

DYNAMIC PRUNING

Per dynamic pruning si intende una strategia che ha lo scopo di rendere la fase di scoring più veloce semplicemente calcolando lo score di un sottoinsieme dei documenti e non di tutti.

Assunzione

L'utente è interessato ai soli top-K documenti, e durante il calcolo dello score si può capire se un certo documento non ha chance di essere in top-K. Anche perché lo score di un documento può essere determinato in anticipo o addirittura totalmente saltato. Questo meccanismo è detto **SAFE TO RANK**.

Fondamenti di Dynamic Pruning

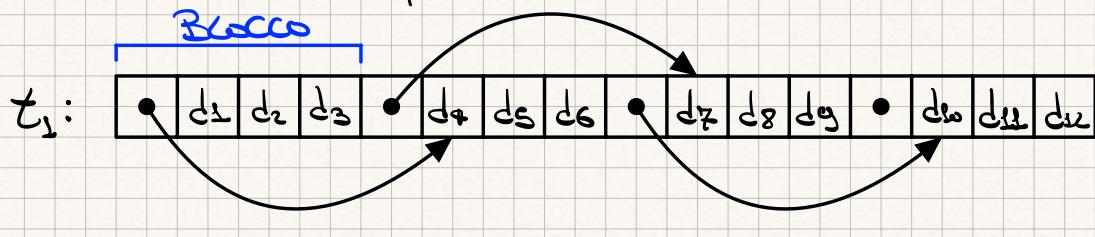
- 1) Terminatione anticipata
- 2) Comparare i limiti superiori degli score con threshold

SKIPPING

Dato che per raggiungere uno specifico docID in una posting abbiamo bisogno di leggere tutti i docIDs nel mezzo. Dobbiamo cambiare struttura dati.

Blocchi

Dividiamo la posting list in blocchi, e all'inizio di ogni blocco mettiamo un puntatore al blocco successivo.



Così per spostarsi a un certo docId si possono saltare quelli intermedi, che non vanno così decompressi.

Dimensione ottimale dei blocchi

1) Statica per ogni posting list

2) $\text{BlockDim} = \sqrt{N}$ con N numero di posting di quella posting list.

TERMINAZIONE ANTICIPATA

La valutazione dello score di un documento viene terminata precocemente se **tutte**, o **qualcuna** delle sue posting list relative ai termini della query non vengono recuperati dall'inverted index o non vengono usati dalla ranking function.

UPPER BOUND

Dato una funzione di score (TFIDF, BM25, ...) possiamo ricevere un limite superiore o **max score** per tutti i documenti contenuti nella posting list di quel termine della query. $\sigma(t)$.

$$\sigma(t) \geq s(t, d)$$

Il term upper bound $\sigma(t)$, si calcola offline, semplicemente calcolando $S(t, d) \forall d \in \mathcal{D}$ per poi prendere il massimo.

Dato una query q e un documento d possiamo calcolare il **limite superiore** di tutti i^e documenti semplicemente sommando $\sigma(t)$ dei termini della query o anche lo score attuale:

$$\sigma_q(q, d) = \sum_{t \in q} \sigma(t)$$

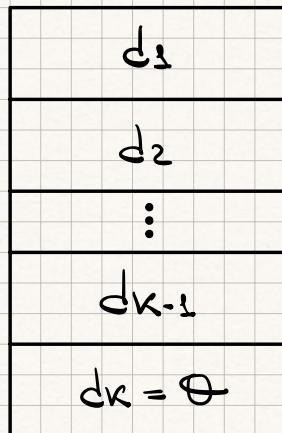
e/o

$$\sigma(q, d) = \sum_{t \in \hat{q}} s(t, q) + \sum_{t \in q - \hat{q}} \sigma(t)$$

Dove con \hat{q} si intende il set di termini già processati.

Heap e Threshold

Venne usata una coda prioritaria (**heap**) per i docs, l'ultimo documento in coda ha il più piccolo score θ , detto anche **threshold**. Se non si sono analizzati ancora K doc, il k -esimo score sarà θ



Da qui possiamo analizzare delle proprietà:

- 1) Il valore di threshold può solo crescere.
- 2) Un doc con $\text{score} < \theta$ non finirà mai fra i top- K .
- 3) Pruning Condition: per q e d , se $\sigma_d(q)$ calcolato usando gli score parziali e i $\sigma(t)$ è minore o uguale di θ il processo di scoring può essere terminato in anticipo.

