Text Databases

Text Databases

- E’ un database in grado di memorizzare, gestire ed interrogare documenti testuali *non strutturati*

Text Databases

- L’obiettivo è di minimizzare il tempo necessario per localizzare le informazioni
- I risultati di una interrogazione sono ordinati in ordine decrescente di *rilevanza*
- Un documento è rilevante se l’utente che formula l’interrogazione giudica che il documento e l’interrogazione si riferiscono entrambi allo stesso argomento

Text Databases

- L’enfasi è sulla caratterizzazione dei requisiti informativi dell’utente
  - Ritrova tutti i documenti che contengono informazioni sulle squadre di tennis dei college americani che (1) hanno partecipato al torneo NCAA e (2) contengono informazioni sull’allenatore della squadra

Text Databases

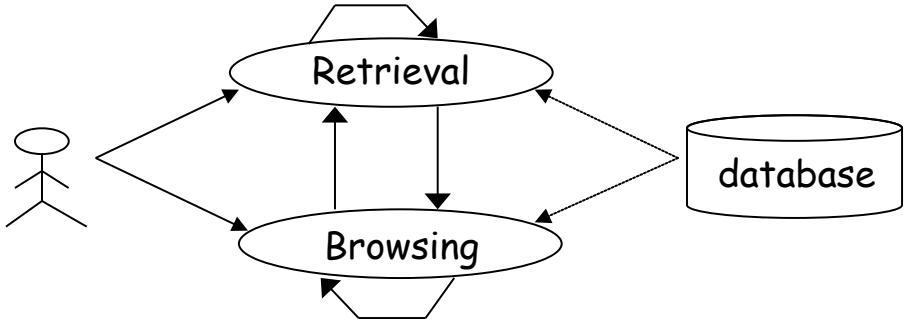
- I database di testi sfruttano tecniche sviluppate per i sistemi di Information Retrieval (IR)
- L'ambito dell'IR ha prodotto negli ultimi 20 anni:
  - Modelli per la rappresentazione di documenti
  - Architetture e linguaggi
  - Interfacce e metodi di visualizzazione
- Nonostante questo l'area dell'IR è sempre stata di interesse limitato

Text Databases

- L’avvento del WEB ha cambiato le cose:
  - È un repository universale di conoscenza
  - Accesso universale a costi ridotti
  - Nessuna autorità centrale
  - Il web ha però introdotto nuove problematiche (ad es. bassa qualità di definizione e struttura delle informazioni): le tecniche di IR sono viste come una chiave per trovare le soluzioni

Text Databases

- Le interazioni dell'utente:



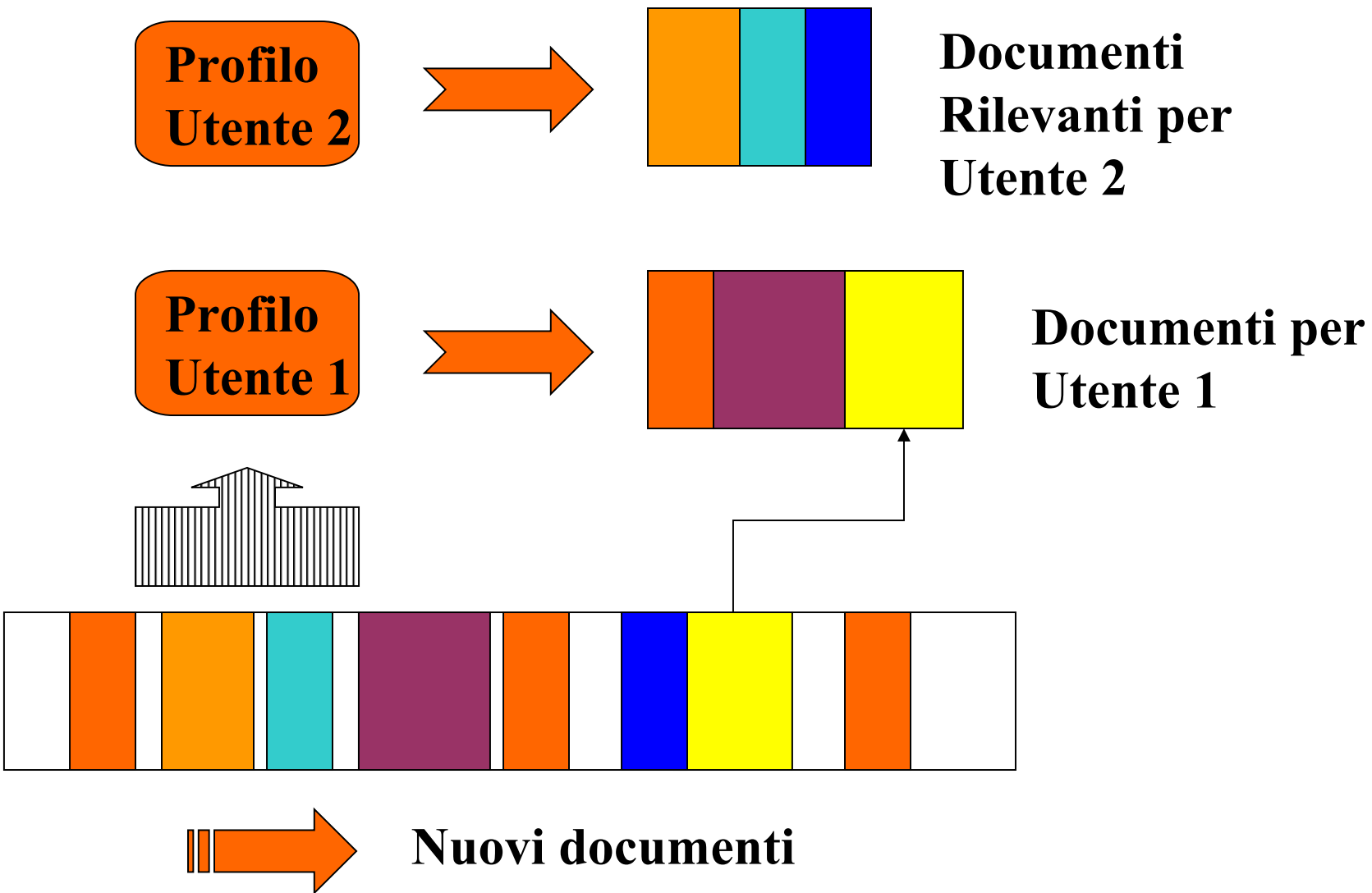
browsing: ricerca di informazioni, ma i cui obiettivi principali non sono ben definiti all'inizio e possono cambiare durante l'interazione con il sistema

Retrieval

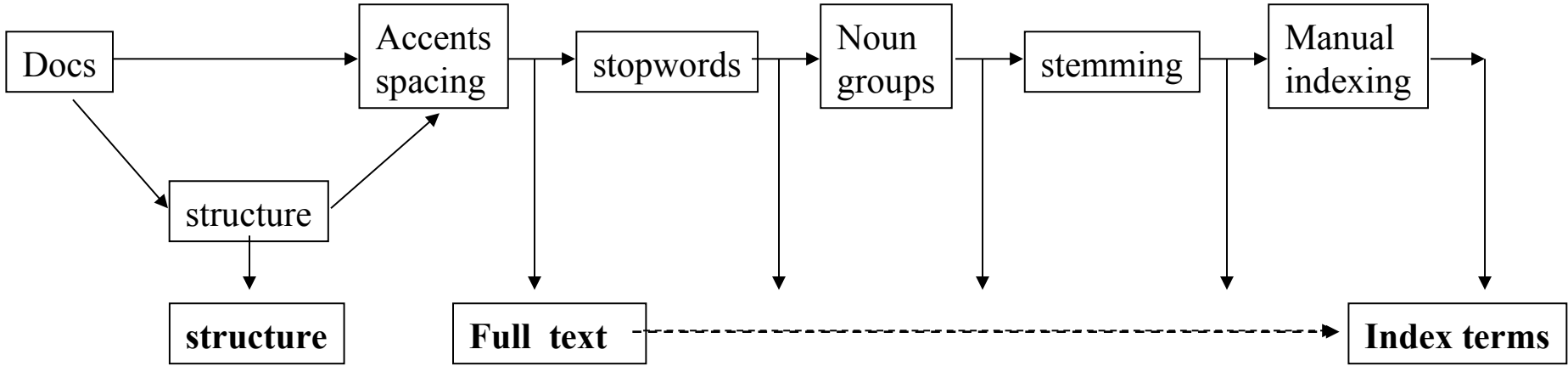
- Il retrieval può essere ulteriormente suddiviso in:
  - Ad hoc retrieval: insieme di documenti relativamente statico, richieste che variano
  - Filtering: insieme di richieste relativamente statico, documenti che variano

Ad Hoc

Filtering


Profilo Utente 2
Documenti Rilevanti per Utente 2
Profilo Utente 1
Documenti per Utente 1
Nuovi documenti

