Information retrieval

L'information retrieval si distingue dal data retrieval.
Il data retrieval ha a che fare con dati che hanno una struttura e una semantica definite. Il retrieve consiste nel cercare tutti gli oggetti che soddisfano esattamente delle condizioni definite. L'information retrieval invece lavora con il linguaggio naturale di testi non strutturati e semanticamente ambigui. Lo scopo principale dell'information retrieval è quella di ritornare risultati che sono rilevanti per la query effettuata e pochi risultati non rilevanti. Per fare ciò è necessario prima di tutto avere un linguaggio di query. Esistono vari tipi:

- Key-word based -> sequenze di parole divise da dei separatori
    - Multiple-word query
    - Phrase query -> ricerca di documenti che hanno una sequenza data di parole (frase)
    - Proximity query -> ricerca di una frase con una distanza massima tra le parole della query
- Boolean query -> singole o multiple parole o pattern concatenate da operatori logici
- Pattern matching -> ritorna risultati che hanno un match approssimativo del pattern cercato
    - Ricerca per
        - Prefissi
        - Suffissi
        - Sotto-stringhe
        - Range -> insieme di stringhe alfabeticamente comprese tra due parole
- Structural query -> assume che i documenti in cui si cerca abbiano una struttura
- Concept-based query -> usa un vocabolario di parole controllate e permette all'utente di comporre una query utilizzando questo vocabolari per stringere o allargare la specificità della query. Utile in campi specifici (PUBMED).

Le query possono poi essere processate in due modi diversi;

- Online query processing -> testo volatile e corto e la ricerca è fatta client-side
- Store first- query later -> quando il testo in cui cercare è grande

Un query matching model è un modello che data una query Q e un testo T dice se T è rilevante per Q e quando lo è.

Text operation

Document Processing

Il preprocessing dei documenti è una procedura che trasforma un document in un insieme di termini. Alcune operazioni che possono essere compiute in questa fase sono:

- Analisi lessicale -> gestione cifre, segni di punteggiatura e maiuscole/minuscole
    - Divisione del testo in token -> insieme di caratteri con un significato
- Eliminazione delle stopword -> rimuove le parole inutili
    - Stop-word -> insieme di parole usate di frequente (articoli, preposizioni, etc)
    - Riduce il numero degli index term
- Stemming -> gestione delle variazioni sintattiche delle parole
    - Plurali, coniugazioni di verbi
    - Riduce il numero degli index term
    - Riduce il numero di parole che esprimono un concetto comune e che iniziano con la stessa radice (stem)
    - Alcuni metodi di stemming:
        - Table lookup -> crea una tabella di tutti gli index term e dei loro stem. Veloce e semplice ma richiede un spazio di archiviazione elevato
        - Successors variety -> il successor variety di una stringa è il numero di caratteri diversi che la seguono in un insieme di parole. In un testo il successor variety di una stringa tende a diminuire all'aggiungersi di un carattere fino a raggiungere un segmento limite che è lo stem.
        - Affix removal -> rimuove suffissi e prefissi dai termini per trovare uno stem. Basato su un insieme di regole che dipendono dalle regole grammaticali della lingua. Il più famoso stemmer è porter
    - Di solito lo stemming influenza positivamente le performance di retrieval, ma può introdurre anche degli errori dovuti all'aver rimosso troppo o troppo poco dalle parole. La vera efficacia dello stemming dipende dalla natura del vocabolario

- Selezione index term -> selezionare i termini da utilizzare come index term, ovvero termini la cui semantica rappresenta il documento
    - Selezione manuale -> fatta da esperti
    - Selezione automatica -> utilizzo principalmente di nomi o si usano dei parser o taggers
        - Un parser sintattico è uno strumento che analizza ogni frase e identifica le sue part, le etichetta e gli assegna una classe semantica e funzionale. L'approccio migliore per fare ciò è quello statistico che si basa sull'utilizzo di Treebank, ovvero di testi di esempio già analizzati
        - Un tagger assegna ad ogni parola solo la part-of-speech

Thesauri

Un thesauro è una lista di parole importanti per un certo dominio e ad ogni parola è associata un insieme di parole correlate (sinonimi, contrari, derivazioni). Un thesauro può essere usato per

- Indexing -> normalizzazione degli index term e riduzione del rumore
- Ricerca -> per assistere l'utente con la formulazione della query e per restringere o allargare le richieste della query

I thesauri sono utili in applicazioni che lavorano in un dominio specifico (PUBMED). Un thesauro può essere realizzato manualmente o in automatico unendo più thesauri o utilizzando una collezione di documenti

Word similarities

La misura di similarità di due parole può essere basata su

- Path -> distanza nella gerarchia degli iperonimi
    - Due parole sono simili se sono vicine nella gerarchia del thesauro.
        - Path-distance similarity -> sim(c1,c2) = 1 / (shortest_path(c1,c2) + 1)
        - Wu-palmer similarity -> sim(c1,c2) = 2 * depth(LCS(c1,c2)) / (depth(c1) + depth(c2)) dove LCS(c1,c2) è il primo nodo padre in comune ad entrambe le parole
- Contenuto informativo
    - Il contenuto informativo di un termine può essere valutato come IC(c) = -logP(c) dove P(c) è la probabilità che una parola casuale è istanza del concetto di c
    - Resnik similarity -> sim(c1,c2) = -logP(LCS(c1, c2))

