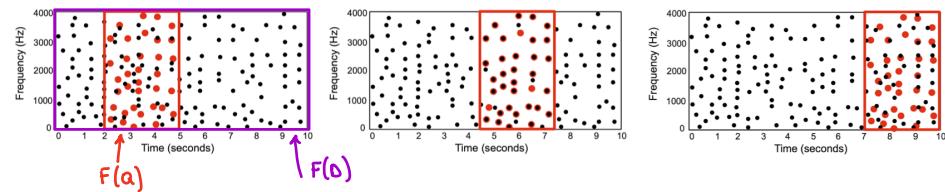
Sia Q un audio fragment con constellation map $F(Q)$ e D il "db recording" con constellation map $F(D)$.

il db viene considerato come se
fosse un intero audio
lungissimo

Mappa composta da solo i picchi

Vogliamo shiftare $F(Q)$ su $F(D)$ e quando il numero di matching point è significativo, abbiamo

un match.



Formalmente:

Dato un audio fragment X , la sua constellation map $F(X)$ è un insieme finito di coordinate (n, k) dove n è un timestamp e $k \in \{0, \dots, K\}$ è un freq. stamp.

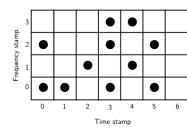
Shiftare la query Q di m posizioni produce la constellation map $m + F(Q)$ definita così:

$$m + F(Q) = \{(m + n, k) \mid (n, k) \in F(Q)\}$$

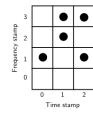
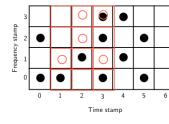
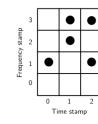
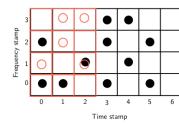
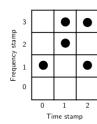
Semplicemente dobbiamo calcolare la cardinalità dell'intervalle della shifted query constellation map $(m + F(Q))$ con la recording constellation map $F(D)$

$$\Delta_F(m) = |(m + F(Q)) \cap F(D)| \text{ for } m \in \mathbb{Z} \quad \text{dove: } \Delta_F : \mathbb{Z} \mapsto \mathbb{N}$$

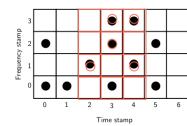
Esempio:



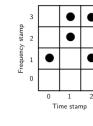
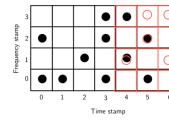
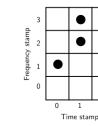
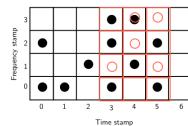
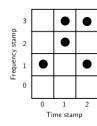
$$\begin{aligned} F(D) &= \{(0,0), (0,2), (1,0), (2,1), (3,0), (3,2), (3,3), (4,1), (4,3), (5,0), (5,2)\} \\ F(Q) &= \{(0,1), (1,2), (1,3), (2,1), (2,3)\} \\ \Delta_F(0) &= 1 \end{aligned}$$



$$\begin{aligned} F(D) &= \{(0,0), (0,2), (1,0), (2,1), (3,0), (3,2), (3,3), (4,1), (4,3), (5,0), (5,2)\} \\ F(Q) &= \{(0,1), (1,2), (1,3), (2,1), (2,3)\} \\ \Delta_F(1) &= 1 \end{aligned}$$



$$\begin{aligned} F(D) &= \{(0,0), (0,2), (1,0), (2,1), (3,0), (3,2), (3,3), (4,1), (4,3), (5,0), (5,2)\} \\ F(Q) &= \{(0,1), (1,2), (1,3), (2,1), (2,3)\} \\ \Delta_F(2) &= 1 \end{aligned}$$



$$\begin{aligned} F(D) &= \{(0,0), (0,2), (1,0), (2,1), (3,0), (3,2), (3,3), (4,1), (4,3), (5,0), (5,2)\} \\ F(Q) &= \{(0,1), (1,2), (1,3), (2,1), (2,3)\} \\ \Delta_F(4) &= 2 \end{aligned}$$

FingerPrint indexing

contiene un index

↑

Per essere veloci nel fare il retrieval, dobbiamo indicizzare i fingerprints con le posting lists.

Per fare ciò, consideriamo un cambio di terminologia:

- Frequency stamp $k \rightarrow$ hash h , farà il ruolo di un index term e l'insieme di

possibili frequency stamp $\{0, \dots, K\}$ saranno considerati H (dizionario) come un

- Constellation map $F(D) \rightarrow$ feature representation of D
- Per ogni hash $(h) \in H$ costruiamo una posting list $L(h)$
- La query sarà composta da un sottoinsieme di hash

3				●	●		
2	●			●		●	
1			●		●		
0	●	●		●		●	
	0	1	2	3	4	5	6
	Time stamp						

$L(3) = (3, 4)$
 $L(2) = (0, 3, 5)$
 $L(1) = (2, 4)$
 $L(0) = (0, 1, 3, 5)$

Fingerprint retrieval (boolean) con questa rappresentazione

Il nostro obiettivo è calcolare $\Delta_F(m) = |(m + F(Q)) \cap F(D)|$ for $m \in \mathbb{Z}$

Definiamo la shifted posting list $L(h) - n$ come: $L(h) - n = \{l - n \mid l \in L(h)\}$

Un elemento $(n, h) \in F(Q)$ contribuisce in $\Delta_F(m)$ se $m \in (L(h) - n)$, cioè

$\Delta_F(m)$ conta quanto spesso lo shift index m appare nella shifted list $L(h) - n$

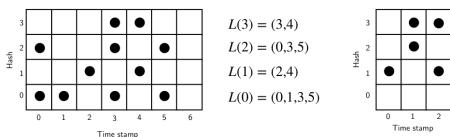
Introduciamo una characteristic function $\mathbf{1}_B(a) : A \rightarrow \{0, 1\}$

$$\mathbf{1}_B(a) = \begin{cases} 1 & \text{if } a \in B \\ 0 & \text{if } a \notin B \end{cases} \quad \text{for } B \subseteq A$$

Abbiamo $A = \mathbb{N}$ reale (\mathbb{Z}) e $B = L(h) - n$, abbiamo che

$$\mathbf{1}_{L(h)-n}(ma) = 1 \quad \text{sse } m \in (L(h) - n), \quad \text{cioè} \quad \Delta_F(m) = \sum_{(n, h) \in F(Q)} \mathbf{1}_{L(h)-n}(m)$$

Example



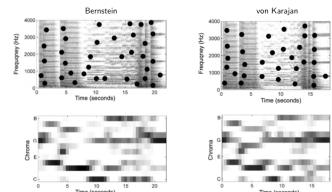
Performance

- Let $N = |F(D)|$ be the number of database items (very large)
- Let $M = |F(Q)|$ be the number of query items (quite small)
- Let $L = |\mathcal{H}|$ be the number of different hashes
- Assuming that the hash values are **evenly distributed** over \mathcal{H} , each of the L posting lists contains N/L elements
- **Space occupancy** for the database items is $O(N)$
- **Time complexity** for query processing is $O(MN/L)$
- The naive stripe and sheet approach requires $O(MN)$ time
- The indexing approach **boosts the performance** by a factor of L

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

Audio Matching

Dato un breve audio vogliamo restituire tutti gli esatti da tutte le registrazioni in un DB che musicalmente corrispondono alla query (la query non sta nel db, no exact matching). Non possiamo usare gli spectral peaks (e constellation map) perché non sono disegnati per gestire "music variations" (sono completamente diversi per le due canzoni). Usiamo il chromogram, il quale cattura le caratteristiche musicali invece che acustiche.