MAX SCORE

L'algoritmo prende in ingresso le posting list dei termini della query, cui relativi: $\sigma(t)$.

$\sigma(t)$	PL		
1.3	11	13	26
2.1	11	22	24
3.7	11	24	27
4.0	11	24	28

Da notare come le posting list siano ordinate in ordine crescente di $\sigma(t)$. Le posting list vengono poi divise in due set:

- 1) ESSENTIAL LIST
- 2) NON-ESSENTIAL LIST

Nella lista NON-ESSENTIAL ci vanno tutte quelle liste la cui $\sum \sigma(t_i) < \theta$. Se un docId compare solo nelle liste NON-ESSENTIAL non ha nessuna chance di entrare nei top-k.

La prima posting list il cui $\sigma(t_i)$ permette alla $\sum \sigma(t_i)$ di superare la threshold θ delimita l'inizio della lista ESSENTIAL. Questo confine j è detto **pivot**.

L'algoritmo processa prima la lista ESSENTIAL con DAAT e sceglie i docId candidati a stare in top-k, poi scatta la lista NON-ESSENTIAL skipando i docId non candidati a essere nei top-k.

Una volta controllati tutti i NON-ESSENTIAL bisogna capire se il docId sotto analisi ha uno score sufficiente per stare in top-K, se entro allora sia Θ che il pivot vanno aggiornati.

δ_{sempio}

$S(t)$	PL	Θ	Score Parziale
1.3	11 13 23	6.0	0.0
3.4	11 22 26		
7.1	11 24 27		
11.1	11 23 28		

Pivot = 2

Portiamo con $\Theta = 6.0$, quindi con il K-esimo doc con $s(q, d) = 6.0$. Il pivot è 2.

Processiamo docId = 11

Per prima cosa processiamo le liste ESSENTIAL:

$t_3:$	7.1	11	24	27	\Rightarrow	Score Parziale
$t_4:$	11.1	11	23	28		$= S(t_3, 11) + s(t_4, 11)$

Ora spostiamo i puntatori al docId successivo:

$t_3:$	7.1	11	24	27
$t_4:$	11.1	11	23	28

Sappiamo che il docId = 11 può superare Θ con gli $\sigma(t)$ delle Non-ESSENTIAL ($2.8 + 34 = 6.2$) possiamo processarle.

$t_1:$	1.3	11	13	23	\Rightarrow	$\boxed{4.5}$	$\frac{\text{Score}}{\text{Parziale}}$
$t_2:$	3.4	11	22	26			$= 2.8 + s(t_1, 11) + s(t_2, 11)$

Ora spostiamo il puntatore al docId che ha valore maggiore o uguale a quelli da processare nella seconda iterazione, consigliati perché abbiamo già spostato il puntatore delle ESSENTIAL (24, 23):

$t_1:$	1.3	11	13	23
$t_2:$	3.4	11	22	26

Alla fine dell'iterazione avremo:

$\sigma(t)$	PL	Θ	$\frac{\text{Score}}{\text{Parziale}}$
1.3	11 13 23	6.0	4.5
3.4	11 22 26		
7.1	11 24 27		
33.1	11 23 28		

Algorithm 3.1: The MaxScore algorithm

Input : An array p of n posting lists, one per query term,
sorted in increasing order of max score contribution
An array σ of n max score contributions, one per query term,
sorted in increasing order

Output: A priority queue q of (at most) the top K $\langle \text{docid}, \text{score} \rangle$ pairs,
in decreasing order of score