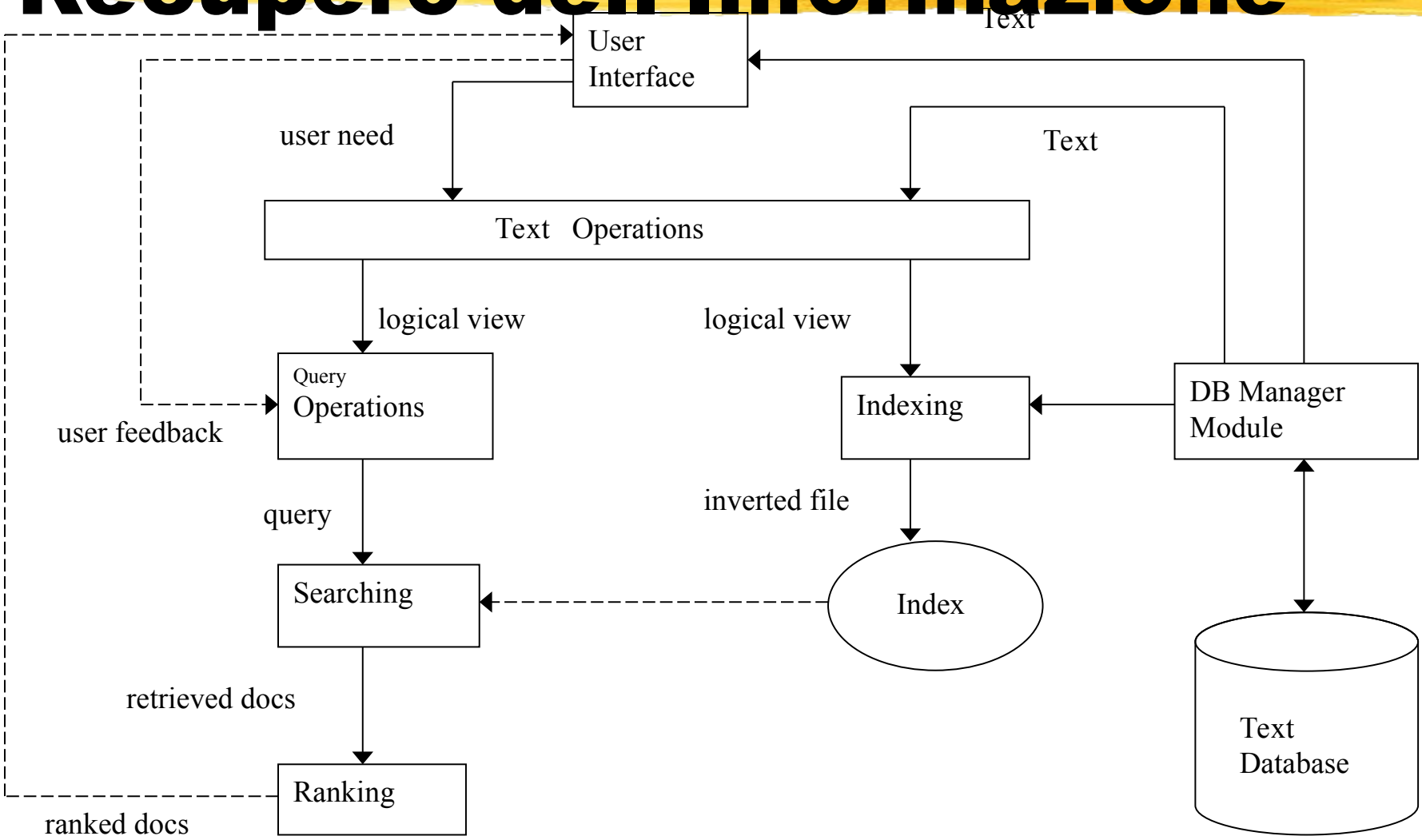
Struttura Logica di un Documento



Struttura Logica di un Documento

- structure: struttura interna del documento (capitoli, sezioni, sottosezioni)
- stopwords: articoli e congiunzioni
- noun groups: si eliminano aggettivi, avverbi, verbi
- stemming: ci si riduce a radice comune (es. plurale, singolare)

Recupero dell'Informazione



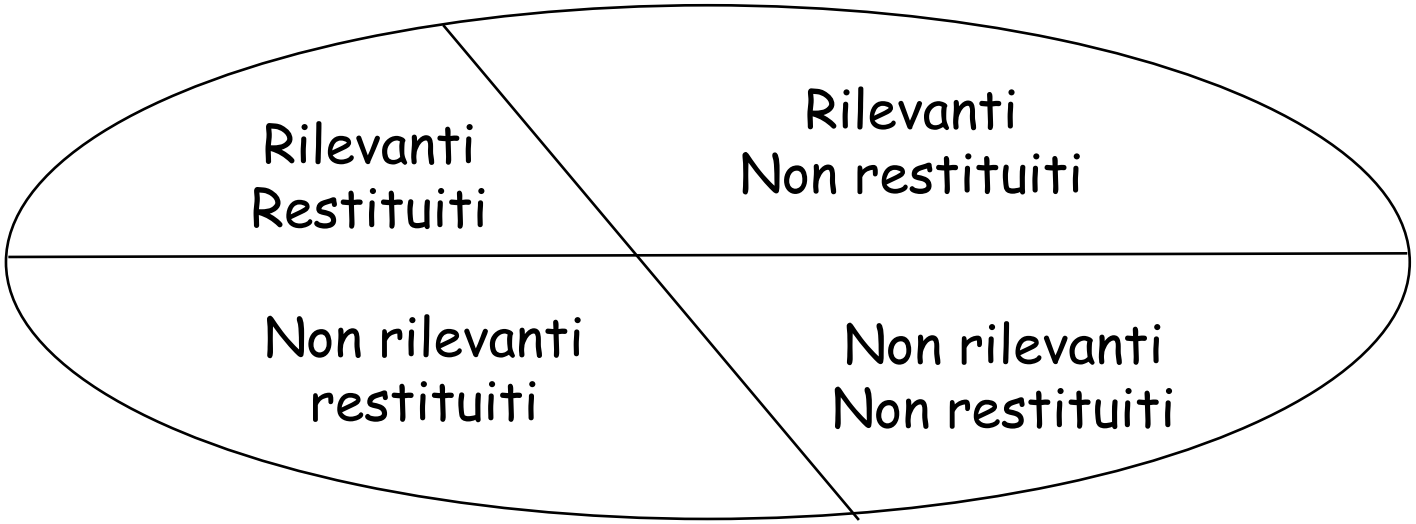
Text Databases

- Lo scopo è di reperire *tutti e soli* quei documenti che interessano l'utente
- Un sistema con tali caratteristiche non può però essere realizzato in pratica
- Per tale motivo si valuta un sistema tanto più efficiente quanto più e' in grado di avvicinarsi a tale requisito

Text Databases

- Due criteri di valutazione:
  - precisione (precision)
  - richiamo (recall)

Valutazione di una Interrogazione



Documenti contenuti nel database

Richiamo

- Il potere di richiamo è la percentuale di documenti rilevanti restituiti rispetto al totale di documenti rilevanti presenti nel sistema

$$\text{Richiamo} = \frac{\text{Rilevanti Restituiti}}{\text{Totale Rilevanti}}$$

Richiamo

- Il potere di richiamo ideale è uguale ad uno
- In generale il potere di richiamo sarà un valore inferiore ad uno perché il numero di documenti pertinenti restituiti è inferiore al numero di documenti pertinenti presenti nel sistema

Precisione

- La precisione è la percentuale di documenti rilevanti sul totale dei documenti restituiti

$$\text{Precisione} = \frac{\text{Rilevanti Restituiti}}{\text{Totale Restituiti}}$$

Precisione & Richiamo

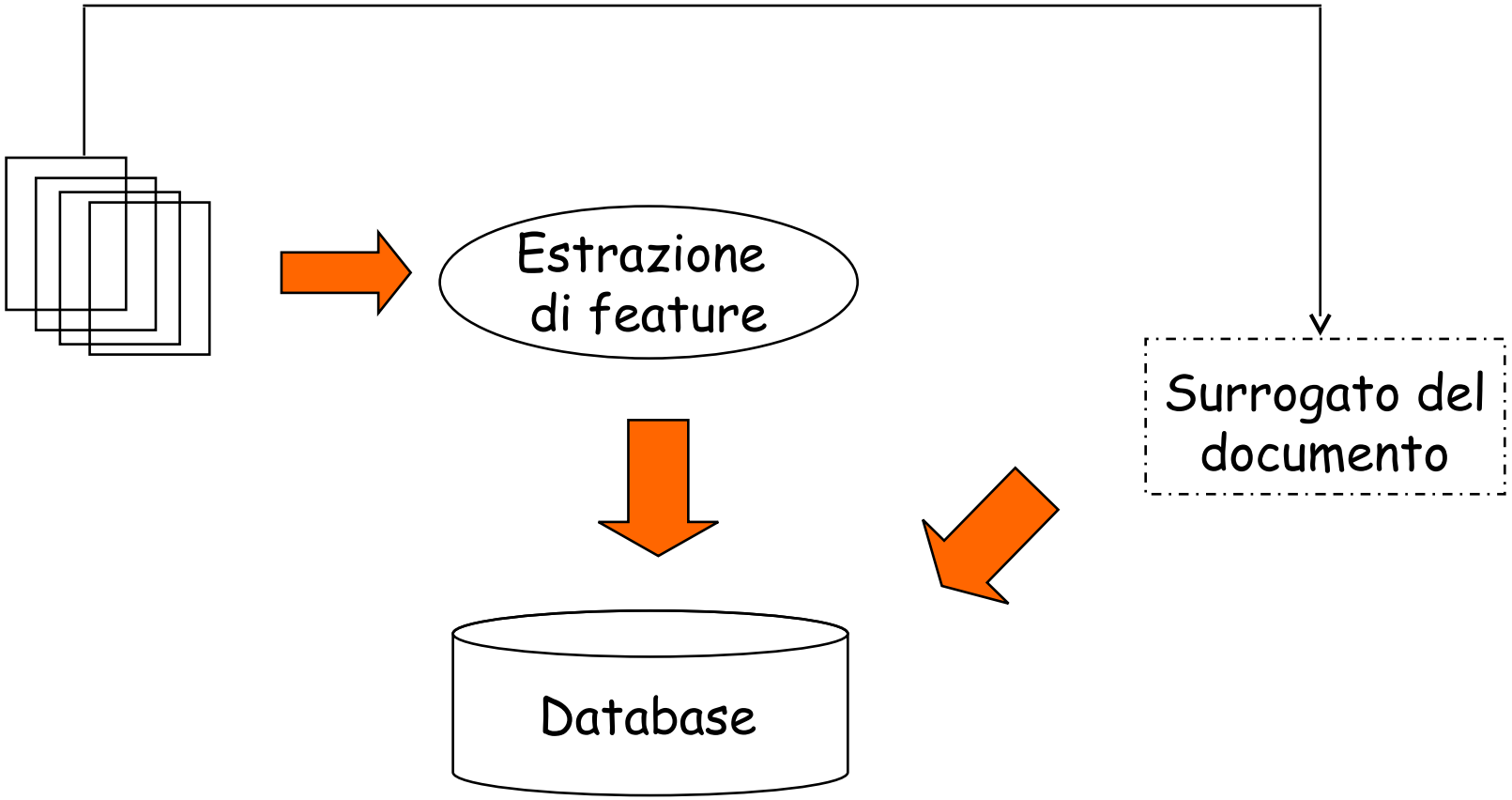
- La condizione ideale è avere il 100% di precisione e richiamo
- In generale, aumentando il numero di documenti restituiti si aumenta il potere di richiamo a spese della precisione

Text Databases

- Due problemi principali:
  - Sviluppo di tecniche efficienti per la rappresentazione dei documenti all'interno del sistema
  - Sviluppo di tecniche per la formulazione e l'esecuzione delle interrogazioni

Rappresentazione dei Documenti

Modellazione



Surrogato del Documento

- Un’insieme di dati *strutturati* che descrivono il documento
- Ogni documento dello stesso tipo sarà descritto dallo stesso surrogato
- Il surrogato del documento non descrive pienamente il contenuto informativo del documento ma costituisce una sua sintesi