Differisce dalla constellation map dal fatto che usa sull'asse delle y i 12 chromas (dodici note sull'ottava del piano) invece che la frequency stamp.

Chromogram Matching

Dato una query audio fragment Q e un DB recording D (assumiamo che tutti gli audio del DB siano concatenati uno dopo l'altro), definiamo:

- Una chroma feature è un "vertical slice" nel chromogram normalizzato rispetto all'euclidean distance.

$X = (x_1, \dots, x_N)$ sono le chroma features per Q (N è piccolo)

$Y = (y_1, \dots, y_M)$ sono le chroma features per D (M è grande)

- Distance tra due chroma features x e y misurata con la cosine distance:

$$c(x, y) = 1 - \langle x | y \rangle \quad \text{cosine similarity}$$

L'idea è di shiftare la sequenza X sulla sequenza Y e comparare localmente, con la cosine distance, X con sottosequenze di Y .

Il matching è diagonale, cioè comparemo sequenze di stessa lunghezza, $X = (x_1, \dots, x_N)$

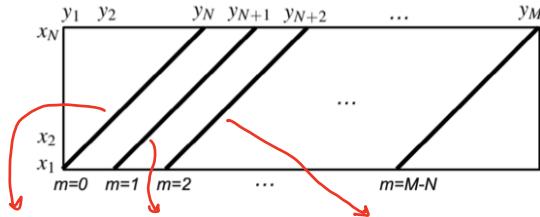
con tutte le sottosequenze $Y_m = (y_{1+m}, \dots, y_{N+m})$ di Y dove $m \in \{0, \dots, M-N\}$

Iter 1) Compono: x_1 con y_1 , x_2 con y_2 ...

Iter 2) Compono: x_1 con y_2 , x_2 con y_3 ...

Iter $M-N$) Compono: x_1 con y_{M-N} , ..., x_N con y_M

shift index



Compare x_1 con y_1 , Compare x_1 con y_2 , Compare x_1 con y_3
 Compare x_2 con y_2 , Compare x_2 con y_3 , Compare x_2 con y_4

Per ogni iterazione (match m) viene fatta la media delle cosine distances connisp:

$$\Delta_{Diag}(m) = \frac{1}{N} \sum_{i=1}^N c(x_n, y_{n+m}) \rightarrow \text{Matching function} \quad (\text{Venne applicata su ogni diagonale})$$

Il best match tra Q e Δ è dato da m^* , cioè il minimo costo tra tutti i diag. costs

$$m^* = \operatorname{argmin}_{m \in \{0, \dots, M-N\}} \Delta_{Diag}(m)$$

Vantaggio: Robust to noise

Svantaggio: Stiamo comparando chromatograms di stessa lunghezza, fallisce nel rilevare le differenze in tempo (perché stiamo cercando chromatogram della stessa lunghezza della Q)
 es: uno più veloce e uno più lento

Dynamic Time Warping

$$C_{n,m} = c(x_n, y_m)$$

Due sequenze

Data la cosine distance e la cost matrix tra X e Y , vogliamo trovare un allineamento tra esse il quale ha il "minimal overall cost".

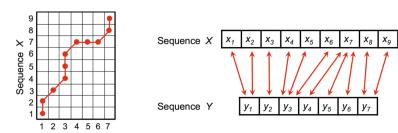
Intuitivamente, stiamo allungando o accorciando una delle due sequenze per fittarne.

Intuitivamente vogliamo trovare un allineamento (path) nella cost matrix che sia di basso costo.

Non vogliamo per tutti i path, ma solo i warping path.

Warping Path: Un warping path P di lunghezza L è una sequenza $P = (P_1, \dots, P_L)$ che soddisfa:

- Boundary condition: $p_1 = (1, 1)$ and $p_L = (N, M)$



y_{initio} e $y_{termine}$ devono essere gli stessi per entrambe le seq.

- Monotonicity condition: $n_1 \leq \dots \leq n_L$ and $m_1 \leq \dots \leq m_L$

→ Il movimento deve essere solo right or up ($\uparrow, \nearrow, \rightarrow$)

- Step size condition: $p_{l+1} - p_l \in \{(1,0), (0,1), (1,1)\}$ for $l \in \{1, \dots, L\}$

→ Lo step deve essere grande +, non possiamo stare fermo o saltare

Warping Path cost: è la somma dei costi lungo il path

$$c_P(X, Y) = \sum_{l=1}^L c(x_{n_l}, y_{m_l}) = \sum_{l=1}^L C_{n_l, m_l}$$

DTW distance: Tra tutti i possibili path, è il path di costo minimo (ottimale)

$$DTW(X, Y) = \min \{c_P(X, Y) \mid P \text{ is an } (N, M) \text{ warping path}\}$$

Considerare tutti i possibili path è esponenziale in N e M , però usando il dynamic programming possiamo disegnare un algo $O(N \cdot M)$.

DTW algorithm

Sia l' accumulated cost matrix $D \in \mathbb{R}^{N \times M}$ così definita:

$$D_{nm} = DTW(\underline{X[1:n]}, \underline{Y[1:m]})$$

$$X[1:n] = (x_1, \dots, x_n) \quad Y[1:m] = (y_1, \dots, y_m)$$

L'elemento D_{nm} è il time warp path cost minimo della sottosequenza.

- 1) Inizializziamo la matrice con i valori della prima riga e prima colonna

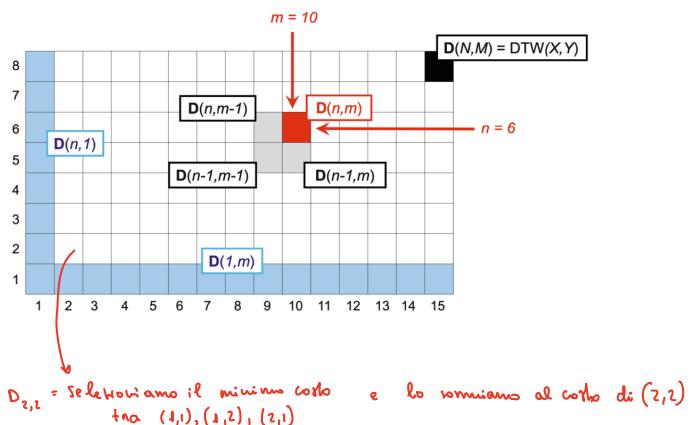
$$D_{n,1} = \sum_{k=1}^n C_{k1} \text{ for } n \in \{1, \dots, N\}$$

$$D_{1,m} = \sum_{k=1}^m C_{1k} \text{ for } m \in \{1, \dots, M\}$$

- 2) Calcoliamo i valori rimanenti ricorsivamente dai valori inizializzati

$$D_{n,m} = C_{n,m} + \min\{D_{n-1,m-1}, D_{n-1,m}, D_{n,m-1}\} \text{ for } n \in \{2, \dots, N\} \text{ and } m \in \{2, \dots, M\}$$

Esempio:



DTW Matching

Nell' operazione di matching vogliamo trovare la subsequence in Y che minimizza la DTW distance con X , cioè vogliamo trovare (a^*, b^*) t.c.:

$$(a^*, b^*) = \underset{(a,b): 1 \leq a \leq b \leq M}{\operatorname{argmin}} DTW(X, Y[a:b])$$

$Y[a:b] = (y_a, y_{a+1}, \dots, y_b)$

Per fare ciò usiamo delle modifiche al DTW algorithm visto precedentemente:

- La prima riga in D viene initializzata con $D_{1,m} = C_{1,m}$ for $m \in \{1, \dots, M\}$ per realizzare l'idea del "skippare l'inizio di Y quando matchato con X ".
oss: Con questa modifica, quindi, non dobbiamo iniziare un path dalla cella (1,1) ma possiamo scegliere una qualsiasi cella della prima riga.

