Complementi di Basi di Dati

Intro Models

La maggior parte dei dati disponibili oggi non è in forma relazionale.
Questi dati hanno proprietà diverse rispetto a quelli gestiti da sistemi relazionali.
Basandoci su una scala che va da dati strutturati a dati non strutturati, i dati non strutturati necessitano di metodi specifici per essere gestiti (Information Retrieval).
I dati al centro della scala sono dati semi-strutturati, avendo proprietà di entrambe le tipologie di dati. Uno dei modi per rappresentarli è XML.

Dati semi-strutturati

Il formato principale per la loro rappresentazione è **XML**. XML può essere usato per rappresentare dati strutturati (scambio dati tra applicazioni), ma anche per rappresentare dati semi-strutturati, sfruttando la flessibilità e la capacità di indicare sia i dati che lo schema all'interno.
Nel primo caso, i dati possono essere salvati anche all'interno di un DBMS e poi convertiti in XML. Nel secondo caso, il modello relazione non è adatto.
Infatti, XML è stato creato per lo scambio di dati tra applicazioni.
Molte query non sono fattibili in SQL senza usare ricorsione o diventando inefficienti. Il trend è quello di sviluppare sistemi specifici per XML.

Alcuni limiti del modello relazionale sono:

- Avere campi di testo che contengono migliaia di caratteri senza una struttura
- Non si possono aggiungere nuovi campi senza modificare lo schema
- Per scambiare dei dati, bisogna inviare l'intero schema con i dati

Alcune proprietà dei dati semi-strutturati sono:

- La struttura è irregolare o parziale
- La struttura è costrituira a posteriori
- Lo schema è molto flessibile e si evolve velocemente, come per i dati

Dati non strutturati

Quando si passa da dati semi-strutturati a dati non strutturati, lo schema viene completamente a mancare. La disciplina principale che studia questi dati è detta Information Retrieval. I dati che non hanno uno schema sono molto importanti, basti pensare all'internet ed ai motori di ricerca, che si basano principalmente su sistemi di Information Retrieval.
Le query per dati non strutturati sono molto semplici, composte da liste di parole chiave.
In modelli relazionali, le query sono precise e ritornano risultati in maniera booleana (la tupla è presente o meno nel risultato).
Nel caso invece di Information Retrieval, questo metodo non è utilizzabile, dato che le possibili risposte sono tantissime e che molti documenti potrebbero essere attinenti o meno

alla risposta cercata. Non è quindi semplice capire se una risposta è corretta o meno. Le risposte vengono quindi rankate in base al grado di rilevanza. Vengono quindi usate metriche per descrivere la completezza e correttezza delle risposte.

Un'ultima nota: molti DMBS stanno integrando al loro interno funzionalità derivate dall'Information Retrieval, come indicizzare le colonne che contengono solo testo o colonne con dati multimediali.

Intro XML

XML deriva da SGML, come anche HTML. È possibile definire linguaggi di markup per domini specifici. Inizialmente XML è nato come formato per scambiare dati, ma viene anche utilizzato per salvare dati e per cercare all'interno di dati.

Il **prolog** contiene informazioni utili per interpretare il documento (dichiarazione che il documento è in formato XML, la grammatica per validare il documento, commenti per software che useranno il documento).

Il **body** del documento è formato da un singolo elemento, che a sua volta contiene tutti gli altri elementi con una struttura ad albero.

Gli **elementi** possono avere diverse forme, l'importante è che siano scritte bene:

- Ogni elemento deve essere incluso in tag di apertura e chiusura
- I nomi degli elementi e attributi sono case sensitive
- Devono essere nestati correttamente
- I valori degli attributi devono essere tra "" o ''
- Un elemento non può avere più di un attributo con lo stesso nome

All'inizio di un documento XML può essere definito anche un **Document Type Declaration** (**DTD**). Serve per definire tutti i vincoli che il documento deve rispettare.
Un documento XML è valido se contiene un DTD, e se lo rispetta.