Word sense disambiguation

Il WSD è l'associazione di una parola con una definizione o significato. Il WSD si compone di due fasi

- Determinare tutti i possibili significati di una parola -> utilizzo di un dizionario o di un thesauro
- Assegnare ad ogni occorrenza della parola il significato corretto -> un approccio è quello di capire il contest dai termini vicini alla parola considerata
    - Esempio wordnet -> il WSD per i nomi utilizza come contesto l'insieme di altri nomi che appaiono nella frase considerata. Dato un nome N, per ogni significato Sn di N viene calcolata la confidenza CSn nel scegliere Sn come senso di N. Si sceglie come senso quello con la confidenza maggiore. La confidenza CSn è influenzata da:
        - La similarità di Sn con tutti i sensi dei nomi nel contesto basata su
            - La distanza nella gerarchia delle ipernimie
            - La distanza delle parole coinvolte
        - La frequenza di quel senso

Full-text indexing

La ricerca sequenziale o online di un pattern all'interno di un testo non ancora preprocessato è fattibile solo se il testo è piccolo. In caso invece il testo sia molto grande bisogna ricorrere all'utilizzo di indici, ovvero di strutture dati che servono a velocizzare la ricerca.

Siano n la dimensione del database, m la lunghezza del pattern di ricerca e M la memoria disponibile.

Trie o prefix tree

E' albero che memorizza sulle foglie le stringhe e ogni edge è etichettato con una lettera. Tutti i discendenti di un nodo hanno un prefisso in comune. Il costo di ricerca di un pattern è O(m).

Inverted index

Un inverted index è compost da

- Un vocabolario che contiene tutte le differenti parole in un testo

- Posting list -> per ogni parola una lista di documenti che lo contengono e la frequenza con cui appaiono (document-based) o le posizioni in cui appaiono (word based). Ogni document è identificato da un docID.

Costruzione dell'inverted index basato sul trie:

1. Per ogni parola del testo
2. Si cerca la parola nel trie — O(1) per carattere
3. Se la parola non è presente la si aggiunge con una posting list vuota — O(1)
4. Se la parola è nel tries si aggiunte un elemento alla posting list — O(1)

Il costo complessivo è O(n).
L'inverted index è poi scritto su disco in due file, uno per le posting list e uno per i termini (vocabolario).
Il file del vocabolario è memorizzato in ordine lessicografico. Per accedere ad un termine si utilizza una ricerca binaria, ma si ha lo svantaggio che l'aggiornamento è costoso. Si utilizza quindi un B+tree che però utilizza più spazio.
Nel caso la memoria principale non sia sufficiente per tutto l'indice si usa la costruzione con indici parziali. Si costruisce l'indice come descritto prima. Quando la memoria termina si salva l'indice su disco e lo si rimuove dalla memoria. Si inizia a costruire un altro indice ripetendo questi passaggi in caso la memoria termini di nuovo. Alla fine si ha su disco un insieme di indici parziali che dovranno essere uniti.
Dati due indici I1 e I2 l'unione dei vocabolari ordinati ha un costo O(|I1|+|I2|). Se la stessa parola compare in entrambi gli indici si concatena la lista delle occorenze dei due indici. Si ripete il procedimento fino ad avere un solo indice.
La ricerca su un inverted index può essere:

- Single word based
- Prefix o range query -> può essere fatta con binary search, trie o B-tree
- Boolean retrieval -> si crea un syntax tree per la query dove ogni nodo interno ha un operatore mentre sulle foglie le query basiche. Si risolve le query basiche e in base agli operatori modifica il risultato:
  - OR -> effettua unione delle posting list
  - AND -> effettua intersezione delle posting list
  - BUT -> effettua differenza delle posting list
- Phrasal retrieval -> meglio con un inverted index word based
  - Cerca tutti i documenti per ogni singola parola e posizione
  - Interseca i documenti
  - Controlla se è presente una continuità delle posizioni delle parole chiave
  - È meglio iniziare a cercare per la continuità dal termine meno presente
- Proximity retrieval -> usa un approccio simile a quello del phrasa retrieval per trovare tutti i documenti in cui le parole cercate sono in un contesto che soddisfa i vincoli di distanza

Full-text search

Postgres

Operatori di ricerca su testo sono sempre esistiti nei DBMS ma non ci sono supporti linguistici e solitamente sono lenti perché non ci sono indici sul testo. In postegres il testo è memorizzato in campi testuali all'interno dei record del db.
Postegres fornisce due tipi di dati per la ricerca su testo:

- Tsvector -> lista ordinate di lemmi, ovvero di parole che sono state normalizzate (token, stemming)
  - Es. SELECT 'there are two cats in the room'::tsvector; -> 'in' 'are' 'the' 'two' 'cats' 'room' 'there'
  - Es. SELECT to_tsvector('english','there are two cats in the room') -> 'cat':4 'two':3 'room':7
- Tsquery -> lemmi che possono essere ricercati e combinati con operatori logici (&,|,!,<->[followed by])
  - Es. SELECT 'cat & room'::tsquery;
  - Es. SELECT to_tsquery('english','cat & room');

Operatore @@ ritorna true se la query fa match con il testo
Es. SELECT 'there are two cats in the room'::tsvector @@ 'cat & room'::tsquery;
Postgres fornisce due tipi di indici per velocizzare le operazioni sul testo:

- GiST (Generalized search tree) -> lossy perché utilizza una signature di dimensione fissa per ogni documento
- GIN (Generalized inverted index)

Lucene

Libreria java per text search engine. Offre funzionalità per indexing, searching e retrieving dei documenti ma non si occupa di ricevere le query dall'utente o di mostrare i risultati.