MAXSCORE(p, σ):

```
1   q ← a priority queue of (at most)  $K$   $\langle \text{docid}, \text{score} \rangle$  pairs,  
    sorted in decreasing order of score  
2   ub ← an array of  $n$  document upper bounds, one per posting list,  
    all entries initialised to 0  
3   ub[0] ←  $\sigma[0]$   
4   for  $i \leftarrow 1$  to  $n - 1$  do → Calculate così i term upper bound  
5     ub[i] ← ub[i - 1] +  $\sigma[i]$  de uscere poi nei confronti  
6   θ ← 0  
7   pivot ← 0  
8   current ← MINIMUMDOCID( $p$ )  
9   while pivot <  $n$  and current ≠ ⊥ do  
10    score ← 0  
11    next ←  $+\infty$   
12    for  $i \leftarrow \text{pivot}$  to  $n - 1$  do // Essential lists  
13      if  $p[i].\text{docid}()$  = current then  
14        score ← score +  $p[i].\text{score}()$   
15        p[i].next()  
16      if  $p[i].\text{docid}()$  < next then  
17        next ←  $p[i].\text{docid}()$   
18    for  $i \leftarrow \text{pivot} - 1$  to 0 do // Non-essential lists  
19      if score + ub[i] ≤ θ then  
20        break  
21      p[i].next(current)  
22      if  $p[i].\text{docid}()$  = current then  
23        score ← score +  $p[i].\text{score}()$   
24    if q.push( $\langle \text{current}, \text{score} \rangle$ ) then // List pivot update  
25      θ ← q.min()  
26      while pivot <  $n$  and ub[pivot] ≤ θ do  
27        pivot ← pivot + 1  
28    current ← next  
29  return q
```

WAND

L'idea principale è quella di mantenere ordinate le posting list per docId corrente crescente.

L'algoritmo, a routine, valuta la condizione di pruning o **operatore WAND** (weighted and).

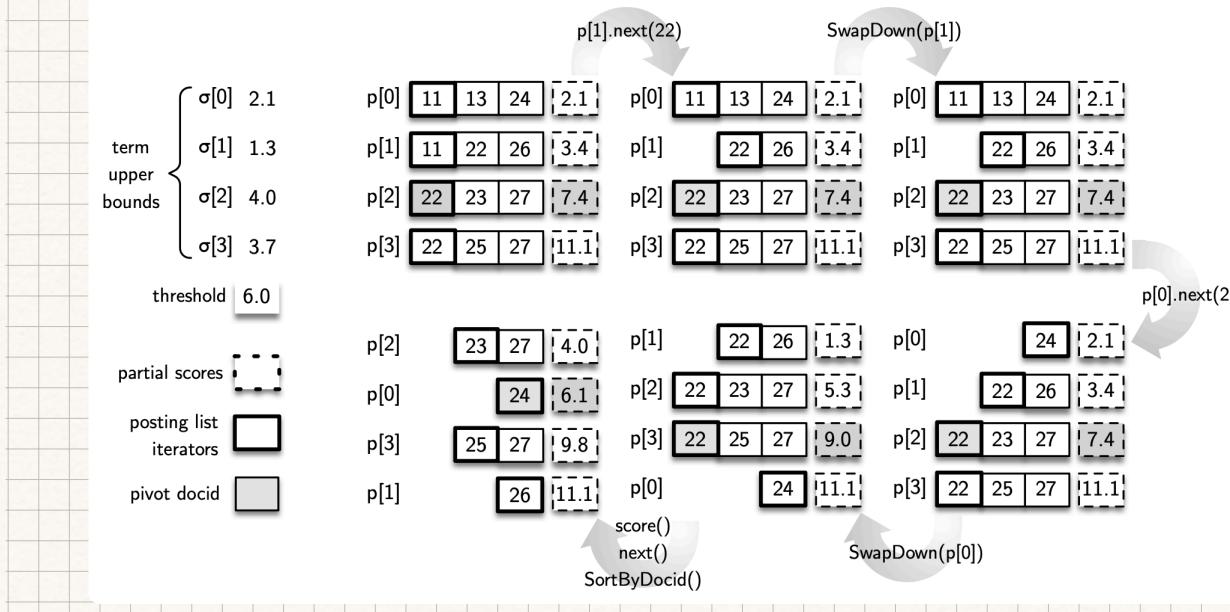
L'operatore WAND per $(X_0, w_0, \dots, X_m, w_m, \theta)$ è vero se

$$\sum_{i=1}^N w_i X_i \geq \theta$$

Nel caso di un sistema IR, i pesi w_i sono i term upper bound $\bar{t}_{ti}(d)$, mentre X_i è vero se il termine della query i -esimo è contenuto nel documento d . Sotto queste assunzioni l'operatore WAND $(X_0, \bar{t}_{t_0}, \dots, X_m \bar{t}_{t_m}, \theta)$ diventa:

$$\sum_{i=0}^m \bar{t}_{ti}(d) \geq \theta$$

Esempio WAND



Poiché dalla prima iterazione illustrata, abbiamo $\Theta = 6,0$ e le posting list già ordinate per il docID corrente.