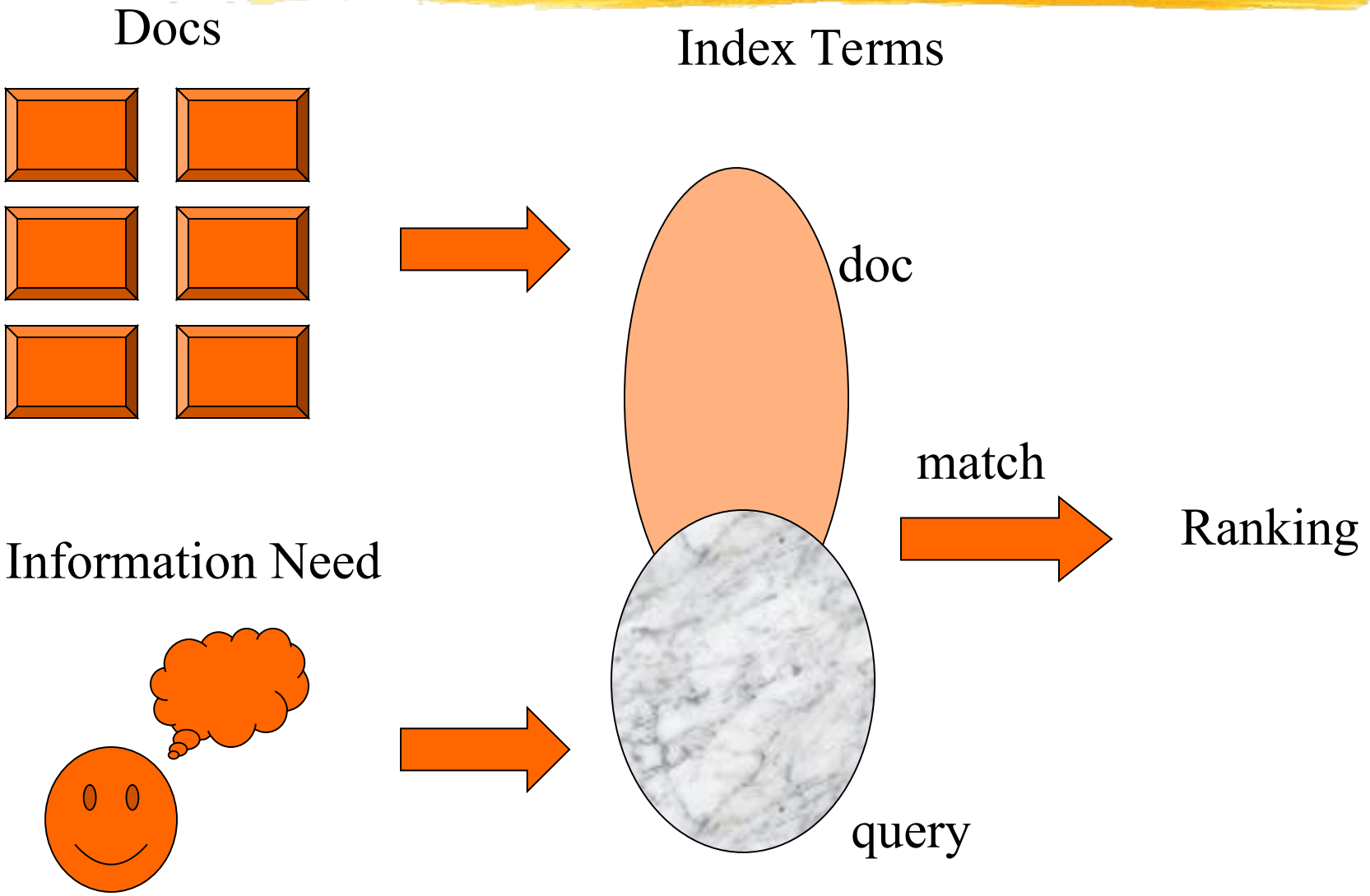
Surrogato del Documento

- Esempi:
  - scheda bibliografica che descrive un libro
  - scheda di descrizione di un oggetto d’arte
  - scheda di descrizione di un video

Estrazione di Feature

- Nel caso di documenti testuali le feature sono i termini utilizzati come indici
- Gli indici possono essere:
  - Una parola chiave o un insieme di parole chiave
  - Un insieme di concetti che caratterizzano il contenuto informativo del documento

Text Databases



Text Databases

- Vedere il testo come un insieme di parole chiave è limitativo
- Questo causa spesso insoddisfazione da parte dell'utente
- Il problema è ulteriormente complicato dal fatto che spesso gli utenti non sono in grado di formulare interrogazioni che riflettono i loro requisiti informativi

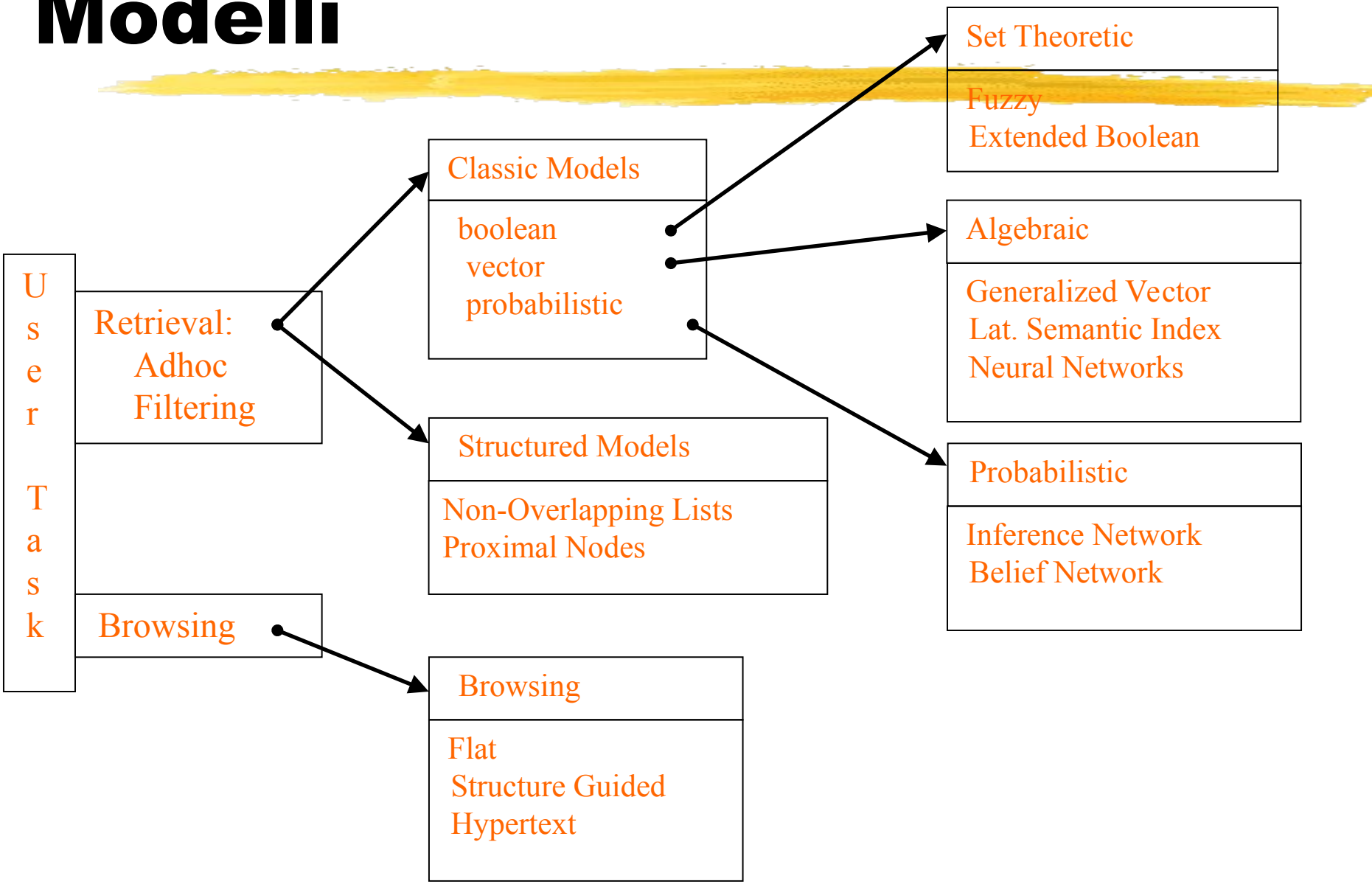
Text Databases

- Uno dei problemi più critici è quello di decidere i criteri di rilevanza di un documento rispetto ad una interrogazione: ranking

Ranking

- Il ranking è un ordinamento dei documenti restituiti da una interrogazione che riflette il grado di rilevanza dei documenti rispetto all'interrogazione
- I criteri per effettuare il ranking dipendono dal modello adottato per rappresentare i documenti

Modelli


User Task
Retrieval:
Adhoc
Filtering
Browsing
Classic Models
boolean
vector
probabilistic
Structured Models
Non-Overlapping Lists
Proximal Nodes
Browsing
Flat
Structure Guided
Hypertext
Set Theoretic
Fuzzy
Extended Boolean
Algebraic
Generalized Vector
Lat. Semantic Index
Neural Networks
Probabilistic
Inference Network
Belief Network

Modelli Classici – Concetti Base

Ogni documento è rappresentato da un insieme di termini indice rappresentativi
Un indice è una parola utile per ricordare l’argomento del documento
Solitamente gli indici sono dei nomi
I motori di ricerca assumono che tutte le parole nel testo siano indici (rappresentazione full text)

Modelli Classici – Concetti Base

Non tutti i termini che compaiono in un documento sono egualmente rappresentativi del suo contenuto informativo:

- di solito i termini troppo frequenti non sono buoni candidati per diventare indici

L'importanza di un indice è rappresentata da un peso ad esso associato

Modelli Classici – Concetti Base

- Sia
  - *ki* il termine indice
  - *dj* un documento
  - *wij* il peso associato a *ki* nel documento *dj*
- wij quantifica l'importanza dell'indice *ki* per descrivere il contenuto informativo del documento *dj*
- wij = 0 indica che *ki* non compare in *dj*

Modelli Classici – Concetti Base

- *vec(dj) = (w1j, w2j, …, wtj)* è il vettore di pesi associati al documento *dj*, dove *t* è il numero totale di indici
- *gi(vec(dj)) = wij* è una funzione che restituisce il peso di *ki* nel documento *dj*

Modelli Classici – Concetti Base

- Si assume che i pesi degli indici siano indipendenti
- Questa assunzione è una semplificazione perché esistono delle correlazioni tra termini che compaiono in un documento
- Questo facilita la definizione dei pesi ma rende meno precisa la ricerca:
  - Es: computer → network

Modello Booleano

E’ un modello semplice basato sulla teoria degli insiemi

Le interrogazioni sono espressioni booleane

- Semantica precisa
- Formalismo consolidato
- $q = ka \wedge (kb \vee \neg kc)$

Modello Booleano

- I pesi assumono valori binari:
  - $wij \varepsilon \{0,1\}$
- Un peso uguale a uno indica che il termine compare nel documento
- Un peso uguale a zero indica che il termine non compare nel documento