- Con lo stesso concetto, l'intera ultima riga può essere usata come fine ed essere il global DTW cost.

Realizziamo l'idea del "skippare la fine di Y quando matchato con X ".

L' index b^* può essere determinato come

$$b^* = \underset{b \in \{1, \dots, m\}}{\operatorname{argmin}} D_{N,b} \rightarrow \text{minimo costo sull'upper row della matrice } D$$

L' index a^* può essere determinato facendo backtracking da b^* verso la prima riga di D e la colonna corrispondente al "primo hit" nella prima riga corrisponde a a^* .

Version Identification

Nel version identification task, dato l'intero audio recording vogliamo restituire tutti gli audio recording che possono essere considerati una versione di esso. Date due seq. X e Y vogliamo trovare sottoseq. di X e Y più simili possibile tenendo in considerazione le temporal deformations.

Intuitivamente, stiamo cercando molte matching subsequences con minimo costo ma preferibilmente lunghe (perché se sono corte rischiano di essere rumore, usiamo una soglia T) Utilizziamo anche esso i chromogram, ma invece di usare una cost matrix utilizza una similarity matrix.

La similarità tra due seq. è data da $s(x, y) = \langle x | y \rangle$ e di conseguenza la similarity matrix $S \in \mathbb{R}^{N \times K}$ è così definita $S_{n,m} = s(x_n, y_m)$.

Usiamo una soglia $T > 0$ e una penalizzazione $\delta < 0$ per mantenere forte similarità e penalizzare low similarity (subsequences sotto la soglia T).

Definizioni:

- Path P è una seq. che soddisfa i monotonicity e step size conditions ed è associato con due induced segments $\Pi_1(P) = \{n_1, \dots, n_L\}$ e $\Pi_2(P) = \{m_1, \dots, m_L\}$
- Lo score $\sigma(P)$ di un path è definito come

$$\sigma(P) = \sum_{l=1}^L S_{n_l, m_l}$$

- Il task di ottimizzazione corrisponde a trovare il path con similarità massima tra tutti i possibili path con start e end position arbitrarie.

$$P^* = \operatorname{argmax}_P \sigma(P)$$

Score - Maximisation Algorithm

Definiamo la accumulated score matrix $D \in \mathbb{R}^{N \times K}$ come:

$$D_{nm} = \max \{ \sigma(P) \mid P \text{ is a path ending at } (n, m) \}$$

1) **Inizializziamo la matrice**

$$D_{1,1} = \max \{0, S_{1,1}\}$$

$$D_{n,1} = \max \{0, D_{n-1,1} + S_{n,1}\} \text{ for } n \in \{2, \dots, N\}$$

$$D_{1,m} = \max \{0, D_{1,m-1} + S_{1,m}\} \text{ for } m \in \{2, \dots, M\}$$

2) **Ricorsivamente calcoliamo i restanti valori**

$$D_{n,m} = \max \begin{cases} 0 \\ D_{n-1,m-1} + S_{n,m} \\ D_{n-1,m} + S_{n,m} \\ D_{n,m-1} + S_{n,m} \end{cases}$$

3) **Scegliamo il path di similitudine massima**

$$\sigma(P^*) = \max_{(n,m) \in \{1, \dots, N\} \times \{1, \dots, M\}} D_{n,m}$$

Example

Se otteniamo un negativo num. si pone 0

↑

Similarity matrix (non normalizzata)

x_5	-2	-2	1	-2	1	0
x_4	-2	1	-2	1	-2	-2
x_3	0	1	-2	-2	1	-2
x_2	0	-2	1	2	-2	1
x_1	1	-2	1	1	0	-2
y_1	y_2	y_3	y_4	y_5	y_6	

Accumulated sim. matrix

x_5	0	1	4	2	4	4
x_4	0	3	1	3	3	3
x_3	1	2	0	2	5	3
x_2	1	0	2	4	2	3
x_1	1	0	1	2	2	0
y_1	y_2	y_3	y_4	y_5	y_6	

Belli ha le frasi non arrivano alla 1^a riga o a un valore uguale a 0

Bag of Words

È una rappresentazione delle parole nella quale ogni documento è rappresentato come un vettore ed ogni componente del vettore corrisponde ad una parola del vocabolario e il valore è il numero di volte che la parola occorre nel documento.

Il problema di questo approccio è che crea vettori sparsi ed, inoltre, non cattura il contesto/semantica tra le parole.

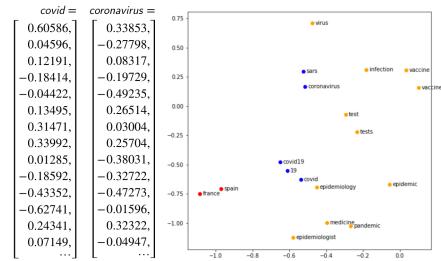
Per questo vengono usati i word embeddings.

Word Embeddings

Una parola è rappresentata da un real-valued vector denso chiamato "embedding".

I vettori di ogni parola dovrebbero essere t.c. siano simili per parole che appaiono in contesti simili.

La distanza tra questi vettori dipende dai contesti usati per la loro rappresentazione, quindi la vicinanza è basata sulla semantica.



La similarità usata è la "cosine similarity" implementata come dot product di vettori normalizzati. Solitamente i word embeddings sono composti da 300 valori (300-dimensioni).

Word Embeddings learning

Per imparare i word embedding della varie parole possiamo usare il Word2Vec framework.

DATA una grande collezione di testo, usiamo un approccio unsupervised.

- 1) Ogni parola del vocabolario viene rapp. come vettore
- 2) Per ogni posizione t in un testo, abbiamo la parola centrale c e il "contesto" dato dalle altre parole.
- 3) Usiamo la similarità dei word embedding di c e o per calcolare la probabilità di o dato c , $P(o|c)$
- 4) Aggiustiamo i vettori per massimizzare la probabilità.

Calcolo delle probabilità $P(w_{t+j}, w_t)$

Per ogni parola w usiamo due vettori:

- Un vettore v_w quando la parola w è la parola centrale
- Un vettore u_w quando la parola w è una context word

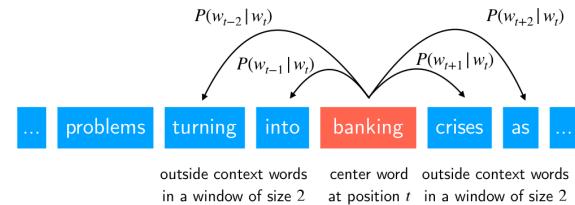
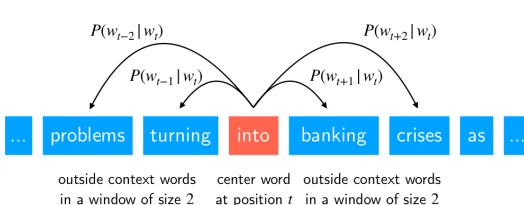
$$P(o|c) = \text{softmax}(u_o \cdot v_c) = \frac{e^{u_o \cdot v_c}}{\sum_{w \in V} e^{u_w \cdot v_c}}$$

dove: $w_t = c$ e $\|w_{t+j}\| = 0$

Quale context word è la più simile a c ?