XML dà molta flessibilità di utilizzo. Quando viene utilizzato per salvare dati, va fatta particolare attenzione all'utilizzo corretto degli elementi:

- I dati devono essere salvati nel contenuto degli elementi
- I metadati degli elementi devono essere attributi dell'elemento
- Le gerarchie devono essere usate correttamente

SQL/XML

La maggior parte dei dati è salvato in database relazionali, ma XML viene usato per scambiare dati tra applicazioni. Viene usato SQL/XML per convertire da linguaggio relazionale a XML.
Per usare questa combinazione i dati devono avere due funzionalità:

- Estrarre XML da una o piò tabelle relazionali: più semplice, XML viene creato per gestire sia dati strutturati che semi
- Salvare XML in una o più tabelle relazionali: più complesso, per lo stesso motivo

Estrarre XML da una tabella è semplice, ci sono due modi per farlo.
Il **primo** consiste in:

- Rappresentare la tabella in XML (**mapping**)
- Estrarre i dati usando tecnologie XML (**XQuery**)

Si parte dalla tabella relazionale, si procede con il mapping facendo diventare il nome della tabella il nome del documento, ogni riga un elemento <row>, ogni valore è incluso in un elemento col nome della colonna, ed i valori nulli vengono rappresentati con attributo nil=true.
Si ottiene così un file XML. Ora si può procedere con XQuery per ottenere il risultato.

Il **secondo** consiste in:

- Estrarre i dati in una tabella SQL
- Trasformare la tabella SQL in XML

Si parte dalla tabella relazionale, si scrive una query SQL per ottenere i dati d'interesse. Si aggiunge poi un costruttore XML per ottenere il risultato sperato in formato XML.

Per garantire queste funzionalità i sistemi usano tecniche personalizzate (usare colonne OO per salvare XML in un singolo campo, oppure dividere i documenti per salvarli in campi diversi).

Il linguaggio SQL/XML

È una **estensione di SQL**. Fornisce diverse funzioni. Viene prima computata la query SQL, poi vengono aggiunti i comandi XML costruendo la risposta richiesta. I comandi sono:

- **XMLELEMENT**: permette di creare un elemento XML. Può essere usato anche concatenato
- **XMLATTRIBUTES**: permette di aggiungere attributi
- **XMLFOREST**: permette di creare liste di elementi
- **XMLCONCAT**: permette di concatenare diversi argomenti, producendo una foresta di elementi
- **XMLAGG**: funziona come il GROUP BY di SQL
- **XMLGEN**: permette di specificare codice XML per ottenere un risultato

XQuery

XQuery è un linguaggio di query per dati XML, e permette quindi di accedere a dati strutturati e semi. XQuery **opera su sequenze** (a differenza di **SQL** che **opera su relazioni**) che possono contenere **valori atomici** e **nodi**. Una espressione XQuery riceve zero o più sequenze e produce una sequenza. Le sequenze sono ordinate e non annidate, inoltre non c'è differenza tra un oggetto in una sequenza con l'oggetto stesso.

I documenti XML sono rappresentati come alberi, con diversi nodi.
Quando viene eseguita una query XQuery, l'ordine non corrisponde a quello corretto.

Le **Path Expressions** possono essere usate per estrarre valore da nodi e alberi, e per controllare le loro proprietà. Anche in questo elaborano sequenze. Ogni step è valutato in un contesto e produce sequenze.

FLWOR Expressions

Simile a query SQL, compost da 5 parti:

- **For**: associa una o più variabili all'espressione
- **Let**: crea degli alias di tutti i risultati di una espressione
- **Where**: filtra in base ad una condizione
- **Order by**: ordina il risultato
- **Return**: creare il risultato della FLWOR

La differenza tra for e let è che con for, ogni elemento è collegato alla variabile uno ad uno, nel let invece tutti gli elementi (la sequenza) viene collegata una volta sola.

Si possono usare anche espressioni condizionali come **if else**, operatori di comparazione, operatori logici e aritmetici.
Si possono anche dare file in input, funzioni per le sequenze dei nodi, funzioni aggregate come count, avg, max, min, sum.

NoSQL

Andando nel contesto dei Big Data, nasce la necessità di database con schema flessibili (o senza schema), per dati distribuiti ed accessibili ad alte velocità.

In generale, i database relazionali non sono scalabili, hanno velocità limitate dalle transazioni, hanno uno schema rigido ed hanno difficoltà con dati non strutturati.

Nascono quindi database NoSQL, che quindi non posseggono uno schema o non seguono le regole SQL, ed inoltre non rispettano le proprietà ACID dei database relazionali, ma le proprietà BASE.