Un index prende i suoi dati da una directory su file system. Un Document rappresenta un oggetto che è memorizzato nell'indice. Un field è un oggetto che contiene una coppia chiave valore che sono memorizzati nei document. La classe Analyzer istanzia un analizzatore che si occupa del filtraggio delle stopword nella lingua selezionata. L'IndexWriterConfig contiene le configurazioni dell'indedWriter che si occupa di inserire i documenti nell'index. Il QueryParser è utilizzato per creare un parser che può cercare dati in un indice. Una query:

- + prefisso a un termine indica che esso è richiesto
- - prefisso a un termine indica che è proibito
- Il nome di un campo seguito da: permette di scegliere in quale campo deve essere cercato
- Una frase tra "" indica che il document deve contenere tutta la frase
- Si può anche innestare una query tra ()
- Si possono usare operatori booleani
- Si possono usare wildcard
- Proximity search con ~

L'indexReader apre l'indice mentre l'indexSearcher ricerca i risultati della query. Un collector è utilizzato per collezionare i risultati e ordinarli in base ad un criterio (es. topScore)

Modelli

Un modello di information retrieval è un'astrazione di un vero processo di recupero di informazioni. I risultati ottenuto dal modello sono buoni tanto più il modello è una buona approssimazione della situazione di recupero.
Un modello in generale è composto da tre elementi:

- D -> insieme di documenti
- Q -> insieme di possibili interrogazioni
- R(q,d) : QxD -> R funzione che associa un numero reale che quantifica la similarity di una query q e a un documento d

Per similarità si intende un valore che più elevato più indica che due oggetti sono simili. Al contrario la dissimilarità, detta anche distanza, è un valore che indica quanto due oggetti sono diversi. Queste due misure sono utili per:

- Classificazione -> dato un oggetto non etichettato lo si assegna ad una classe specifica
- Clustering -> trovare i raggruppamenti naturare di oggetti basandosi su misure di similarità

La misura di similarità si basa sul confronto della rappresentazione logica delle features degli oggetti interessati. Queste feature possono dover essere:

- Generate -> utilizzando features già presenti
- Pulite -> rimozione rumore
- Normalizzate
- Ridotte -> un numero eccessivo feature ridotte con dimensionality reduction

Un modello IR classico utilizza

- Index term -> insieme di parole la cui semantica aiuta nella ricerca di documenti (principalmente i nomi)
- Pesi -> non tutti i termini sono ugualmente utili per descrivere il contenuto di un document

Indichiamo con k_i un generico index term, d_j un generic documento e w_{ij} è il peso del termine i nel documento j. Ogni peso è mutualmente indipendente ovvero non ogni peso non dipende dagli altri.

Modello Booleano

Modello che permette di risolvere esattamente query che utilizzano operatori logici. Ogni peso vale o 0 o 1. Gli operatori booleani si basano sugli operatori di insiemistica. È un modello semplice ma che non fornisce un ranking dei documenti e permette solo di avere documenti che fanno match esatto con la query.

Modello Vettoriale

L'idea alla base del modello vettoriale è quella di rappresentare tutto come un vettore in uno spazio a molte dimensioni. Le feature permettono di rappresentare un oggetto nello spazio delle features. La misura della distanza in questo spazio può essere utlizzata come dissimilarità.
Dopo il preprocessing si forma un vocabolario di t termini unici. Questi termini formano uno spazio di dimensione t. Per ogni termine in un documento o in una query è assegnato un valore di peso -> sia documenti che query possono essere rappresentati come un vettore di pesi.
La similarità del coseno è una possibile misura di similarità. Se due vettori sono paralleli allora la similarità è massima, altrimenti se sono ortogonali è minima.

$$sim(d_j, q) = \frac{d_j . q}{|d_j||q|} = \frac{\sum_{i=1}^{t} w_{ij} * w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} * \sqrt{\sum_{i=1}^{t} w_{iq}^2}}$$

La frequenza di un termine è collegata all'importanza di quel termine e può essere utilizzata per misurare quanto un termine descrive il contenuto del documento. Vi sono 3 possibili misure:

- Si utilizza solo la frequenza f_{ij} del termine k_i nel documento d_j
- Si normalizza rispetto alla frequenza massima dei termini nel documento $f_{ij} = \frac{f_{ij}}{\max_l f_{lj}}$
- Si utilizza il log della frequenza se è > 0 altrimenti è pari a 0 $f_{ij} = \begin{cases} 1 + \log_{10} f_{ij} & f_{ij} > 0 \\ 0 & \text{altrimenti} \end{cases}$

Le parole con frequenza basso sono molto più informative di quelle ad alta frequenza. Si vuole quindi dare un peso maggiore a termini poco frequenti. Sia df_t il numero di documenti in cui appare il termine t. Si ha quindi che df_t è una misura inversa del contenuto informativo del termine t.

Definiamo inverse document frequency $idf_t = \log\_10(\frac{N}{df_t})$ dove Nè il numero totale di documenti. Si usa il log per smorzare l'effetto dell'idf.