Viene usata la softmax function per mappare valori arbitrari (positivi/negativi) in una distribuzione di probabilità (non usiamo la max perché non è derivabile e noi vogliamo backpropagare).

- L'esponentiale fa diventare tutto positivo
- Il dot product calcola la similarità tra vettori
- La normalization è fatta sull'intero vocabolario per avere una probability distribution.



Loss Function

Per ogni posizione $t = 1 \dots T$ predice le context words all'interno di una finestra fissata di dimensione m , data una parola centrale w_t .

Likelihood:
$$L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P_\theta(w_{t+j} | w_t)$$

center word
product over all terms in text
window of size m
context word

dove Θ denota tutte le vars da essere minimizzate

Per trasformare prodotti in somme

Loss Function:
$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P_\theta(w_{t+j} | w_t) \rightarrow$$
 Average negative log likelihood

Minimizzare la loss function corrisponde a maximizzare la likelihood.

Word embedding limitations

- L'ordine e i local context sono importanti (es: "not good", "not bad")
- Non è possibile creare embeddings per bigram, trigram (sparsity problems)
- Mappa una parola a solo un vettore, non gestisce semantiche differenti della stessa parola!

Neural Language Models

Distribuzione di probabilità sul vocabolario

Vogliamo imparare un language model usando i word embedding.

Input: word embeddings x_1, \dots, x_n

Output: language model $P(w_i | w_{i-1}, \dots, w_1)$

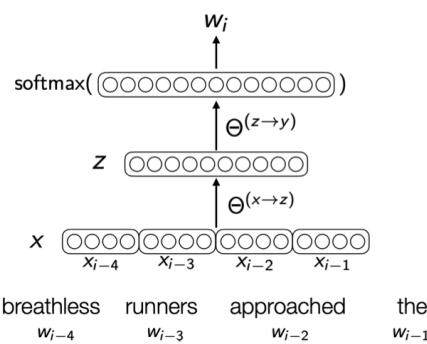
→ Dato un input di word embeddings, da in uscita il word embedding più probabile (predice il next token).

Feed Forward Neural LM

Le Feed Forward Neural LM sono composte da:

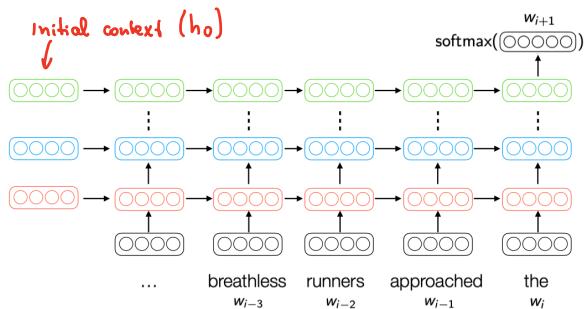
- Input di dimensione fissata (ad esempio $n=4$) i quali word embedding vengono concatenati in uno singolo $x = [x_{i-4} | x_{i-3} | x_{i-2} | x_{i-1}]$ e una activation function che converte x in z
- Un hidden layer z con activation function $\Theta^{(z \rightarrow y)}$ che converte z in una vocabulary size representation
- Un output layer che applica la softmax che converte la dense representation in una probability distribution sull'intero vocabolario.

Problema: L'input size è fissata ma non so quante parole all'interno devo usare per stimare correttamente il next token



Recurrent Neural LM

le RNN non hanno un input fisso, gli viene dato in input l'intero testo e ad ogni step, w_i , compone il precedente context embedding h_{i-1} con l'embedding della parola corrente (x_i) per creare un nuovo contesto h_i .



Problema:

→ Man mano, il contesto delle prime parole analizzate diventa sempre più debole!

OSS: Una versione alternativa può essere la bidirectional RNN che permette di usare anche le parole successive oltre alle precedenti.

Come usare questo language model? Transfer learning

Transfer learning

Il language model ottenuto (allenato su collezione di testo in grande quantità) può essere poi usato per trasferire i suoi layer base (senza output) ad un altro task più specifico.

Si fanno i layers del language model, si costruiscono "on-top-of-it" dei nuovi layer per il task specifico (output) e si allenano sul DB specifico (fine-tuning)



Il problema maggiore dei word embeddings è che ciascuna parola ha un singolo word embedding e i sinonimi quindi hanno rappresentazioni diverse.

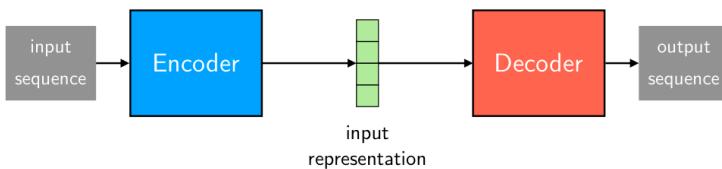
Diventata quindi importante fare leva sul contesto e imparare le rappresentazioni che dipendono da esso.

Per imparare a costruire contextualized word embeddings, possiamo usare i transformers.

Transformers

I transformers sono le migliori nel per il language modeling e sono composti da due blocchi:

- 1) Encoder: Riceve una input sequence (text) e costruisce una sua rappresentazione (embedding)
- 2) Decoder: Usa la rapp. dato dall' encoder, insieme ad altri input, per generare la output sequence



→ oss: sia l'encoder che il decoder
possono essere usati indipendentemente
(encoder-only o decoder-only)

Encoder-only Models

Questi modelli, chiamati anche auto-encoding models, possono accedere a tutti i token della input sequence e svolgono al meglio tasks dove è richiesta la comprensione dell'intero testo, ad esempio: Sentence Classification, word classification ...

Encoder-only models conosciuti:

- **BERT** (Google, 2018)
- **RoBERTa** (Facebook, 2019)
- **ALBERT** (Google, 2020)
- **DistilBERT** (HuggingFace, 2019)
- **ELECTRA** (Google, 2020)

Decoder Models

Questi modelli, chiamati auto-regressive models, possono accedere ai token della input sequence posizionati prima dell' output e svolgono al meglio task di "text generation".

Decoder only models conosciuti:

- GPT (OpenAI, 2018)
- CTRL (Salesforce, 2019)
- Transformer XL (Google, 2019)
- GPT-2 (OpenAI, 2019)
- **GPT-3** (OpenAI, 2020)

Encoder - Decoder Models

Questi modelli, chiamati anche sequence-to-sequence, sono composti via dall' encoder, che può accedere a tutti i token della input sequence, sia dal decoder che può accedere ai token della input sequence posizionati prima dell' output.

Questi modelli svolgono meglio tasks che richiedono la generazione di nuove sequenze dipendentemente da un dato input.

- **BART** (Facebook, 2019)
- mBART (Facebook, 2019)
- Marian (Microsoft, 2018)
- **T5** (Google, 2019)

Encoder - decoder models conosciuti:

Transformers learning Phase

1) Train a transformer su grandi quantità di testo in modo self-supervised.