ACID:

- **A**: atomicità, ogni transazione è tutto o niente
- **C**: consistenza dei dati, tutti i dati devono sottostare alle regole definite
- **I**: isolamento, le diverse transazioni non devono influenzarsi
- **D**: durabilità, dati caricati non devono essere persi

BASE:

- **BA**: disponibilità basica, vengono tollerati fallimenti parziali. Ci sarà una risposta per ogni richiesta, con dati anche possibilmente inconsistenti
- **S**: stato "soft", lo stato del db può cambiare nel tempo, anche senza input (causato da eventual consistency)
- **E**: consistenza eventuale, può esserci inconsistenza inizialmente, ma eventualmente sarà consistente. Il sistema non controlla la consistenza dopo ogni transazione

Non si possono avere entrambe le proprietà per colpa del CAP theorem:

- **C**: consistenza, si riferisce ad un sistema che deve sempre sempre operare al 100%
- **A**: disponibilità, si riferisce ad un sistema sempre disponibile quando richiamato
- **P**: partition tolerance, un sistema che può continuare a funzionare anche in caso di fallimenti all'interno del sistema. Un singolo nodo che fallisce non deve portare ad un crash totale

In contesti distribuiti e in larga scala, c'è bisogno di disponibilità e partition tolerance, quindi le proprietà ACID non reggono. I sistemi relazionali seguono un modello CA, mentre altri sistemi non relazionali seguono i modelli CP e AP.

Nei sistemi NoSQL **non c'è uno schema ben definito**. I dati vengono **aggregati** in diverse strutture. Non c'è più quindi bisogno di effettuare join, i dati collegati vengono contenuti direttamente nel dato stesso.

Esistono diversi tipi di database NoSQL, ognuno utilizzato per contesti applicativi differenti:

- **Key-value**: struttura semplice, praticamente un dizionario. Il valore è il dato grezzo (JSON o altro). Sistema usato in memoria o con persistenza, permette di gestire grandi carichi di dati
- **Document-based**: basato su documenti. Ogni struttura ad albero può essere rappresentata da documenti XML o JSON. Simile a key-value, col valore che diventa un documento. Permette di aggregare diversi dati all'interno del documento, senza effettuare join. Lo schema è dinamico e si adatta ad ogni cambiamento
- **Columnar-based**: vengono salvati i dati in colonne piuttosto che in righe. Facilità operazioni su colonne come count, sum, ecc.
- **Graph-based**: basato su grafi, sfruttando tutte le loro proprietà. Segue i principi ACID

Information Retrieval

Una definizione di IR può essere: L'information retrieval è il **cercare materiale** (tipicamente documenti) **di natura non strutturata** (tipicamente testo) **che soddisfa la richiesta di informazione all'interno di una grande collezione** (tipicamente salvata in computers).

Si pensa tipicamente alle ricerche sul web, ma anche casi come e-mail, ricerche sui propri pc, librerie digitali e news rientrano in questo campo.

Alcune definizioni di base sono:

- **Documento**: file senza struttura
- **Collezione**: insieme di documenti
- **Query**: insieme di parole chiave per esprimere una richiesta di informazioni
- **Obiettivo**: ottenere i documenti con le informazioni rilevanti all'utente, aiutando l'utente a completare il task posto

La differenza tra **Data Retrieval** e **Information Retrieval** è che nel primo si gestiscono dati ben strutturati, nel secondo invece si gestiscono dati con linguaggio naturale, tipicamente senza struttura e ambigui. Nel primo caso si usano query SQL per ottenere risultati, nel secondo le query vengono effettuate in linguaggio naturale con utilizzo di parole chiave.

Comparare la query testuale col testo dei documenti e determinare se sono un buon match è il problema principale della IR. Infatti, effettuare un matching uno ad uno delle parole non basta. C'è quindi bisogno della nozione di rilevanza, sulla quale vengono basati diversi modelli.

Classic models

Boolean Retrieval

Il modello **Boolean Retrieval** permette di effettuare query con espressioni booleane. Vengono quindi combinate diverse parole chiave con operatori logici.