A questo punto possiamo definire il peso di ogni termine come $w_{ij} = f_{ij} * idf_i$ che

- Aumenta all'aumentare del numero di occorrenze del termine i nel documento j
- Aumenta con la rarità del termine nell'insieme di tutti i documenti.

I principali vantaggi di questo modello sono la possibilità di ottenere anche risultati che approssimano le condizioni della query ma mancano informazioni riguardanti la sintassi e la semantica dei termini e si assume che gli index term sono mutualmente indipendenti.

In pratica pur avendo un dizionario di termini molto grande molti dei documenti e delle query non contengono la maggior parte dei valori, quindi i vettori sono sparsi. Inoltre i termini che non sono sia nella query sia nel documento non influenzano la similarità del coseno, in quanto il loro prodotto sarà 0.

Se si usa un inverted index per memorizzare i vettori dei documenti allora si cercano prima di tutto solo i documenti che contengono almeno una dei termini presenti nella query. Se si suppone che in media ci sono B documenti che hanno un termine in comune con la query allora ricerca questi documenti costa O(|Q|B) che è molto minore rispetto al costo per esaminare tutti i documenti O(|V|N)

Modello probabilistico

Il modello probabilistico ordina i documenti in base alla probabilità che un documento d sia rilevante per una query q, ovvero P (R=1|d, q) dove R = 1 indica che il risultato è rilevante altrimenti R = 0.

Il Binary Indipendent Model è un modello in cui query e documenti sono rappresentati da vettori dove ogni componente ha valore 0 o 1 per indicare se il termine corrispondente è rilevante o no. Si assume nessuna correlazione tra i termini, non sempre vero in generale ma funziona in pratica.

La misura di similarità utilizzata è $sim(d_j, q) = \frac{P(R|d_j,q)}{P(\bar{R}|d_j,q)}$ e dalla regola di Bayes $P(A|B)P(B) = P(B|A)P(A)$

$$sim(d_j, q) = \frac{P(R|d_j, q)}{P(\bar{R}|d_j, q)} = \frac{P(d_j|q, R)P(R|q)}{P(d_j|q, \bar{R})P(\bar{R}|q)}$$

Dove

$P(d_j|q, R)$ -> probabilità di estrarre il documento d_j dall'insieme dei documenti rilevanti per q

$P(R|q)$ -> probabilità di estrarre un document rilevante per q da tutti i documenti

Supponendo che $P(R|q) = P(\bar{R}|q)$ allora

$$sim(d_j, q) = \frac{P(d_j|q, R)}{P(d_j|q, \bar{R})}$$

Assumendo l'indipendenza dei termini

$$sim(d_j,q) \approx \frac{\left(\prod_{w_{ij}=1} P(k_i \mid R,\vec{q})\right) \times \left(\prod_{w_{ij}=0} P(\bar{k}_i \mid R,\vec{q})\right)}{\left(\prod_{w_{ii}=1} P(k_i \mid \bar{R},\vec{q})\right) \times \left(\prod_{w_{ii}=0} P(\bar{k}_i \mid \bar{R},\vec{q})\right)}$$

Dove P (k_i|R, q) è la probabilità che il termine k_i sia presente in un documento estratto casualmente dal set di quelli rilevanti per R. Applicando il log e ricordando che la somma delle probabilità opposte è 1

$$\begin{aligned} sim(d_j,q) &\approx \log \prod_{i=1}^{t} \left(\frac{P(k_i \mid R,\vec{q})}{P(k_i \mid \bar{R},\vec{q})}\right)^{w_{ij}} \left(\frac{1-P(k_i \mid R,\vec{q})}{1-P(k_i \mid \bar{R},\vec{q})}\right)^{1-w_{ij}} \\ &\approx \sum_{i=1}^{t} \left(w_{ij} \log\left(\frac{P(k_i \mid R,\vec{q})}{P(k_i \mid \bar{R},\vec{q})}\right) + (1-w_{ij}) \log\left(\frac{1-P(k_i \mid R,\vec{q})}{1-P(k_i \mid \bar{R},\vec{q})}\right)\right) \end{aligned}$$

Ignorando le costanti e concentrandosi

sui termini che sono comuni per la query e i documenti

documents

$$sim(d_j,q) \approx \sum_{i=1}^{t} w_{iq} \times w_{ij} \times \overbrace{\left(\log \frac{P(k_i \mid R,\vec{q})\left(1-P(k_i \mid \bar{R},\vec{q})\right)}{\left(1-P(k_i \mid R,\vec{q})\right)P(k_i \mid \bar{R},\vec{q})} \right)}^{c_i}$$

$$\approx \sum_{i=1}^{t} w_{iq} \times w_{ij} \times \left(\log \frac{P(k_i \mid R,\vec{q})}{1-P(k_i \mid R,\vec{q})} + \log \frac{1-P(k_i \mid \bar{R},\vec{q})}{P(k_i \mid \bar{R},\vec{q})} \right)$$

Il termine c_i è composto dalla probabilità che il termine k appaia se il documento è rilevante e la probabilità che il termine appaia se il documento non è rilevante. Se $c_i = 0$ allora vuole dire che le probabilità sono identiche, invece è positivo se è più probabile che appaia in documenti rilevanti.