Modello Booleano

- Si supponga:
  - $q = ka \wedge (kb \vee \neg kc) = (ka \wedge kb) \vee (ka \wedge \neg kc)$
- La query può essere equivalentemente formulata come una disgiunzione di vettori della forma (*ka*,*kb*,*kc*)
  - $vec(qdnf) = (1,1,1) \vee (1,1,0) \vee (1,0,0)$
- I vettori vengono chiamati componenti congiuntive della query *q (qcc)*

Modello Booleano

$sim(q,dj) = 1$ se $\exists\, vec(qcc) \mid$
$vec(qcc) \in vec(qdnf)) \wedge$
$(\forall ki, gi(vec(dj)) = gi(vec(qcc)))$
0 altrimenti

Un documento viene restituito solo se la sua similitudine con l'interrogazione è pari ad uno

Modello Booleano

- Esempio:
  - *dj* = (0,1,0,.....) non è rilevante per *q* anche se contiene il termine *kb*
  - *di* = (1,1,0,....) è rilevante per *q*

Modello Booleano: Svantaggi

Nesuna nozione di matching parziale
Nessun meccanismo di ranking
I bisogni informativi di un utente devono essere tradotti in una espressione booleana
Le interrogazioni formulate dagli utenti sono spesso troppo approssimate

Modello Vettoriale

L’utilizzo di pesi binari è troppo limitante
Pesi non binari consentono di attuare matching parziali
I pesi sono usati per calcolare un *grado di similitudine* tra una interrogazione e ogni documento nel db
I documenti sono restituiti in ordine decrescente di similitudine

Modello Vettoriale

Sia i documenti che le interrogazioni sono rappresentate come dei vettori di pesi
Si definisce:

wij > 0 quando *ki* $\in$ *dj*

wiq >= 0 associato alla coppia *(ki,q)*

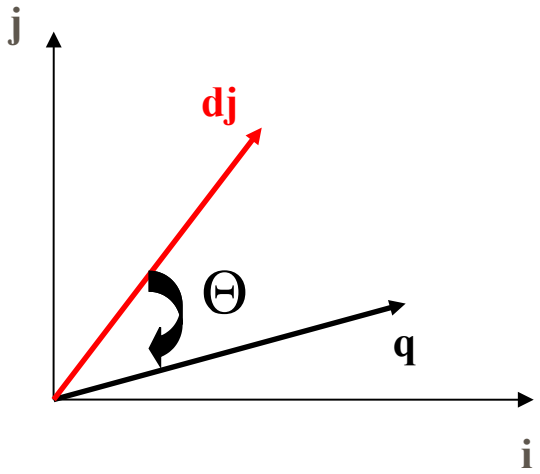
vec(dj) = (w1j, w2j, ..., wtj)

vec(q) = (w1q, w2q, ..., wtq)

Modello Vettoriale

- Nel modello vettoriale sia la query che il documento sono rappresentati come dei vettori in uno spazio t-dimensionale (dove t è il numero complessivo di termini indice)

Modello Vettoriale



Modello Vettoriale

- $sim(q,dj) = \cos(\Theta)$
  $= [vec(dj) \bullet vec(q)] / |dj| * |q|$
  $= [\Sigma\ wij * wiq] / |dj| * |q|$
- Dato che *wij > 0* and *wiq > 0:*
  - *0 <= sim(q,dj) <=1*
- Un documento è restituito anche se soddisfa solo parzialmente l'interrogazione

Modello Vettoriale

Quali strategie adottare per computare i pesi *wij* e *wiq* ?

Un buon peso deve tener conto di due fattori:

- Quanto un termine descrive il contenuto informativo di un documento
  - *Fattore tf*, la frequenza di un termine all'interno di un documento

Modello Vettoriale

- Quanto un termine compare all'interno di tutti i documenti nel db

  Fattore idf, l'inverso della frequenza all'interno di un documento

- $wij = tf(i,j) * idf(i)$

Modello Vettoriale

Sia:

N il numero totale di documenti nel db

ni il numero di documenti che contengono *ki*

freq(i,j) la frequenza di *ki* in *dj*

Il fattore *tf* normalizzato è:

$f(i,j) = freq(i,j) / max(freq(l,j))$

Dove il massimo è calcolato su tutti i termini che compaiono in *dj*

Modello Vettoriale

- Il fattore *idf* è dato da:
  - *idf(i) = log (N/ni).*

  il *log* è usato per rendere comparabili i valori di *tf* e *idf*.

Modello Vettoriale

Il metodo più utilizzato è quello di usare come peso:

$wij = f(i,j) * log(N/ni)$

Tale strategia è chiamata schema di pesatura *tf-idf*

Modello Vettoriale

Per i pesi da utilizzare nelle interrogazioni, Salton & Bucley propongono

- $wiq = (0.5 + [0.5 * freq(i,q) / max(freq(l,q)]) * log(N/ni)$

Modello Vettoriale

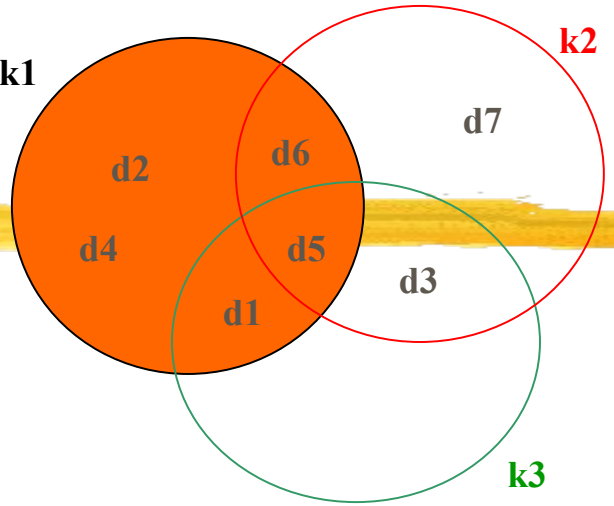
Vantaggi:

- I pesi migliorano la qualità delle risposte alle interrogazioni
- Possibilità di matching parziale
- La formula per il ranking ordina i documenti in base alla rilevanza che hanno per l’interrogazione

Svantaggi:

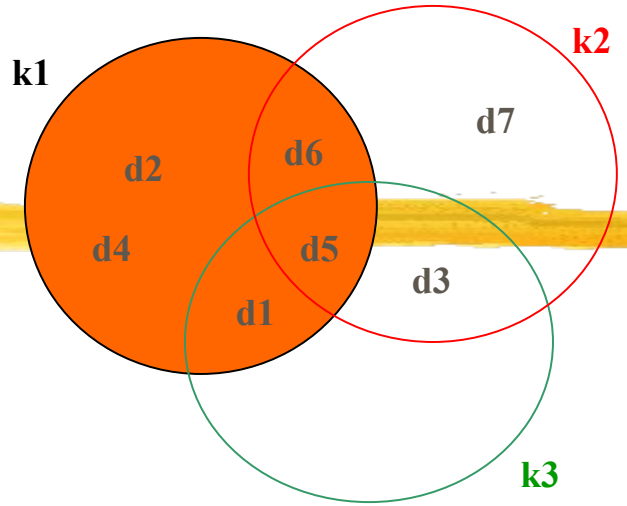
- Considera tutti gli indici come indipendenti

Esempio



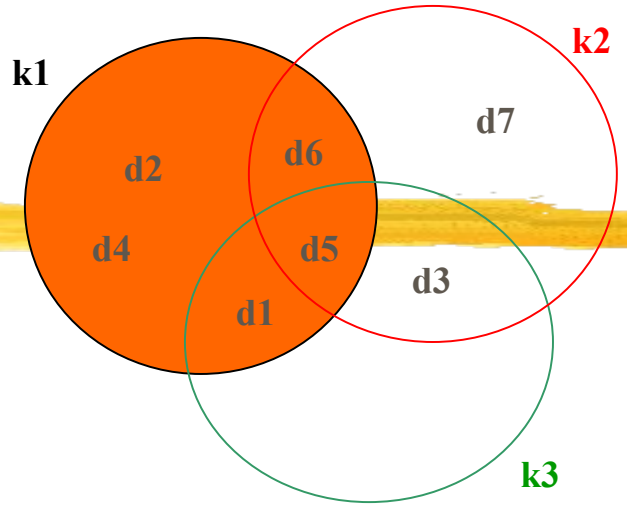
	k1	k2	k3	q • dj
d1	1	0	1	2
d2	1	0	0	1
d3	0	1	1	2
d4	1	0	0	1
d5	1	1	1	3
d6	1	1	0	2
d7	0	1	0	1
q	1	1	1	

Esempio



	k1	k2	k3	q • dj
d1	1	0	1	4
d2	1	0	0	1
d3	0	1	1	5
d4	1	0	0	1
d5	1	1	1	6
d6	1	1	0	3
d7	0	1	0	2
q	1	2	3	

Esempio



	k1	k2	k3	q • dj
d1	2	0	1	5
d2	1	0	0	1
d3	0	1	3	11
d4	2	0	0	2
d5	1	2	4	17
d6	1	2	0	5
d7	0	5	0	10
q	1	2	3	

Modello Probabilistico

- L’obiettivo è quello di utilizzare tecniche di calcolo delle probabilità
- Data una interrogazione esiste sempre un insieme di documenti che costituiscono la risposta ideale
- La specifica dell’interrogazione consiste nel definire le caratteristiche della risposta ideale

Modello Probabilistico

- Il problema è capire quali sono tali caratteristiche
- All'inizio viene effettuata una ipotesi su quali queste caratteristiche possono essere
- Tale ipotesi viene poi raffinata durante un processo di iterazione

Modello Probabilistico

Data una query *q* e un documento *dj*, il modello probabilistico cerca di stimare la probabilità che l'utente consideri il documento *dj* rilevante
Il modello assume che tale probabilità dipenda solo dalla interrogazione e dal modo in cui il documento è rappresentato

Modello Probabilistico

- L’insieme di documenti che costituiscono la risposta ideale (denotato da *R)* contiene i documenti che si ipotizza essere rilevanti
- Problemi:
  - Come computare la probabilità che un documento sia rilevante
  - Come determinare lo spazio degli eventi

Ranking

Ranking nel modello probabilistico:

sim(q,dj) =
P(dj rilevante per q) / P(dj non-rilevante per q)

Modello Probabilistico

- $wij \in \{0,1\}$
- P(R | vec(dj)) : probabilità che dj sia rilevante
- P(¬R | vec(dj)) : probabilità che *dj* non sia rilevante

Ranking

$$sim(dj,q) = P(R \mid vec(dj)) / P(\neg R \mid vec(dj))$$

$$= \frac{[P(vec(dj) \mid R) * P(R)]}{[P(vec(dj) \mid \neg R) * P(\neg R)]} \quad \text{(Bayes)}$$

$$\sim \frac{P(vec(dj) \mid R)}{P(vec(dj) \mid \neg R)}$$

Ranking

- P(R) probabilità che un documento selezionato a caso dal db sia rilevante
- P(vec(dj) | R) : probabilità di selezionare il documento dj dall'insieme R di documenti rilevanti

Ranking

$$sim(dj,q) \sim \frac{P(vec(dj) \mid R)}{P(vec(dj) \mid \neg R)}$$

$$\sim \frac{[\Pi\ P(ki \mid R)] * [\Pi\ P(\neg ki \mid R)}{[\Pi\ P(ki \mid \neg R)] * [\Pi\ P(\neg ki \mid \neg R)]}$$

P(ki | R) : probabilità che l'indice ki sia presente in un documento selezionato a caso dall'insieme R di documenti rilevanti

Ranking Iniziale

sim(dj,q) ~

$$\sim \Sigma wiq * wij * \left(\log \frac{P(ki \mid R)}{P(\neg ki \mid R)} + \log \frac{P(ki \mid \neg R)}{P(\neg ki \mid \neg R)}\right)$$

Come stimare P(ki | R) e P(ki | ¬R) ?