Un transformer può essere allenato (come language model) in due modi:

- Masked language Model (MLM) → Train Encoder
 - a) Prendi una seq. e nascondi random dei token
 - b) Predici i token nascosti usando gli altri token (context tokens)
 - c) Aggiorna i pesi basandoti sulla miss prediction vs actual token
- Causal language Model (CLM) → Train Decoder
 - a) Prendi una seq. e nascondi l'ultimo token
 - b) Predici il token nascosto usando gli altri token (context tokens)
 - c) Aggiorna i pesi basandoti sulla miss prediction vs actual token

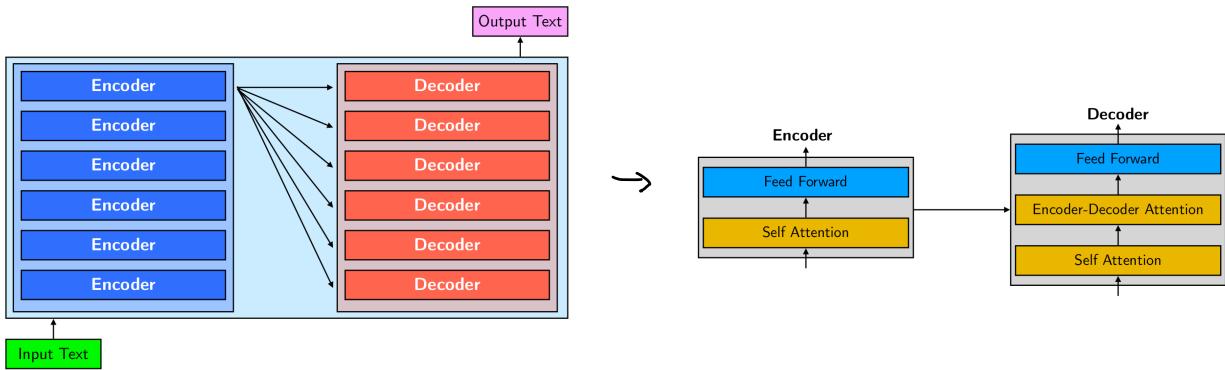
Questa prima tecnica possono svolgerla solo le big company perché per fare training da zero è richiesta una grande quantità di dati e risorse computazionali.

2) Fine-tune del pretrained language model con un dataset specifico per il task.

Essendo il modello pre-trained su tanti dati, richiederà meno dati per fare il finetune.

Transformer Overview

Un transformer non è in realtà composto da un singolo encoder e/o decoder, ma da più istanze di essi.



• Input text

Viene diviso in token, ed ogni token rappresentato da un embedding di 512 dimensioni tramite l'uso di uno static embedding (es: Word2Vec).

I token embeddings vengono usati solo dal primo encoder e gli altri encoders usano in input l'output dell'encoder precedente.

La lunghezza dell'input text (n° embeddings) è un hyper parameter che idealmente dovrebbe essere uguale alla lunghezza max di un batch nel training set, ma viene limitata solitamente a 500/512 input tokens.

• Self Attention :

È una rete neurale che permette all'encoder di "guardare" nelle altre posizioni della input sequence per cercare parole che possono portare info utili per un miglior encoding della parola. (considera l'interazione di ogni parola con tutte le altre per determinare il contesto)

• Feed Forward

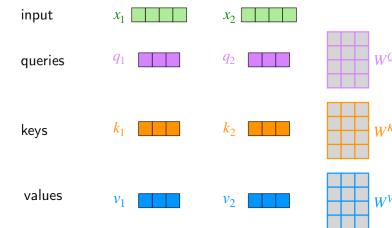
• Encoder-Decoder attention: È una rete neurale che aiuta il decoder nel concentrarsi solo su punti rilevanti della input sequence.

Self Attention Computation

1) Per ogni input embedding x_i , vengono calcolati 3 vettori: calcolati proiettando linearmente

l'input embedding in un vector space più piccolo (es: $\mathbb{R}^{512} \rightarrow \mathbb{R}^{64}$)

- Query $q_i \Rightarrow q_i = W^Q x_i$
- Key $k_i \Rightarrow k_i = W^K x_i$
- Value $v_i \Rightarrow v_i = W^V x_i$



oss:

- le matrici usate per la proiezione (projection matrix) vengono "imparate".
- Queries, keys e values sono "learned abstractions" of input embeddings.

2) Sono self attention tna ogni word pair nella input sequence

a) Prendiamo un input embedding alla volta, ad esempio x_1 , ed usiamo il suo query vector q_1 .

b) Calcoliamo lo score di q_1 vs gli altri input (usando il loro key vector)

$$s_{11} = q_1 \cdot k_1 \quad \text{e} \quad s_{12} = q_1 \cdot k_2$$

Dot product

oss: Stiamo calcolando la similitudine tna q_1 e i key vectors degli altri input

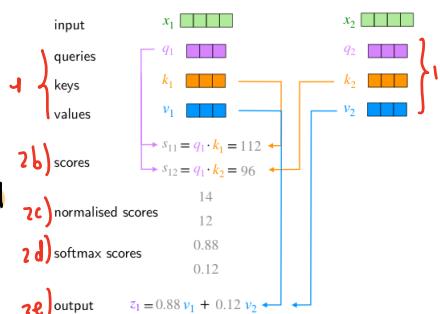
c) Normalizziamo gli score tramite la square root della dim. dei key vectors.

Viene fatto per avere gradienti più stabili.

d) Per ogni query vector convertiamo gli scores in una probability distribution sulle input, usando una softmax.

Il softmax score indica l'importanza del key embedding rispetto al query vector.

e) Usiamo i valori dei vettori per calcolare l'output sommandoli pesati per il loro softmax score



OSS:

- È possibile velocizzare la computazione **combinando gli embeddings in matrici**

$$X \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \times W^Q = \begin{bmatrix} Q \end{bmatrix}$$

$$X \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \times W^K = \begin{bmatrix} K \end{bmatrix}$$

$$X \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \times W^V = \begin{bmatrix} V \end{bmatrix}$$

This is called attention head

$$Z \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \text{softmax} \left(\frac{\begin{bmatrix} Q \\ K^T \end{bmatrix}}{\sqrt{64}} \right) \times \begin{bmatrix} V \end{bmatrix}$$

- Invece di usare una singola Q, K, V , possiamo usare più matrici indipendentemente imparate ($Q_0, K_0, V_0, \dots, Q_7, K_7, V_7$) che proiettano l'input embeddings in differenti representation subspaces.

Ogni matrice condiziona a differenti attention heads (z_0, \dots, z_7) che possiamo concatenare e mappare, tramite una nuova learned output matrix W^O , in un output space corretto.

Positional Encoding

Per gestire l'ordine dei token, il transformer aggiunge un **positional embedding p_i** vector ad ogni input embedding, cioè $x_i = x_i + p_i$.

I **positional embedding** seguono un pattern specifico che il modello impara ed aiutano a determinare la posizione del token.

In BERT, sono così calcolati:

$$p_t^i = \begin{cases} \sin(\omega_k t) & \text{if } i = 2k \\ \cos(\omega_k t) & \text{if } i = 2k + 1 \end{cases} \quad \text{where } \omega_k = \frac{1}{10000^{2k/d}}$$

Questa combinazione di seni e coseni encodes la sequenza alternata di 0s e 1s positions dei primi 512 numeri naturali.

Decoder

I decoder lavorano a steps, allo step i il primo decoder riceve in input l'output dell'ultimo encoder (corrispondente allo step $i-1$) e i corrispondenti positional embeddings.

L'output dell'ultimo encoder è trasformato in un insieme di attention embeddings.

Med e Ved prima di essere dato in input al decoder.