Bisogna trovare un modo per calcolare le $P(k_i|R,q)$ e $P(\bar{k}_i|R,q)$. Si assume che la percentuale di documenti rilevanti è molto piccola rispetto ai documenti totali, quindi si inizia fissando $P(k_i|R,q) = 0.5$ e $P(\bar{k}_i|R,q) = n_i/N$ con n_i numero di documenti che contengono k_i. Questa ipotesi iniziale permette di ritornare documenti che contengono i termini della query. Le probabilità iniziali possono essere migliorate sulla base dei documenti ottenuti dalla prima ricerca.

Sia V un sottoinsieme (i primi r) dei documenti ritornati dalla ricerca iniziale. Sia V_i un sottoinsieme di V che contiene i documenti in cui appare k_i, allora si possono aggiornare le probabilità come

$$P(k_i|R,q) = V_i / V \qquad P(\bar{k}_i|R,q) = (n_i - V_i)/(N - V)$$

Per evitare che ci possano essere degli zeri e per applicare un po' di smoothing esistono altre versioni. Gli svantaggi principali del modello probabilistico sono che bisogna trovare la separazione iniziale dei documenti tra rilevanti e non rilevanti, i pesi sono binary quindi non si considera la frequenza con cui essi appaiono, e si considera che tutti i termini siano indipendenti.

Modello fuzzy

La logica fuzzy è un sovrainsieme della logica booleana. La funzione caratteristica della logica fuzzy permette di esprimere un grado di appartenenza di un oggetto ad una certa classe F(x) : X -> [0,1]. Gli operatori logici sono:

- Fuzzy OR = max(F(x1), F(x2))
- Fuzzy AND = min(F(x1), F(x2))
- Fuzzy NOT = 1 – F(x)

La logica fuzzy può essere utilizzata per risolvere query con operatori logici nel modello vettoriale. Sia Q una query formata da sottoquery A_i unite da operatori logici. Ogni query A_i crea un fuzzy set e per ogni document d_j si può calcolare la sua appartenenza all'insieme come $sim(A_i, d_j)$. A questo punto per ogni documento d_j si ha che $sim(Q, d_j)$ può essere calcolata con gli operatori fuzzy prima descritti e i valori di appartenenza calcolati per ogni A_i.

Valutazione

Le misure principali di un sistema IR sono

- Velocità con cui crea un index
- Velocità di ricerca in funzione delle dimensioni dell'index
- Espressività del linguaggio di query, ovvero se è possibile esprimere query complesse e quanto è veloce ad eseguire queste query

La misura chiave però è l'user happiness, ovvero quanto il sistema IR può risolvere efficacemente le query sottoposte dall'utente. Per misurare l'user happiness si tende a guardare la rilevanza dei documenti ritornati da una ricerca. Per compiere questa misura sono necessari:

- Un benchmark di documenti
- Un benchmark di query
- Un modo per indicare vedere se un documento è rilevante o no per una data query (non necessariamente binario)

Un'altra misura è l'efficacia che dipende dal retrieval task adoperato:

- Batch task -> l'utente sottopone una query e viene ritornato un insieme di documenti -> si valuta la qualità di questo insieme rispetto la query
- Interactive task -> l'utente specifica le informazioni richieste attraverso dei passaggi interattivi con il sistema -> si valuta l'user effort, il design dell'interfaccia, l'aiuto dato dal sistema e la durata della sessione interattiva

Siano R i documenti rilevanti per una query

Siano A i documenti ritornati dal sistema per quella query

Si definisce recall il rapporto tra il numero di documenti rilevanti in A e il numero totale di documenti in R

Si definisce precision il rapporto tra il numero di documenti rilevanti in A e il numero totale di documenti in A
In un buon sistema la precisione diminuisce all'aumentare dei documenti ritornati o al crescere del recall -> non è un teorema ma è basato su conferme empiriche.
L'ideale sarebbe avere entrambi i valori a 1 ma non è possibile essendo precision e recall inversamente proporzionali.
Per valutare dei risultati con il rank si misura come la precision si comporta scegliendo come insieme di risultati documenti con un rank sempre più basso. Scendendo con il ranking il recall sale. L'idea è di considerare 11 livelli di recall e di misurare la precision in ognuno di questi punti. Questo permette di dire quale è la precision se sono stati ritornati una % di documenti rilevanti. Nel caso si vogliano confrontare i risultati di due query basati su livelli di recall diversi da quelli standard conviene normalizzare i livelli.
Per fare ciò si associa ad ogni livello standard r_j il valore massimo di precision che vi è tra il livello j e il livello j+1.
Per avere un'idea generale delle performance del sistema bisogna vedere come variano precision e recall su un insieme consistente di query. Per ogni livello di recall r definiamo precision media

$$\bar{P}(r) = \sum_{i=1}^{Nq} \frac{P_i(r)}{Nq}$$

Un grafico precision media/recall permette di avere una vision qualitativa dell'intero sistema e anche un'idea dell'ampiezza di ricerca del sistema. È anche importante comparare singolarmente l'efficacia di ogni singola query utilizzando un solo valore di precision, in quanto la precision media su molte query potrebbe nascondere anomalie importanti e in caso si stiano comparando due algoritmi si potrebbe essere interessati quali dei due è migliore su un preciso insieme di query.
Alcune misure di precision sono