Attuando delle assunzioni:

P(ki | R) = 0.5

$$P(ki \mid \neg R) = \frac{ni}{N}$$

dove ni è il numero di documenti che contengono *ki*

Miglioramento Ranking Iniziale

$$sim(dj,q) \sim \sim \Sigma wiq * wij * \left(\log \frac{P(ki \mid R)}{P(\neg ki \mid R)} + \log \frac{P(ki \mid \neg R)}{P(\neg ki \mid \neg R)}\right)$$

Sia:

V : l'insieme dei documenti inizialmente recuperati

Vi : sottoinsieme di V contenente ki

Rivalutazione:

$$P(ki \mid R) = \frac{Vi}{V}$$

$$P(ki \mid \neg R) = \frac{ni - Vi}{N - V}$$

Modello Probabilistico

Vantaggi:

I documenti sono ordinati in ordine descrescente rispetto alla probabilità di rilevanza

Svantaggi:

E’ necessario fare una stima iniziale di P(ki | R)

Non si tengono in considerazione i fattori tf e idf

Modelli Classici: confronto

Il modello Booleano è il meno potente in quanto non consente il matching parziale
Risultati sperimentali indicano che il modello vettoriale ha prestazioni migliori del modello probabilistico

Estrazione di Feature dal Testo

- Quattro fasi:
  - Analisi lessicale del testo
  - Eliminazione delle “stopword”
  - Normalizzazione delle parole rimanenti
  - Selezione dei termini caratterizzanti (indici)

Analisi lessicale

- L’obiettivo è quello di trasformare il testo da una sequenza di caratteri ad una sequenza di parole
  - Eliminazione della punteggiatura
  - Conversione da maiuscolo a minuscolo

Eliminazione delle stopword

- Parole troppo frequenti nei documenti (ad esempio che compaiono in più dell'80% dei documenti) non sono utili per determinare il risultato di una interrogazione
- Esempi di stopword sono articoli, proposizioni e congiunzioni
- L'eliminazione delle stopword consente di ridurre notevolmente le dimensioni del documento originale

Eliminazione delle stopword

- L’eliminazione delle stopword può però ridurre il potere di richiamo:

to be or not to be

Normalizzazione

- Spesso l’utente specifica un termine in una query ma i documenti rilevanti ne contengono una sua variante
- La fase di normalizzazione sostituisce le varianti di una stessa parola con la loro radice comune (es: connesso, connettere, connessione,ecc.)
- Non esiste consenso comune sull’utilizzo della fase di normalizzazione

Selezione degli indici

- Due alternative:

  full-text index: tutte le parole sono utilizzate per caratterizzare il contenuto informativo del documento

  - selezione dei termini rilevanti:
    1. Mediante un processo automatico
    2. Tramite l’utilizzo di un Thesaurus

Selezione degli indici

- Un metodo consolidato è quello di identificare *gruppi di nomi* all'interno del testo (es. computer science)
- Un gruppo di nomi è un insieme di nomi la cui distanza nel testo non supera una soglia predefinita (es: 3)

Thesaurus

- Un thesaurus consiste di:
  - un insieme di vocaboli ed espressioni-chiave rilevanti per un particolare dominio
  - un insieme di sinonimi per ogni vocabolo nell'insieme
- I thesauri sono di solito definiti da esperti del settore

Thesaurus

- Esempio (dal thesaurus di Peter Roget):

  cowardly: aggettivo

  sinonimi: chicken (slang), chicken-hearted, craven, dastardly, faint-hearted, gutless, lily-livered, pusillanimous, unmanly, yellow (slang), yellow-bellied (slang)

Thesaurus

- La fase di indicizzazione e di ricerca avviene solo facendo riferimento ai termini del thesaurus
- L’utilizzo di thesauri è vantaggioso per domini in cui è possibile la standardizzazione dei termini di ricerca (esempio: ambito medico, legale,ecc.)

Thesauri con reti semantiche

- In un thesaurus che utilizza una rete semantica i termini possono essere strutturati attraverso una rete di collegamenti concettuali
- Una relazione semantica puo’ essere:
  - preferenziale
  - gerarchica
  - associativa

Relazione preferenziale

- La relazione preferenziale rappresenta l’equivalenza (sinonimia) tra termini:
  - SP (sinonimo preferenziale)
  - USA (è il contrario di SP)

- Allievo SP Alunno
- Alunno USA Allievo

Relazione gerarchica

- La relazione gerarchica rappresenta la relazione di specializzazione che esiste tra i termini
  - TL (termine largo)
  - TS (termine stretto)
- Veicolo TL Auto
- Auto TS Veicolo
- IL vertice della gerarchia si chiama TA (termine più ampio)

Relazione associativa

- La relazione associativa esprime un legame biunivoco tra le componenti lessicali
  - RT (termine in relazione)
- La tipologia di relazione dipende dal contesto

Relazione associativa

- Esempi:
  - antinomia: vittoria--sconfitta
  - concomitanza: sintomo--malattia
  - proprietà: trampolino--altezza
  - inclusione: contenuto--contenente
  - localizzazione: partita--stadio

Reti semantiche

- Un thesaurus con rete semantica rende più efficienti le interrogazioni perché consente di ricercare automaticamente i termini sinonimi, quelli più ampi o più ristretti ed i termini correlati

Thesaurus

- Per semplificare le ricerche al thesaurus è associato un *indice inverso* cioè una tabella che per ogni parola contenuta nel thesaurus contiene una lista dei documenti che la contengono

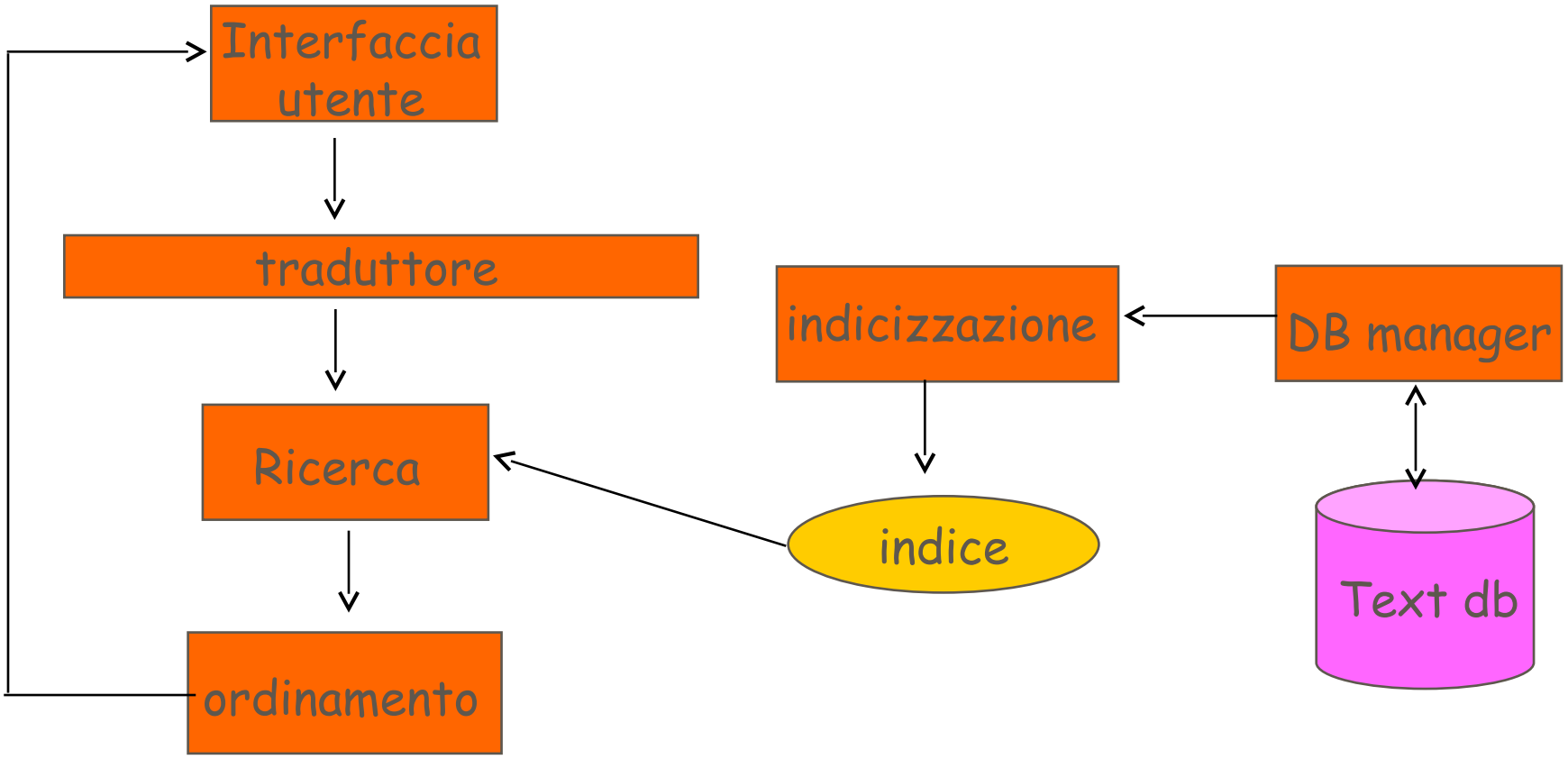
Indice inverso

termine	documenti
a	1,3,4,6,7
b	1,7
c	3,7
d	4,7

Indici inversi

- Solitamente gli indici inversi vengono estesi per mantenere per ogni termine non solo i documenti che lo contengono ma anche il numero di occorrenze di tale termine nei vari documenti

Architettura di riferimento



Interrogazioni

Interrogazioni

- Due tipi fondamentali:
  - su stringhe, con caratteri jolly e sottostringhe
  - con operatori booleani

Interrogazioni su stringhe

- E possibile ricercare tutti i testi che contengono una certa parola o una sua “variante”
  - comput? --> computa, computo,...
  - comp* --> computer, compilato, compito, ecc.