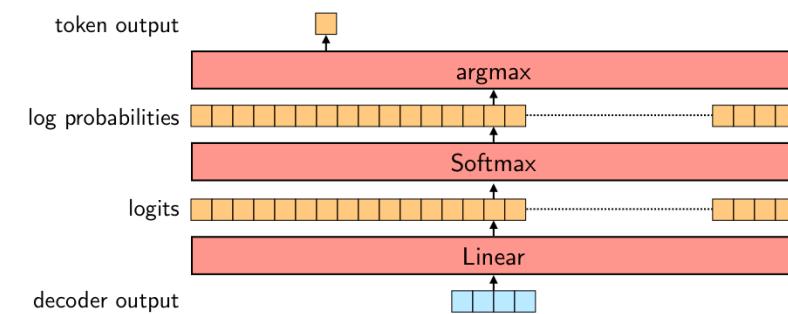
- Il self attention layer può solo vedere le "earlier positions" mentre le future position vengono mascherate (settate a $-\infty$) prima della softmax calculation.
- Il "encoder - decoder attention" layer lavora come il multi-headed self attention layer ad eccezione che crea la Q matrix dal layer sottostante e prende le Med e Ved matrix dall'output dell'encoder stack self attention layer.
- L'ultimo decoder da in output un vettore di float che vengono convertiti in parola come segue:

1) Viene usato un linear layer che proietta un output embedding in un vector space (di dimensionalità uguale al n° di token del vocabolario).

Questo output viene chiamato "logits vector" che corrisponde alla probabilità non normalizzata di un language model.

Un singolo logit corrisponde allo score di un singolo token del vocabolario.

2) Viene usato un softmax layer per trasformare i logits in log probabilities e viene selezionato il token con log probability massima.



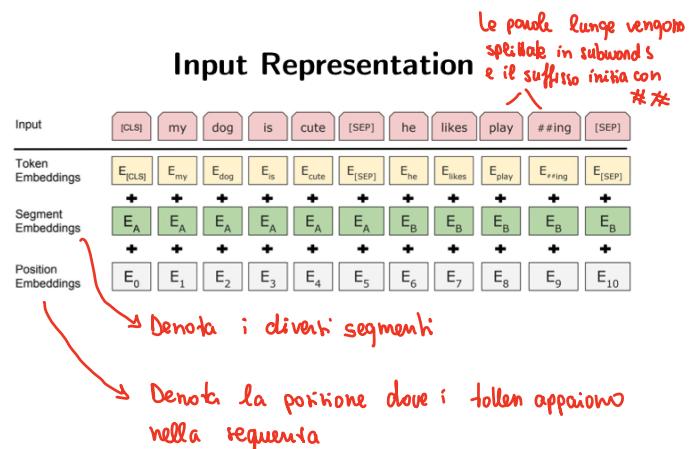
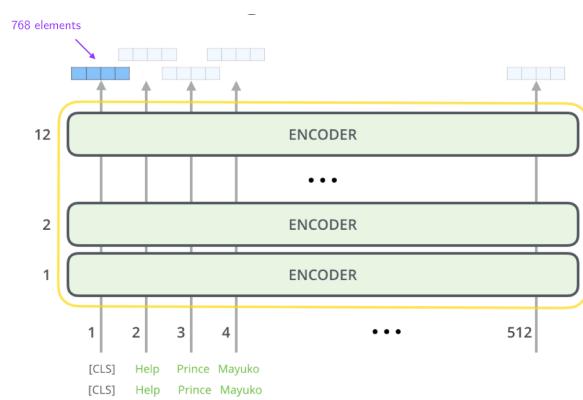
BERT

BERT è un encoder-only model composto da 12 o 24 encoders dipendentemente dalla sua versione (BERT base o BERT large).

Caratteristiche:

- I word embedding sono composti da 728 o 1024 dimensioni
- L'input size è massimo di 512 elementi
- Vengono usati dei token speciali, ad esempio:
 - [CLS] viene usato come primo carattere di ogni input sequence e significa "classification"
 - [SEP] viene usato per indicare segmenti differenti all'interno dell'input e per indicarne la fine.

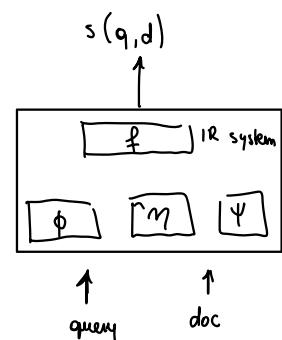
OSS: Anche i token speciali hanno un embedded vector corrispondente!



Transformers Model Representation and scoring

Un IR system, in generale, è così composto:

- 1) Document Representation $\Psi(d)$ $\Psi: D \rightarrow V_1$ ↗ word space
- 2) Query Representation $\phi(q)$ $\phi: Q \rightarrow V_2$
- 3) Query - Document Representation $\gamma(q, d)$ $\gamma: Q \times D \rightarrow V_3$



$$4) \text{ Scoring function } f(\phi(q), \psi(d), \gamma(q,d)) \quad f : V_1 \times V_2 \times V_3 \rightarrow \mathbb{R}$$

Abbiamo visto precedentemente due tipi di rappresentazioni:

- Bag of Words Representation \rightarrow Handcraft representation and scoring functions to impl. (index) IR system

$$V_1 = V_2 = N^{|V|} \rightarrow \text{tf vectors} \quad M = \phi$$

$$f = \text{TFIDF}(\phi(q), \psi(d)) \quad o \quad f = \text{BM25}(\phi(q), \psi(d))$$

- Learning to Rank Representation \rightarrow Handcraft representation & learned scoring function (query-doc pairs)

$\phi(q)$ = query only features

$f = \text{Regression trees}(\phi(q), \psi(d), \gamma(q,d))$

$\psi(d)$ = document only features

$\gamma(q,d)$ = query-doc features

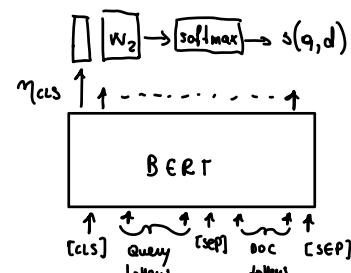
Ora vedremo due famiglie diverse di sistemi che usano i transformers e sono della tipologia "Learned representation" & Handcrafted scoring function

- 1) Interaction-focused systems: Usiamo i transformers per calcolare $\gamma(q,d)$

a) Encoder only (BERT)

1) Mono Bert

- In input vengono dati:
- [CLS]
 - Query tokens
 - [SEP]
 - Document tokens
 - [SEP]



In output viene data: Una vector representation della coppia query-document (m_{cls})

Steps:

$\uparrow l$ in Bert base = 768

I) $\gamma_{cls} = \text{BERT}(q, d)$ dove $\gamma_{cls} \in \mathbb{R}^l$ \rightarrow otteniamo rapp. query-document

II) $[z_0, z_1] = W_2 \gamma_{cls}$ dove $W_2 \in \mathbb{R}^{2 \times l}$ e $z_0, z_1 \in \mathbb{R}$ \rightarrow la proiettiamo in due dimensioni ottenendo due numeri usando W_2

III) $[p_0, p_1] = \text{softmax}([z_0, z_1])$ dove $p_0, p_1 \in [0, 1]$ \rightarrow Applichiamo la softmax e
• p_0 = probabilità di non rilevante (doc rispetto a q) otteniamo due probabilità

• p_1 = probabilità di rilevante (doc rispetto a q)

IV) $S(q, d) = p_1 \rightarrow$ otteniamo come score p_1

2) Other Approach

I) $\gamma_{cls} = \text{BERT}(q, d)$

II) $z = W_1 \gamma_{cls}$ dove $W_1 \in \mathbb{R}^{1 \times l}$ \rightarrow si proietta in una dimensione invece che due

III) $S(q, d) = z \rightarrow z è direttamente lo score$

b) Encoder - Decoder (T5)

Viene usata una tecnica chiamata **Prompt Engineering**, cioè dato un pattern di input fontiamo il transforme a produrre un output preciso.