- Interpolated average precision $\sum_{r=0}^{n} \frac{P_q(r)}{n}$
- Non-Interpolated average precision $\sum_{r=0}^{n} \frac{P_q(r/|R_q|)}{|R_q|}$ dove R_q è l'insieme dei documenti rilevanti per q
- R-precision -> precisione scegliendo i primi R documenti nel ranking
- Precision histogram -> compara i risultati di due algoritmi in base alla loro R-precision $RP_{A|B}(i) = RP_A(i) - RP_B(i)$ e si plottano un istogramma per ogni query

I problemi principali di utilizzare precision/recall sono:

- Il calcolo della recall richiede una conoscenza dettagliata di tutti i documenti, cosa non possibile su un insieme elevato di documenti
- Precision/recall sono misure fatte per sistemi batch, ma molti sistemi moderni sono interattivi
- Precision/recall sono misure che catturano aspetti differenti dell'insieme di documenti ritornati, ma a volte è più utile avere una singola misura che combina entrambi i valori.

Misure alternative a precision/recall sono

- La media armonica $F = \frac{2*precision*recall}{precision+recall}$
- La E-measure $E = 1 - \frac{1+b^2}{\frac{b^2}{recall}+\frac{1}{precision}}$ dove b è un alore che specifica se si è più interessanti alla precision(b>1) o alle recall (b<1) oppure ad entrambi (b=1).

Un'assunzione fatta utilizzando precision e recall è che l'insieme dei documenti rilevanti è lo stesso per ogni utente, mentre ogni utente potrebbe avere idee diverse su quali documenti sono rilevanti e quali no.
Definiamo

- Coverage ratio -> la frazione di documenti che si sa essere rilevanti e che sono stati ritornati. Se molto alto il sistema sta trovando documenti che l'utente si aspettava
- Novelty ratio -> la frazione di rilevanti che erano sconosciuti all'utente. Se molto alto il sistema sta trovando nuovi documenti rilevanti prima sconosciuti all'utente

Un **benchmark** contiene un insieme di: documenti, query e una lista di documenti rilevanti per ogni query.
Alcuni tra i più famosi sono:

- TREC -> insieme di diversi benchmark, quasi 2 milioni di documenti
- CLEF -> controparte europea di TREC fornisce test-suites di dati riutilizzabili che possono essere in monolingua o cross language per fare benchmark

Per mostrare all'utente i risultati di una query si possono usare:

- Riassunti statici -> il riassunto è sempre lo stesso indipendentemente dalla query
    - Le prime 50 parole
    - Estrarre dal testo alcune frasi chiave
    - Generare un riassunto tramite machine learning
- Riassunti dinamici -> il riassunto cambia in base al contenuto della query e prova a spiegare perché il documento è stato scelto.

Tolerant Retrieval

Con inverted index che memorizzano la posizione di ogni termine in ogni documento è possibile risolvere phrase query e proxymity query.
Query con wildcards:

- Prefix query -> (es. mon*) si risolvono con B-tree o dizionario ordinato
- Suffix query -> (es. *mon) si risolvono mantenendo un dizionario ordinato dei termini scritti al contrario
- Le query che hanno una wildcards nel mezzo si risolvono con un permutation index

L'idea alla base dei permutation index è quella di ruotare ogni wildcard cosicché il * sia alla fine della ricerca e la query risulti come una prefix query. Il permutation index memorizza tutte le possibili rotazioni in un dizionario e collegarle con il termine originale. Ogni termine nel permutation index è detto permuterm.
Una alternativa al permutation index sono i q-gram. I q-gram sono sequenze si sottostringhe di lunghezza q della stringa originale. I caratteri #, $ vengono utilizzati come delimitatori di inizio e fine parola. Una stringa di lunghezza L ha L + q – 1 q-grams. Si mantiene quindi un secondo inverted index con i bi-grams (q-grams con q=2) che mappano direttamente i termini del dizionario del primo inverted index. In questo modo ogni query con wildcard può essere scomposta in bi-grams. Per risolvere la query si ricercano tutti i termini che contengono tutti i bi-grams della query (ogni bi-grams è messo ad AND con gli altri). Bisogna però filtrare i risultati così ottenuti perché potrebbero essere presenti dei falsi positivi.
Ci sono due principali usi per lo spell correction:

- Correggere i documenti -> utile per documenti ottenuti tramite OCR
- Correggere la query dell'user per ottenere risultati corretti

Lo spell correction può essere su singola parola o context-sensitive. Per il primo caso si usa un lexicon da cui prendere le parole corrette, può essere un vocabolario di una lingua o generato da dei documenti. E' necessario poi avere un modo per calcolare la distanza tra una parola sbagliata e quella corretta.
La prima misura possibile è l'edit distance che è il numero minimo di operazioni per ottenere una parola dall'altra. Operazioni tipiche sono l'inserimento, la cancellazione, la sostituzione o la trasposizione di un carattere. L'edit distance tra due parole X e Y si trova costruendo una matrice C di dimensioni $|X|+1$ x $|Y|+1$ e si ha
$C_{i,0}= i$ $C_{0,j}= j$ $C_{i,j}=0$ se $x_i=y_j$ altrimenti $C_{i,j}=1+\min(C_{i-1,j-1}, C_{i-1,j}, C_{i,,j-1})$ e l'edit distance è nell'ultima cella in basso a destra.
Un altro problema è quella di trovare dato un pattern di lunghezza m e un testo di lunghezza n e un numero massimo di errori k, tutte le posizioni di testo in cui il pattern appare con al massimo k errori. Si può usare lo stesso algoritmo per l'edit distance inizializzando $C_{0,j}= j$ in quanto ogni posizione può essere valida per l'inizio del match. Le posizioni in cui vi sono i match con massimo k errori sono le posizioni nell'ultima riga in cui il valore è minore o uguale a k. Una variazione di questo algoritmo prevede utilizzo di pesi diversi in base ai caratteri che vengono confrontati. Per esempio una m può essere scambiata più facilmente con una n, quindi ha un peso minore di una q. Data una query si può creare una lista di parole che hanno un certo valore massimo di edit distance. Si interseca questa lista con quella delle parole corrette. Il risultato di questa operazione può