Interrogazioni su stringhe

- Varianti:
  - ricerca di frasi
  - ricerca per prossimità (insieme di parole e indicazione della distanza massima che devono avere all’interno del testo)

Interrogazioni booleane

- I termini dell'interrogazione possono essere composti mediante gli operatori booleani:
  - p1 AND p2
  - p1 OR p2
  - NOT p1
- Gli operatori booleani possono essere combinati:
  - (p1 AND p2) OR (p3 AND p4)

Interrogazioni booleane

- Nei sistemi booleani classici le query non sono approssimate
- Nella pratica si usa una versione “fuzzy” degli operatori

Sistemi di IR: esempi

- DIALOG Corporation: offre più di 500 sistemi di IR su svariati argomenti, quali scienze, medicina, economia e giornali elettronici (www.dialog.com)
- LEXIS-NEXIS: ambito legale ed economico (www.lexis-nexis.com)
- OCLC (the online computer library center) offre l'accesso ad 1.5 milioni di articoli (www.oclc.org)

Sistemi di IR: esempi

- H.W. Wilson: offre più di 40 sistemi di IR per scuole ed istituzioni pubbliche (www.hwwilson.com)

Sistemi di IR: esempi

- CA SEARCH: Chemical Abstract. Contiene 14 milioni di documenti con una frequenza di aggiornamento di 11.000 documenti alla settimana
- MEDLINE: indicizza articoli provenienti da 3.700 riviste mediche
- NewYork Times -- Fulltext: contiene tutte le edizioni del NewYork Times dal 1981 ad oggi

Sistemi di IR: esempi

- PsycINFO: Psycological Abstract. Contiene 1.5 milioni di documenti riguardanti psicologia, sociologia, psichiatria, linguistica ed antroplogia dal 1887 fino ad oggi

Motori di ricerca per il WEB

Motori di ricerca

Motore di ricerca	url	numero di pag. (mil.)
Altavista	www.altavista.cm	140
AOL Netfind	www.aol.com/netfind	-
Excite	www.excite.com	55
Google	google.stanford.edu	25
GoTo	goto.com	-
HotBot	www.hotbot.com	110
InfoSeek	www.infoseek.com	30
Lycos	www.lycos.com	30
Magellan	www.mckinley.com	55
Microsoft	search.msn.com	-
NorthernLight	www.lnsearch.com	67
Web Crawler	www.webcrawler.com	2

Web directory

Web directory	url	siti web (migl.)
eBlast	www.eblast.com	125
LookSmart	www.looksmart.com	300
Lycos Subjects	a2z.lycos.com	50
NewHoo	www.newhoo.com	100
Netscape	www.netscape.com	-
Search.com	www.search.com	-
Snap	www.snap.com	-
Yahoo!	www.yahoo.com	750

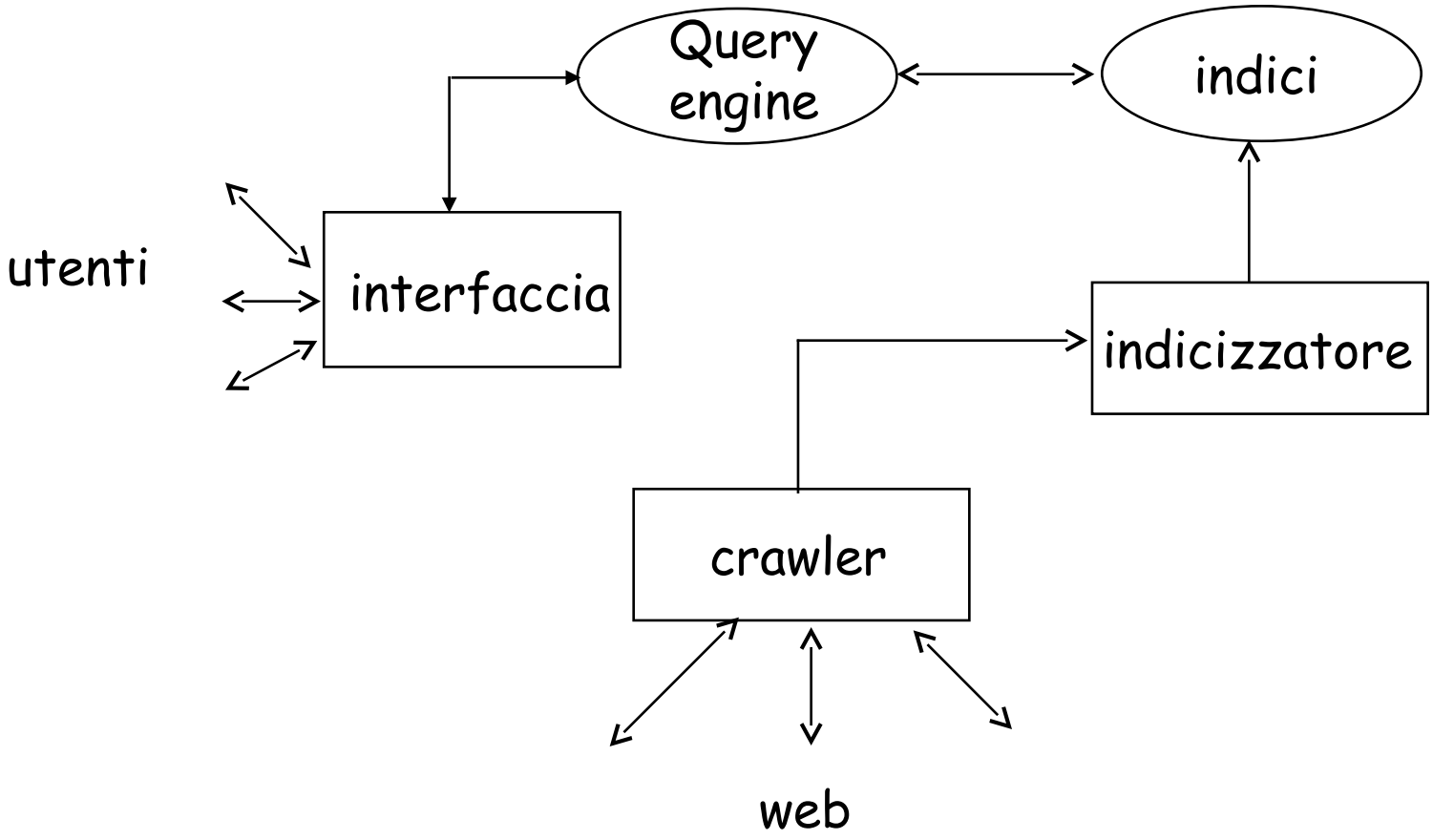
Motori di ricerca

- I più grandi motori di ricerca sono (1998): AltaVista, Yahoo!, HotBot, Northern Light, e Excite
- Tali motori di ricerca coprono circa il 30% del totale delle pagine web

Motori di ricerca

- Due architetture principali:
  - centralizzata
  - decentralizzata

Architettura centralizzata



Architettura centralizzata

- I crawler (robot, spider, walker, knowbot) sono programmi che attraversano il web mandando pagine web nuove o modificate al server per essere indicizzate

Architettura centralizzata

- AltaVista utilizza una architettura centralizzata
- Nel 1998, AltaVista utilizzava 20 macchine multi-processore, ognuna delle quali era dotata di 130 Gb di RAM e 500 Gb di disco
- Il query engine utilizzava più del 70% del totale delle risorse

Indicizzazione

- La maggior parte dei motori di ricerca usa una indicizzazione full-text con una variante degli indici inversi:
  - per ogni parola viene mantenuta una lista degli url dei documenti che la contengono
- Alcuni motori di ricerca effettuano l’eliminazione delle stopword prima dell’indicizzazione

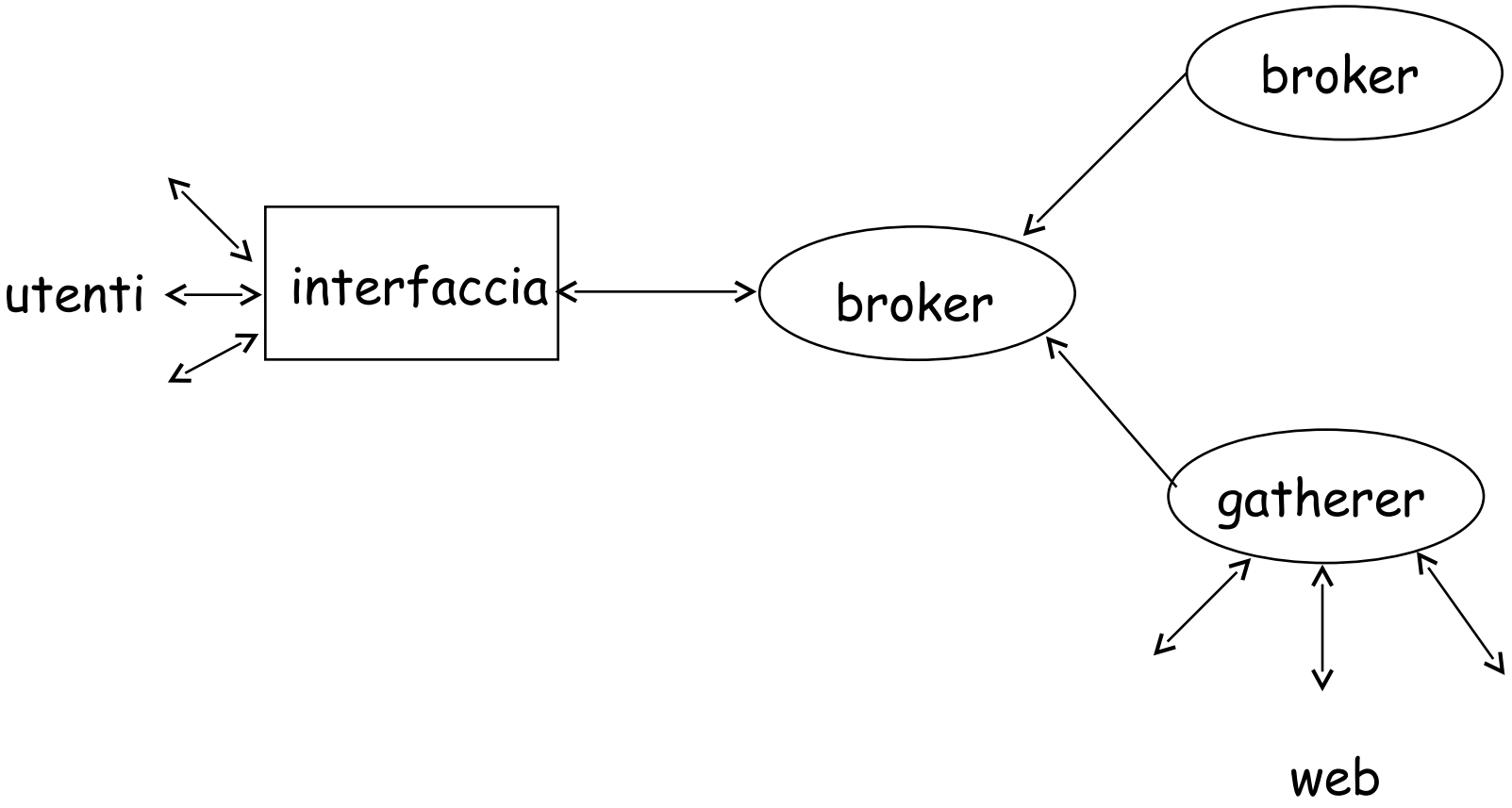
Indicizzazione

- In aggiunta, per ogni pagina indicizzata viene mantenuta una breve descrizione contenente la data di creazione, la dimensione, il titolo e poche righe della pagina stessa
- Se si ipotizza che l'url e la descrizione della pagina richiedano 500 byte di spazio, l' indicizzazione di 100 milioni di pagine richiede 50 Gb di memoria