1) Cerchiamo il pivot.

Poiché dalla $PL[\emptyset].docID() = 11$, abbiamo che $\sigma[1] = 2,1$ mentre $\sigma[\emptyset] + \sigma[1] = 2,1 + 1,3$ superano lo threshold Θ . Perciò 11 non può essere pivot. Continuiamo a sommare $\sigma[i]$ finché non superiamo Θ . $\sigma[\emptyset] + \sigma[1] + \sigma[2] = 7,4 > \Theta$. Il nostro pivot è il docID = 22.

Note: possiamo tranquillamente affermare che nessun $docID < pivot.id$ può mai avere una possibilità di entrare nei top-K, essendo le liste ordinate per current docID.

Attenzione: trovato il docID pivot, non possiamo ancora metterlo nei top-K, poiché non sappiamo se nella prima posting list ($PL[\emptyset]$) c'è un contributo per lo score di 22 nel nostro caso.

2) facciamo $PL[1].nextG5Q(22)$, e successivo riordinamento che in questo caso non avviene perché già ordinate.

3) Proviamo a fare su $PL[\emptyset].nextG5Q(pivot.id)$, ma troviamo docID=24 che fa uscire $PL[\emptyset]$ a finire in ultima posizione (4^a).

4) Consideriamo ancora docID=22 e il suo score approssimato pari a $\sigma[1] + \sigma[2] + \sigma[3] = 9,0$ che è abbastanza per calcolare lo score reale di 22.

Note: Possiamo calcolare lo score quando tutte le posting list fino a quella pivot contengono il pivot-id.

5) Durante la valutazione totale dello score, facciamo next(.) alle posting list corrente.

Algorithm 3.2: The WAND algorithm

Input : An array p of n posting lists, one per query term

An array σ of n max score contributions, one per query term

Output: A priority queue q of (at most) the top K (docid, score) pairs, in decreasing order of score

WAND(p, σ):

```

1   q ← a priority queue of (at most)  $K$  (docid, score) pairs,
2       sorted in decreasing order of score
3   θ ← 0
4   SORTBYDOCID( $p, \sigma$ )
5   while true do
6       σd ← 0
7       pivot ← 0
8       for pivot ← 0 to  $n - 1$  do                                // Find list pivot
9           if  $p[\text{pivot}].\text{docid}() = \perp$  then
10              break
11          σd ← σd + σ[pivot]
12          if σd > θ then
13              break
14          if σd ≤ θ then
15              pivot_id ←  $p[\text{pivot}].\text{docid}()$  → Pendi come pivot il docid della prima
16              if pivot_id =  $p[0].\text{docid}()$  then σd > θ → Posting list che ha permesso di
17                  score ← 0                                         // If matching doc pivot
18                  for i ← 0 to  $n - 1$  do
19                      if  $p[i].\text{docid}() \neq \text{pivot\_id}$  then
20                          break
21                      score ← score +  $p[i].\text{score}()$ 
22                      p[i].next() → Calcoliamo lo score per tutti
23
24                      q.push((pivot_id, score))
25                      θ ← q.min()
26                      SORTBYDOCID( $p, \sigma$ )
27
28      else
29          while  $p[\text{pivot}].\text{docid}() = \text{pivot\_id}$  do
30              pivot ← pivot - 1
31              p[pivot].next(pivot_id)
32              SWAPDOWN( $p, \sigma, \text{pivot}$ )
33
34  return q

```

Summi i valori $\sigma[i]$, finché non superi
la threshold θ .

// No list pivot found
Pendi come pivot il docid della prima
Posting list che ha permesso di
superare θ

Calcoliamo lo score per tutti
i contributi, e facciamo
next alle posting list che
hanno contribuito.

Note: Se durante la ricerca del pivot, non lo troviamo possiamo fermare l'intero algoritmo, perché nessun doc ha la possibilità di entrare nei top-k.