- Essere mostrato come suggerimenti
- Si cercano tutte le possibili correzioni nell'inverted index ma è lento
- Si usa una singola correzione

Le ultime due rimuovono alcune libertà all'utente ma permetto di risparmiare tempo di interazione tra utente e sistema. Data una query bisogna trovare quali termini sono possibili correzioni. Per evitare di utilizzare tutto il vocabolario si può utilizzare un filtro basato sui q-gram. Questi possono essere anche usati direttamente per la correzione (q-gram overlap). Il q-gram overlap consiste nel generare tutti i q-gram della query e di utilizzarli per cercare i termini del vocabolario che matchano con almeno uno de q-gram. Si utilizza il q-gram index per fare ciò. Si ritornano tutti i termini che hanno un numero di q-gram in comune con la query maggiore di una certa soglia. Una misura dell'overlap dei q-gram può essere quella di Jaccard che equivale al numero di q-gram in comune diviso il numero totale di q-gram.
Per migliorare le prestazioni dell'edit distance un insieme di filtri basati su q-gram sono stati proposti:

- Length filter -> stringhe con una differenza di lunghezza maggiore di k non possono essere a un edit distance minore di k
- Count filter -> stringhe con edit distance < k devono avere al massimo [max(length(S1), length(S2))+q- 1] kq q-grams in comune
- Position filter -> non effettuare il match tra q-gram che sono a più di k posizioni distanti

L'obiettivo del filtraggio è quello i ridurre lo spazio di ricerca delle parole che potrebbero essere correzioni.
Data una query P, una collezione di test TC e una soglia di distanza k, il problema del tolerant retrieval consiste nel trovare tutti i testi T tale che ed (P, T) <= k.
Nel caso di correzioni context-sensitive si cercano tutti i termini simili in edit distance ai termini della query. Si provano tutti i possibili risultati e si suggerisce come alternativa quella con più risultati. Questo metodo non è molto efficiente, meglio un metodo basato sull'analisi del log delle query.

Soundex

Soundex è un algoritmo che indicizza i nomi in base al loro suono in inglese. L'obiettivo è quello di memorizzare le parole che hanno un suono praticamente identico con la stessa rappresentazione cosicchè possano essere matchate anche con piccole differenze di spelling. L'algoritmo memorizza principalmente le consonanti e le vocali solo se sono a inizio parola.

Web

I primi motori di ricerca utilizzavano una matrice per rappresentare per ogni pagina quante volte ogni termine apparisse. I risultati per le ricerche basate su keyword erano tutte le pagine che contenevano le keyword. Nasce anche il problema dello spamming, ovvero di pagine che hanno una o più keyword ripetute tantissime volte solo per apparire in cima alle ricerche. Il problema di questo metodo è la mancanza di un criterio che descriva l'importanza di una pagina. L'idea principale è quella di sfruttare la struttura topologica del web, utilizzando i link. Una pagina p che contiene un link alla pagina q considera q un'autorità riguardante la materia trattata. Si assegna quindi un valore di importanza ad ogni pagina per ogni altra pagina che abbia un link che porti ad essa. L'algoritmo Link Analysis rank consiste nel creare un grafo di hyperlink a partire da un insieme di pagine e da come output un peso ad ogni pagina. L'insieme di pagine iniziali può essere:

- Query independent -> rank tutto il web
- Query dependent -> rank solo un piccolo sottoinsieme di pagine relative alla quer..ly

Si chiamano autorità le pagine che sono riconosciute e forniscono un contenuto informativo significativo e affidabile su un argomento. Il valore di autorità di una pagina p è dato dal numero di citazioni e dall'autorità delle altre pagine q che citano p. Il rank di ogni pagina è quindi

$$R(p) = c \sum_{q:q\to p} \frac{R(q)}{N_q}$$

Dove c'è una costante di normalizzazione e N_q è il numero di link uscenti da q.
Il processo di assegnamento dei rank è iterativo. Dato un set S di documenti si inizializza $R_0(p) = \frac{1}{|S|}$ per ogni documento. Fino a che non si è raggiunta una convergenza per ogni documento

$$R'_i(p) = \sum_{q:q\to p} \frac{R_{i-1}(q)}{N_q} \qquad R_i(p) = cR'_i(p) \quad con\ c = \frac{1}{\sum_{p\in S} R'_i(p)}$$

Un problema dell'idea iniziale si presenta quando due o più pagine creano un ciclo tra di loro. Si ha quindi che il ciclo assorbe tutto il rank del sistema. Si può risolvere aggiungendo un rank source E che fornisce sempre il rank di ogni pagina di una quantità fissa.
L'algoritmo di page rank può essere modificato per ottenere:

- Personalizzazione -> cambiando la distribuzione di E(p) con una non uniforme. Se per esempio E(p) = c solo per la homepage di un sito e = a 0 per ogni altra pagina si avrà che le pagine più vicine alla homepage avranno un rank maggiore
- Topic sensitive -> calcola tanti page rank per ogni topic, stima la rilevanza di ogni pagina per ogni topic e il rank finale della pagina è una combinazione pesata di questi valori

Secondo test empirici la velocità di convergenza è pari a O(log n) con n numero di link. Secondo altre ricerche si ha che il rank deve essere aggiornato spesso in quanto in un anno quasi l'80% dei link viene sostituito e ogni settimana 25% di nuovi link vengono creati. Esistono quindi una serie di metodi matematici e approssimati per velocizzare il ricalcolo dei rank.

Si definiscono hub pagine che non contengono molte informazioni ma linkano a pagine dove trovarle. L'autorità non è necessariamente trasferita tra altre autorità. Si ha quindi che dei buoni hub puntano a buone autorità e buone autorità debbano essere puntati da buoni hub. Ogni pagina ha due valori, uno di authority e una di hubness. Un sito è molto autoritativo se riceve molte citazioni e le citazioni da siti importanti pesano di più. L'hubness mostra l'importanza del sito, un buon hub ha link a molti siti autoritativi.
Algoritmo HITS
Prima di tutto si devono trovare un set di R pagine contenenti una parola chiave -> root set
Trovare tutte le pagine a cui linkano queste R pagine -> children
Trovare tutte le pagine che puntano alle R selezionate -> parent
Calcolare il grafo S di tutte le pagine trovate -> base set
Si inizia settando a(V) = h(V) = 1. Ad ogni iterazione h(V) diventa il totale della somma dei valori di authority dei figli, mentre il valore a(V) diventa la somma dei valori di hubness dei padri. Si normalizzano i valori e si itera fino alla convergenza. Problema HITS, se si crea una pagina che punta a molte pagine autoritative allora la pagina diventa un buon hub e se si aggiunge un link alla propria homepage quest'ultima diventa un buon authority.
Nei primi motori di ricerca che si basavano fortemente su tf/idf una forma di spam era quella di creare pagine che contenessero moltissime ripetizioni dei termini chiave per cui si voleva apparire in cima al rank. Un altro metodo è quello di utilizzare meta-tag forvianti e ripetuti. Tecnica più sofisticata è il cloacking, ovvero mostrare pagine diverse agli spider dei motori di ricerca e fornirne altre all'utente.

XML

Extensible Markup language -> linguaggio estendibile utilizzato per la rappresentazione strutturale di documenti. Non vi sono tag predefiniti e permette la definizione di nuovi metalinguaggio attraverso la creazione dei propri tag. XML è utilizzato per lo scambio di dati tra applicazioni diverse. Un elemento XML è composto da un tag di apertura e uno di chiusura e può avere un valore contenuto tra essi. Ogni elemento può poi avere sotto elementi, ovvero coppie di tag inseriti in mezzo ai tag dell'elemento padre. Un documento XML può essere visto come un albero dove i nodi intermedi sono elementi mentre le foglie sono i valori. Ogni elemento può avere pii degli attributi inseriti nel tag di apertura con il formato chiave="valore". Ogni documento può avere un prologo che specifica che il documento è in XML e la versione del linguaggio. E' opzionale. Tutti i documenti XML hanno un elemento root e tutti gli altri elementi sono figli del root.
Un documento XML può avere due livelli di correttezza:

- Ben formato -> contiene un elemento radice e tutti gli altri elementi sono figli di questo e i tag sono usati correttamente
- Valido -> deve essere ben formato e la struttura deve rispettare un DTD o uno schema XML

XML è case sensitive. XML permette anche l'utilizzo di namespace per evitare un conflitto tra i nomi. < namespace:TAG>. Il namespace di default è indicato con l'attributo xmlns mentre gli altri namespace sono definiti con l'attributo xmlns:namespace.
Un XML schema permette di specificare la struttura di un documento. I costrutti principali sono:

- Element -> associa un elemento a un tipo
- Simple type -> definisce un set di stringhe da usare per i valori di un attributo
    - Integer, date, URI
    - Tipi di dati derivati da altri semplice list, union
- Complex type -> definisce attributi, sotto elementi e dati necessari per il tipo

Per dichiarare che un documento XML rispetta uno schema si utilizza l'attributo schemaLocation che contiene il link allo schema. XSL è un linguaggio che permette di definire come un documento XML debba essere presentato. Il parsing di un documento XML può avvenire in due modi:

- Tree based (DOM) -> il documento XML produce una struttura ad albero che può essere navigata.
    - Document Object Model
        - Document -> oggetto che rappresenta il documento stesso
        - Nodo -> un elemento o attributo
        - Elemento -> un oggetto che rappresenta un elemento
        - Attributo -> coppia chiave valore
        - Document fragment -> parte del documento
- Event Based (SAX) -> gestione basata su eventi

DOM memorizza la struttura ad albero del documento in memoria e permette di navigarlo e modificarlo. SAX permette solo un accesso sequenziale del documento e non permette la modifica del documento.