Architettura centralizzata

- Il principale svantaggio dell'architettura centralizzata è che il meccanismo di indicizzazione risiede su un solo server e può diventare un collo di bottiglia per l'intero sistema

Architettura decentralizzata



Metasearcher

- I metasearcher sono server web che inviano una interrogazione a più motori di ricerca
- Il principale vantaggio è che l'utente può interrogare più motori di ricerca tramite un'unica interfaccia

Metasearcher

Metasearcher	url	n.sorgenti
Cyber411	www.cyber411.com	14
Dogpile	www.dogpile.com	25
Highway61	www.highway61.com	5
Inference Find	www.infind.com	6
Mamma	www.mamma.com	7
Metacrawler	www.metacrawler.com	7
MetaFind	www.metafind.com	7
MetaMiner	www.miner.uol.com.br	13
MetaSearch	www.metasearch.com	-
SavvySearch	www.cs.colostate.edu:2000	13

Metasearcher

- I metaseracher si differenziano per:
  - il metodo con cui effettuano il ranking
  - la precisione con cui traducono la query nel linguaggio di query dei vari motori di ricerca che interrogano

Metasearcher

- L’utilizzo dei metasearcher è giustificato dal fatto che solo una piccola parte delle pagine web sono indicizzate da più di un motore di ricerca
- Ad esempio, la percentuale di pagine a comune tra Altavista, HotBot, Excite e Infoseek è inferiore all’1%

Gestione di testi in Oracle*8i*

- Oracle*8i inter*Media Text (include ConText Cartridge)
- permette di indicizzare testi e documenti memorizzati in Oracle8i, in file del sistema operativo o URL, in maniera integrata con i dati relazionali tradizionali con possibilità di interrogazioni basate sul contenuto

Gestione di testi in Oracle*8i*
Esempio

```
create table docs
   (id number primary key,
      text varchar2(80));
insert into docs values (1,'first  document');
insert into docs values (2, 'second
document');
commit;

create index doc_index on docs(text)
  indextype is ctxsys.context;
select id from docs
   where contains(text, 'first') > 0;
```

Gestione di testi in Oracle*8i*

- Indicizzazione di testi
  - sistema di preferenze
    - datastores
    - filters
    - section groups
    - lexers
  - DML
  - ottimizzazione
- Interrogazione di testi
  - operatori
  - thesaurus

Gestione di testi in Oracle*8i*

- Indicizzazione di testi
  - sistema di preferenze
    - datastores
    - filters
    - section groups
    - lexers
  - DML
  - ottimizzazione
- Interrogazione di testi
  - operatori
  - thesaurus

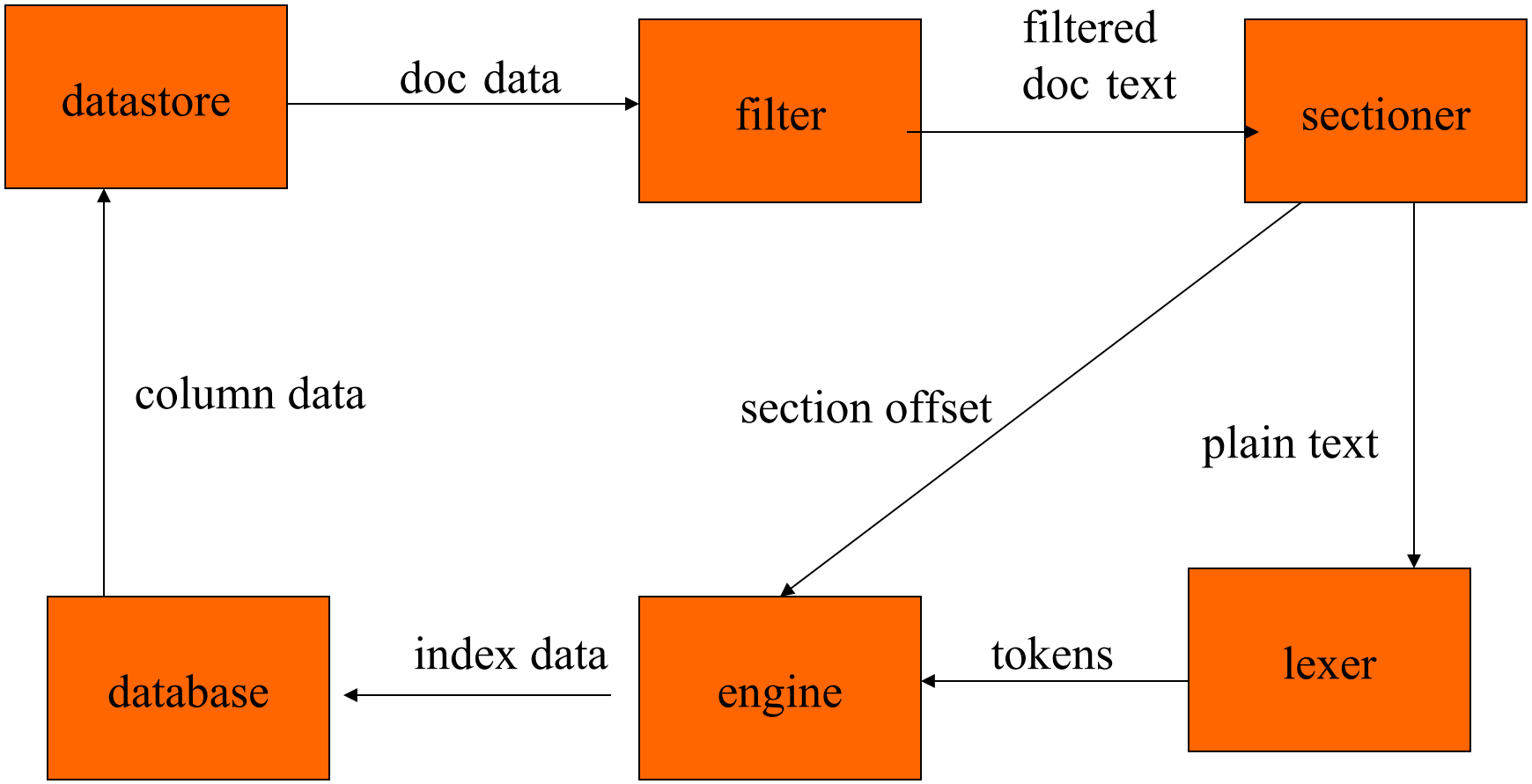
Indicizzazione di testi

- non è possibile eseguire query di tipo contains senza aver prima definito un indice Text
- comando per la creazione di indici

  create index INDEXNAME on TABLE(COLUMN)

  indextype is ctxsys.context
- la colonna indicizzata deve essere di tipo CHAR, VARCHAR, VARCHAR2, LONG, LONG RAW, BLOB, CLOB, BFILE
- la tabella deve avere una chiave primaria, utilizzata per identificare i documenti

Indicizzazione di testi



Indicizzazione di testi

- **Datastore**: scorre le righe della tabella e legge i dati della colonna, restituendo i dati contenuti nel documento
- **Filter**: prende i document data e li trasforma in una rappresentazione testuale (non necessa-riamente plain text, può essere XML o HTML)
- **Sectioner**: converte l'output del filter in plain text (dipende dal formato del testo)
- **Lexer**: suddivide il plain text in token discreti (dipende dal linguaggio)

Indicizzazione di testi

- a partire dai token, dai section offset, e dalla stoplist Oracle8i costruisce un inverted index
- l'inverted index contiene i token e i documenti in cui appaiono (docid, non primary key)
- esempio:

```
(1, 'first document')
(2, 'second document');
```

```
DOCUMENT  ---> doc 1 position 2,
  doc 2 position 2
FIRST     ---> doc 1 position 1
SECOND    ---> doc 2 position 2
```

Sistema di preferences

- permette di personalizzare l’indicizzazione
- è organizzato in classi:
  - DATASTORE trasforma column data in document data
  - FILTER trasforma document data in formatted text
  - LEXER divide il plain document text in token
  - WORDLIST contiene i default per l’espansione delle query
  - STOPLIST contiene la lista delle stopword
  - STORAGE parametri di memorizzazione della index table

Sistema di preferences

- per ognuna di queste classi ci sono uno o più oggetti che rappresentano i possibili comportamenti
- tali oggetti non possono essere usati direttamente negli indici:
  - si crea una preference a partire da tali oggett template
  - la si personalizza settandone gli attributi
  - si usa tale preference per creare l’indice

Sistema di preferences

- esempio

```
ctx_ddl.create_preference('mypref',
                          'FILE_DATASTORE');

ctx_ddl.set_attribute('mypref',
                      'PATH', '/docs');

create index doc_index on docs(text)
     indextype is ctxsys.context
     parameters ('datastore mypref');
```

Sistema di preferences

- nella clausola parameters si possono specificare più classi, semplicemente aggiungendo una coppia keyword-preference

```
parameters('datastore mypref
            filter myfilter')
```

- questo meccanismo è usato per tutte le classi tranne SECTION GROUP e STOPLIST, che hanno una API propria
- esistono dei parametri utilizzati dal sistema di default (quelli per LEXER, WORDLIST e STOPLIST dipendono dal linguaggio)

Datastore

- DIRECT_DATASTORE assume che il documento sia memorizzato direttamente nella colonna indicizzata ne restituisce semplicemente il contenuto
- DETAIL_DATASTORE concatena più righe di una detail table per costruire il document data
- FILE_DATASTORE interpreta la colonna come un nome di file, lo apre e ne restituisce il contenuto
- URL_DATASTORE interpreta la colonna come un URL, effettua una GET e restituisce il contenuto
- USER_DATASTORE invoca una stored procedure per sintetizzare il document data