Il pattern di input potrebbe essere il seguente:

query q document d relevant ?

dove le parole "query", "document" e "relevant" sono sempre uguali (prompt), mentre q e d cambiano.

Chiediamo quindi a T5 di generare il word embedding γ_{out} che corrisponde alla prossima parola da sostituire al punto interrogativo.

Steps:

I) $\gamma_{\text{out}} = \text{TS}(q, d) \rightarrow$ Genera word embedding corrispondente alla prossima parola

II) $[z] = W_v \gamma_{\text{out}}$ dove $W_v \in \mathbb{R}^{|V| \times \ell}$

Moltiplichiamo γ_{out} per una matrice W_v con V righe (dimensione vocabolario) per ottenere un language model z .

Selezioniamo dal language model solo due valori corrispondenti alle parole "true" e "false".

III) $[p_{\text{false}}, p_{\text{true}}] = \text{Softmax}([z_{\text{false}}, z_{\text{true}}]) \rightarrow$ Calcolo la probabilità delle due parole

III) $S(q, d) = p_{\text{true}} \rightarrow$ Seleziona come score la probabilità della parola "true"

Di TS e BERT è possibile scartare l'architettura e i pretrained weights, come fare il fine tuning?

Dato un training set, vogliamo trovare un Θ t.c $S_\Theta(q, d)$ sia soddisfacente.

Il nostro training set sarà composto da coppie $(q, d) \in Q \times D$ e per ogni coppia un label $g \in \{-, +\}$, quindi i nostri sample saranno:

$$T = \{(q, d^+, d^-)_i\}_{i=1, \dots, n}$$

dove:

- $(q, d^-) \rightarrow$ Vuol dire che il documento non è rilevante rispetto alla query
- $(q, d^+) \rightarrow$ Vuol dire che il documento è rilevante rispetto alla query

Vogliamo trovare Θ^* che minimizza la cross entropy loss, cioè:

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \sum_i l_{\text{ce}}(y_i, q_i, d_i)$$

$$\text{dove : } \in \left\{ L_{CE}(q, q, d) \right\} \approx \frac{1}{2n} \cdot \sum_{i=1}^n l_{CE}(q, d^+, d^-)_i$$

Per calcolare l_{CE} devono considerare la somma delle due cross-entropy:

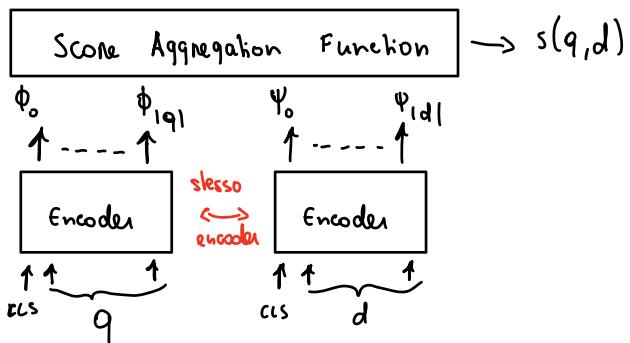
$$L_{CE}(q, d^+, d^-) = L_{CE}(+, q, d^+) + L_{CE}(-, q, d^-)$$

ognuna delle quali è definita come segue:

$$L_{CE}(q, q, d) = \begin{cases} -\log(s_0(q, d)) & \text{se } g=+ \\ -\log(1-s_0(q, d)) & \text{se } g=- \end{cases}$$

2) Representation focused systems: Usiamo i transformers per calcolare $\phi(q)$ e $\psi(d)$

Usiamo un encoder per rappresentare $\phi(q)$ e $\psi(d)$



OSS :

a) Cambio di notazione usato

$$\cdot \phi_0 = \phi_{CLS}$$

$$\cdot \psi_0 = \psi_{CLS}$$

b) Usiamo lo stesso encoder, quindi

$$V_1 = V_2$$

- $\phi(q) = \epsilon_{\text{encoder}}(q)$ dove $\phi(q) \in \mathbb{R}^{|\ell \times (|q|+1)}$
- $\psi(d) = \epsilon_{\text{encoder}}(d)$ dove $\psi(d) \in \mathbb{R}^{|\ell \times (|d|+1)}$
- $s(q, d) = f(\phi(q), \psi(d))$

a) Single Representation System (ignoriamo embedding query e document)

- $\phi(q) \equiv \phi_0 = \phi_{CLS}$
 - $\psi(d) \equiv \psi_0 = \psi_{CLS}$
- } Consideriamo che gli embeddings per i tokens, essendo che sono nello stesso vector space possono fare l'inner product per calcolare la similarità

$$\cdot s(q, d) = \phi_0 \cdot \psi_0$$

↑
inner product

b) Multi Representation System

b₁) Poly Encoder

La query viene rappresentata solo dal primo embedding, per il document vengono usati più embedding (non tutti, m)

- ϕ_0 dato da $\text{enc}(q)$
- $[\psi_0, \psi_1, \dots, \psi_{m-1}]$ dato da $\text{enc}(d)$
- $S = [s_0, s_1, \dots, s_{m-1}] \sim [\phi_0 \cdot \psi_0, \phi_0 \cdot \psi_1, \dots, \phi_0 \cdot \psi_{m-1}]$
Similarity Matrix ←
- $V = [v_0, v_1, \dots, v_{m-1}] = \text{softmax}(S) \rightarrow$ Calcolo la probability distrib.
- $\psi^* = \sum_{i=0}^{m-1} v_i \cdot \psi_i \rightarrow$ Calcolo l'averaged representation
- $s(q, d) = \phi_0 \cdot \psi^* \rightarrow$ Ottengo lo score

b₂) ME - BERT

La query viene rappresentata solo dal primo embedding, per il document vengono usati più embedding (non tutti, m)

- ϕ_0 dato da $\text{enc}(q)$
- $[\psi_0, \psi_1, \dots, \psi_{m-1}]$ dato da $\text{enc}(d)$
- $$s(q, d) = \max_{i=0, \dots, m-1} \phi_0 \cdot \psi_i$$
 chiamato Max-SIM operator

↳ Massimo delle similità fra query embedding e docs embedding selezionati

b₃) COL BERT

La query viene rappresentata da più embeddings (uno per query token) e stesso numero di embedding per il documento