Filters

- NULL_FILTER utilizzato quando il contenuto del documento non è in formato binario, passa semplicemente il testo dal datastore al sectioner
- CHARSET_FILTER converte i documenti da un insieme di caratteri straniero al character set del database
- USER_FILTER filtering ad hoc (es. per convertire tutto in lettere maiuscole)
- INSO_FILTER riconosce automaticamente e filtra oltre un centinaio di formati diversi, tra cui Word e Acrobat, producendo HTML come output

Section Groups

- la classe section group prende un formato di testo (es. XML o HTML) come input e restituisce i section boundaries e plain text
- i section group non sono creati con create_preferences, ma con una API separata

```
ctx_ddl.create_section_group('mygroup',
                             'html_section_group');
```

- il primo argomento è il nome del section group, il secondo è il tipo, che specifica il formato del testo di input e le regole per individuare le sezioni

Section Groups

- tipi di section groups:
  - NULL_SECTION_GROUP: passa semplicemente il testo al lexer senza estrarre nessuna informazione
  - BASIC_SECTION_GROUP: per semplice XML senza DTD, rimuove semplicemente i markup tags
  - HTML_SECTION_GROUP: per HTML, rimuove commenti e i contenuti di SCRIPT e STYLE
  - XML_SECTION_GROUP: per XML (con DTD), gestisce entità
  - NEWS_SECTION_GROUP: per newsgroup style postings, rimuove header lines

Section Groups

- le sezioni hanno tre attributi:
  - TAG: specifica come riconoscere la sezione
  - NAME: specifica come riferirsi alla sezione nelle query (più tag possono essere mappati sullo stesso nome)
  - TYPE: è il tipo di sezione, ci sono tre tipi diversi: ZONE, SPECIAL, FIELD
- ZONE:
  - si registra dove sono start e end
  - permette query di tipo WITHIN
  - se una sezione ZONE si ripete viene trattata separatamente nelle query
  - possono essere annidate

Section Groups

- SPECIAL: non sono riconosciute attraverso tag, ma dal lexer attraverso la punteggiatura
  - SENTENCE
  - PARAGRAPH
- FIELD:
  - il contenuto della sezione è indicizzato separatamente dal resto del documento
  - le query di tipo WITHIN vengono eseguite su questo indice separato
  - sono pensate per sezioni non ripetute e non overlapping

Section Groups

- <A>rat</A><A>ox</A> <B>tiger rabbit</B> <C>dragon<C>snake</C></C>
- ctx_ddl.create_section_group('mygroup', 'basic_section_group');
- sezione ZONE

```
ctx_ddl.add_zone_section('mygroup', 'asec', 'a');
ctx_ddl.add_zone_section('mygroup', 'bsec', 'b');
ctx_ddl.add_zone_section('mygroup', 'csec', 'c');
```

- contains(text, 'rat within asec') > 0

  trova il documento
- contains(text, 'tiger within asec') > 0

  no

Section Groups

- ogni istanza è considerata distinta:

```
contains(text, '(tiger and rabbit) within bsec') > 0
```

trova il documento

```
contains(text, '(rat and ox) within asec') > 0
```

no

```
contains(text, '(dragon and snake) within csec') > 0
```

sì

- nel caso di sezioni FIELD il contenuto di diverse istanze viene unito, quindi la seconda query restituirebbe il documento

Lexers

- BASIC_LEXER: per la maggioranza delle lingue europee, si può modificare il comportamento di default attraverso gli attributi
  - JOINS per specificare caratteri non alfanumerici da trattare come lettere valide
  - PUNCTUATION per specificare i simboli di punteggiatura (importanti per sezioni SENTENCE e PARAGRAPH)
  - per la normalizzazione del testo (es. accenti, maiuscole/minuscole, parole composte)
  - per selezionare indicizzazione di testo o tematica
- JAPANESE_VGRAM_LEXER, CHINESE_VGRAM_LEXER, CHINESE_LEXER, KOREAN_LEXER

Stoplist

- lista delle stopword, che non vengono considerate per l'indicizzazione
- API separata:

```
ctx_ddl.create_stoplist('mylist');
ctx_ddl.add_stopword('mylist', 'the');
```

- stoplist di default language-specific
- si possono aggiungere stopword all'indice senza doverlo ridefinire

```
alter index myidx rebuild parameters ('add
stopword AND');
```

- stop classes (es. NUMBERS) e stop themes

Wordlist

- non ha effetto sull'indicizzazione, ma contiene i setting per espansione stem e fuzzy dei termini utilizzati per rispondere alle interrogazioni
- un solo oggetto BASIC_WORLDLIST con attributi:
  - STEMMER (espansione di una parola a forme differenti, es. ENGLISH, ITALIAN, NULL)
  - FUZZY_MATCH (considera parole mistyped, es. varie lingue e OCR)
  - FUZZY_SCORE (score floor per espansione fuzzy)
  - FUZZY_NUMRESULT (max. numero parole per espansione fuzzy)

Storage

- Permette di settare i parametri di memorizzazione per le 5 tabelle indice
  - I_TABLE (principale)
  - K_TABLE (conversione rowid docid)
  - R_TABLE (conversione docid rowid)
  - N_TABLE (docid invalidi)
  - I_INDEX (indice sulla I_TABLE)
- MEMORY per settare la memoria totale destinata all’indice

Aggiornamento degli indici

- in genere gli indici non vengono aggiornati dopo ogni singola operazione sui documenti
  - indicizzare un singolo documento richiede molto tempo
  - gli inverted index si aggoirnano meglio su un insieme di documenti per volta
  - le applicazioni in genere sono abbastanza statiche, e non è necessaria una completa consistenza
- invalidazione sincrona (marcando il documento) e aggiunta asincrona (coda dr$pending)
- aggiunte gestite su invocazione esplicita (sync) o in background

Ottimizzazione degli indici

- due problemi:
  - frammentazione (a ogni aggiornamento viene aggiunta una nuova riga)

```
DOG    DOC 1 DOC 3 DOC 5
DOG    DOC 7
DOG    DOC 9
DOG    DOC 11
```

  - invalidazione di documenti
- due ottimizzazioni: FAST (solo frammentazione) e FULL (si può specificare maxtime)

Interrogazioni

- query contains:

```
select id
from texttab
where contains(textcol,'query') > 0
```

  - il primo argomento è il nome della colonna, il secondo è il testo dell'interrogazione (max 2000 byte)
  - restituisce un numero, che quantifica il match (0 = no match)
  - per effettuare contains queries è necessario un indice Text

Scoring

```
select id, score(1)
   from texttab
   where contains(textcol, 'query', 1) > 0
   order by score(1) desc
```

- 1 è contains label e può essere un qualsiasi numero, utilizzato per matchare lo score nella select list con quello nella clausola where
- lo score è un numero compreso tra 0 e 100, ed è relativo (significativo solo per la query)
- $3f(1+\log(N/n))$ con f frequenza, N numero totale righe, n numero di righe che contengono il termine

Interrogazioni

- interrogazioni semplici

```
contains(text, 'dog') > 0
contains(text, '{dog}') > 0
```

- si possono interrogare frasi:

```
contains(text,'dog my cat') > 0
```

- le stopword sono trattate come wildcard (matchano qualsiasi parola)

```
contains(text, 'dog the cat') > 0
```

- le stopword da sole vengono eliminate dalla query

Operatori booleani

- AND (&) e OR (|): restituiscono punteggi numerici invece che valori booleani
- AND è il minimo dei punteggi dei suoi operandi, OR il massimo
- NOT (~) è "AND NOT” (differenza):
  - 'dog NOT cat' restituisce i documenti che contengono "dog" ma non contendono "cat”
- il punteggio restituito è quello del figlio sinistro

Operatori di score

- WEIGHT (*) moltiplica il punteggio di un termine di ricerca per renderlo più o meno importante nella query (peso tra .1 e 10)

```
contains(text,'(dog*2) AND cat') > 0
```

- THRESHOLD (>) elimina i documenti sotto una certa soglia
- MINUS (-) sottrae il punteggio dell'operando destro a quello del sinistro
- ACCUM (,) raggruppa più parole o frasi e ne accumula i punteggi

Operatori di espansione delle parole

- WILDCARD (%_) per pattern matching (come in LIKE di SQL)
- FUZZY (?) trova parole simili (usa wordlist)
- STEM ($) trova parole con radice comune (usa wordlist)
- SOUNDEX (!) trova parole con stesso suono (usa una specifica espansione fuzzy)
- EQUIV (=) permette di indicare esplicitamente varie forme della stessa parola

Operatori di prossimità

- operatore NEAR, che ha due forme
- `dog ; cat ; boat`

  il punteggio dipende da quanto i termini sono vicini l'uno all'altro
- NEAR((dog,boat), 10, TRUE)
  - primo argomento è lista di parole
  - secondo è distanza massima
  - terzo specifica se tenere conto dell'ordine in cui appaiono nella lista

Altri operatori

- WITHIN: limita una query a una particolare sezione
- ABOUT: se l'indice ha una componente tematica effettua una query tematica, altrimenti l'interrogazione viene espansa per aumentare il recall
  - `contains(text, 'about(canines)')`

  può restituire documenti che contengono "dog"
  - `about(go home now)` è trasformata in

    `$go,$home,$now`

Altri operatori

- operatori di Thesaurus: corrispondono alle relazioni del Thesaurus e permettono di espandere una parola in suoi sinonimi
  - si basano su un Thesaurus che deve essere stato caricato, di default non viene installato alcun Thesaurus
- SQE (Stored Query Expressions) permette di dare un nome alle interrogazioni e poi effettua macro espansione

Thesaurus

- file con formattazione particolare
- relazioni considerate
  - SYN, UF: sinonimi
  - PT, USE, SEE: preferred term
  - BT: broader term (BTn per gerarchia)
    - BTP (parte), BTG (generico), BTI (instance)
  - analoghi con NT
  - RT: related term

Thesaurus - operatori

- SYN(term [, thesname])
- PT(term [, thesname])
- BT(term [, level [,thesname]])
  BTP/BTG/BTI(term [, level [,thesname]])
  NT(term [, level [,thesname]])
  NTP/NTG/NTI(term [, level [,thesname]])
- TT(term [, thesname])
- RT(term [,thesname])
- TR(term [, lang [, thesname]])
- TRSYN(term [, lang [, thesname]])

Query Tuning e Feedback

- possibilità di esaminare i piani di esecuzione e di avere feedback dall'esecuzione della query (per riformularla meglio)
- explain permette di vedere rappresentazione interna dell'interrogazione (come è stata espansa)
- hierarchical query feedback suggerisce altri termini che si possono voler provare (suggested broader terms, related terms, narrower terms)

Altre funzionalità

- highlighting: per evidenziare i termini specificati nella query nei documenti restituiti
- linguistic extraction, per estrazione automatica dei temi (solo per l'inglese)
- utilizza una knowledge base: termini e frasi organizzati in una gerarchia di categorie, che può essere estesa