- $[\phi_0, \phi_1, \dots, \phi_{m-1}]$ dati da $\text{enc}(q)$

- $[\psi_0, \psi_1, \dots, \psi_{m-1}]$ dati da $\text{enc}(d)$

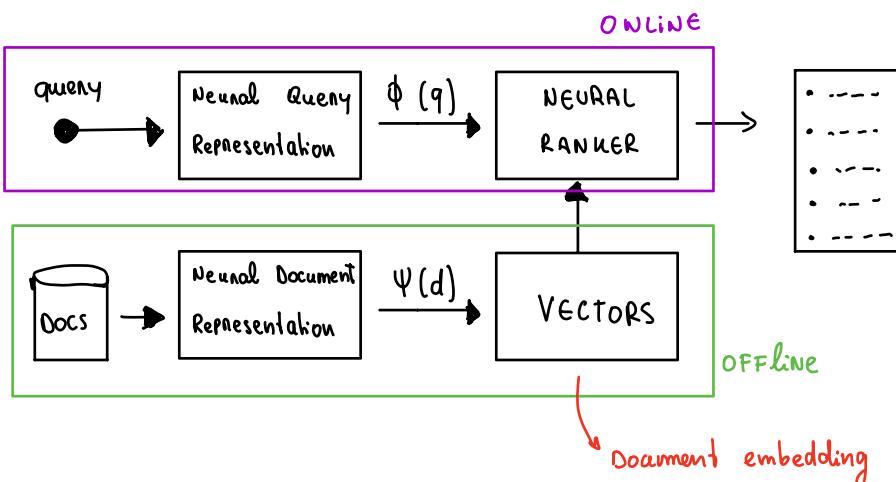
- $$S(q, d) = \sum_{i=0}^{|q|} \max_{j=0 \dots |d|} \phi_i \cdot \psi_j$$
 chiamato Sum Max Sim

↓ Somma le similitudini massime su tutti i query embeddings

OSS: Con i representation focussed system possiamo precompute document representation
a differenza degli interaction-focussed system.

Embedding Yndex

Quando usiamo i representation focussed systems, possiamo non usare gli inverted index, ci basta salvare i document embeddings.



Data la rappresentazione vista, vogliamo cercare i documenti con gli embedding più

Simili al query embedding, cioè vogliamo trovare Ψ^* tale che:

$$\Psi^* = \underset{\Psi \in \Psi}{\operatorname{argmax}} \quad \boxed{\phi \cdot \Psi} \quad \begin{array}{l} \rightarrow \text{Similarity Measure } s(q, d) \\ \downarrow \text{dot product} \\ \rightarrow \text{Maximum inner product search (MIP)} \end{array}$$

Abbiamo quindi bisogno di una struttura dati per salvare Ψ (insieme docs embeddings) e un algoritmo di ricerca per implementare il Maximum inner product search.

1) Flat Index

Hettiamo i docs embeddings dentro una matrice e poi scanniamo durante la ricerca tutta la matrice per trovare i k documents.

Funziona bene, ma è lenta, possiamo fare meglio

Possiamo definire

$$\Psi^+ = \underset{\Psi \in \Psi}{\operatorname{argmin}} \quad \|\Psi - \phi\| \quad \text{Nearest Neighbour search (NN search)}$$

Il problema di questa formula è che misura la distanza, ma noi vogliamo l'angolo come nella formula del Maximum inner product search.

Possiamo trasformare i vettori ϕ e Ψ in altri due vettori $\hat{\phi}$ e $\hat{\Psi}$ tale che il MIP e l'NN problem sono equivalenti, cioè risolvendo NN con $\hat{\phi}$ e $\hat{\Psi}$ risolviamo MIP con ϕ e Ψ originali.

Le trasformazioni che possiamo effettuare sono le seguenti:

$$\hat{\phi} = \begin{bmatrix} \phi / \|\phi\| \\ 0 \end{bmatrix}, \quad \hat{\Psi} = \begin{bmatrix} \Psi / M \\ \sqrt{1 - \|\Psi\|^2 / M^2} \end{bmatrix} \quad \rightarrow \quad \begin{array}{l} \text{MIP solution} = \Psi^* \\ \text{NN solution} = \hat{\Psi}^+ \end{array} \quad > \text{identiche}$$

Dimostrazione

Vogliamo dimostrare che $\min \|\hat{\phi} - \hat{\Psi}\| = \max \phi \cdot \Psi$

$$\begin{aligned}
 & \min \|\hat{\phi} - \hat{\psi}\| = \\
 &= \min \|\hat{\phi} - \hat{\psi}\|^2 \\
 &= \min \left\{ \|\hat{\psi}\|^2 + \|\hat{\phi}\|^2 - 2\hat{\phi} \cdot \hat{\psi} \right\} \quad \begin{array}{l} \|\hat{\psi}\|^2 = \frac{\|\psi\|^2}{H^2} + 1 - \frac{\|\psi\|^2}{H^2} = 1 \\ \|\hat{\phi}\|^2 = \frac{\|\phi\|^2}{H^2} + \phi = 1 \end{array} \\
 &= \min \left\{ 2 - 2\hat{\phi} \cdot \hat{\psi} \right\} = \\
 &\Rightarrow \min \left\{ -\hat{\phi} \cdot \hat{\psi} \right\} = \\
 &\Rightarrow \max \left\{ \hat{\phi} \cdot \hat{\psi} \right\} = \quad \rightarrow \\
 &= \max \left\{ \phi \cdot \psi \right\} \quad \leftarrow
 \end{aligned}$$

$\hat{\phi} \cdot \hat{\psi} = \frac{1}{\|\phi\|} \cdot \phi \cdot \psi \cdot \frac{1}{\|\psi\|}$
 Positive scalar
 Positive scalar
 Possiamo ignorare i fattori
nel massimo

Dimostrando questa equivalenza, possiamo usare le Nearest Neighbours techniques
 e metodi successivi cercano più o meno il closest (sono approssimati) per essere veloci.

2) LSH (Locality Sensitive Hashing) Index

Se due item sono vicini nella dimensione originale, lo sono anche dopo essere stati trasformati con LSH.

LSH è una proiezione randomica da l dimensioni a meno dimensioni in modo che punti vicini nella dimensione originale, c'è un'alta probabilità che lo siano anche nella projected dimension.

3) IVF (Inverted File) Index

Dividiamo lo spazio in cluster (ad esempio con k-means), creiamo una posting list per cluster che contiene i punti di esso in ordine di distanza dal centroid. Quando facciamo la search con la query, vediamo il closest centroid rispetto

alla query e viene considerata la distanza con i punti del cluster.

OSS:

- Stessa effectiveness del flat index, ma più veloce (se i cluster sono bilanciati)
- Soffre della curse of dimensionality (funziona bene in 2-3 dimensioni), anche LSH soffre della curse of dimensionality

4) Product Quantization (PQ) index

Dividere lo spazio in subspaces, applicare k-means su di essi indipendentemente e poi sommare le distanze nei vari subspaces.

Funziona bene nell'image search, ma non nel neural IR

5) IVFPQ: Fa un merge delle due tecniche IVF e PQ

6) Hierarchical Navigable Small World (HNSW) index

è un metodo basato su graphi

- a) Rappresenta ogni punto come nodo di un grafo
- b) Connnette ogni nodo con i K nodi più vicini (creando un KNN graph)
- c) Decidiamo un nodo come entry point per la ricerca
- d) Data una query, computiamo la distanza con l'entry point, ci muoviamo poi verso il nodo più vicino rispetto alla query e così via.

Problema: Nel caso peggiore dobbiamo esplorare l'intero grafo

→ è stato dimostrato però che aggiungendo degli archi randomici la ricerca diventa logaritmica nel numero di nodi.

Si chiama small world perché tutti i nodi hanno poche connessioni con gli altri nodi (al più K).

Si chiama hierarchical perché costruiamo una struttura dove abbiamo cluster di

nodi (documenti) in modo da scegliere come entry point il cluster più vicino alla query.

Problema_2 : Molte hyperparameter e richiede molta memoria