Ogni documento viene visto come un insieme di parole (**bag of words**). Questo modello cerca quindi di confrontare la condizione di partenza per capire se il documento rientra o meno nel risultato.
Viene introdotta quindi una "**term document incidence matrix**", che rappresenta con 1 il documento che contiene la parola, 0 viceversa.
In questo modo basta fare una operazione di bitwise AND e si ottiene il risultato richiesto.
Questo metodo però porta ad avere matrici di dimensioni troppo grandi e con dati sparsi. Un miglior modo per rappresentare il tutto è quello di registrare solo i documenti che posseggono la parola.

Per ogni parola *t*, si salva una **lista di documenti che la contengono** (ogni documento ha un proprio ID univoco). Vengono usate **liste dinamiche** per permettere modifiche successive ai documenti.

La fase di **preprocessing del testo** è divisa in vari step:

- **Tokenization**: sequenze di caratteri diventano token di parole
- **Normalizzazione**: si portano le parole in forma normale, portandole tutte in lower case e dovendo gestire casi ambigui
- **Lemmatization**: si trasformano ad esempio i verbi in forma base
- **Stemming**: si riducono le parole nella loro radice linguistica, rimuovendo le parti extra
- **Stop words**: vengono rimosse le stop words, che sono anche le parole più comuni. La tendenza però sta cambiando, per avere una maggiore comprensione del contesto

Vengono inoltre utilizzati dizionari per controllare diversi aspetti delle parole.

Usando solo query booleane, i documenti possono solo rientrare o meno nel risultato. Questo metodo è utile per utenti esperti, non utile per la maggior parte degli utenti.
Per questo motivo si utilizzano metodi di **Ranked Retrieval**, sistemi dove i documenti migliori per un risultato vengono ordinati in **ordine crescente di ranking**.
Quando un sistema produce un insieme di risultati rankati, vengono mostrati solo i primi *k* risultati. Il modello basato su ranking più famoso è il **Vector Space Model**.

Vector Space Model

Come nel caso delle bag of words, i documenti vengono rappresentati come vettori di "**term weights**". Le collezioni vengono rappresentate come matrici di pesi di termini.
Si ottengono quindi dei vettori **V-dimensionali**, dove i **termini** sono **assi dello spazio**.
I documenti sono punti o vettori in questo spazio. Sono altamente dimensionali, con milioni di dimensioni in casi come motori di ricerca.

L'idea principale è quella di **rendere** anche la **query** di ricerca un **insieme di vettori** nello spazio. In questo modo, il ranking può avvenire vedendo la vicinanza del documento alla query nello spazio. In questo caso la vicinanza indica una similitudine tra i due vettori, anche l'inverso della distanza.

L'obiettivo principale è quello di non usare modelli booleani, ma di usare modelli basati sul ranking dei documenti più rilevanti.

Si introduce ora il concetto di **Term Frequency**: la term frequency di una parola *t* in un documento *d* è definita come il numero di volte che la parola compare nel documento.
Vogliamo usare la term frequency per calcolare i punteggi di corrispondenza tra query e documenti. Per ottenere un ranking migliore, si utilizza il **coseno** della query e del documento.

I motori di ricerca si basano sul **concetto di rilevanza**. Come fanno però a capire cosa sta cercando l'utente? Le query possono essere ambigue.
In generale avviene una combinazione della query data dall'utente, dalla sua necessità di ottenere informazioni, e dallo scopo della query. La stessa query eseguita da due utenti diversi può portare a risultati totalmente diversi.

Osserviamo ora il concetto di **Google PageRank**.
Un sistema basato solo sulla term frequency non è sufficiente per gestire miliardi di documenti da rankare. Vengono quindi utilizzati complessi metodi per valutare la qualità, affidabilità e autorità delle pagine web:

- Metodi basati sulle proprietà della topologia della rete (Google PageRank)
- Metodi basati su diversi field boosting (titolo, sottotitolo e body hanno pesi differenti)
- Metodi basati sulla semantica e sulle proprietà dello spazio-tempo (novità di una pagina)

PageRank quindi è impostato come un ciclo dove le pagine dominanti rimangono tali all'infinito.

Un **Search Engine Result Page** (**SERP**) mostra di default solo i primi 10 risultati. Di questi risultati, solo i primi 5 sono visibli senza scrollare. Osserviamo quindi il comportamento degli utenti.
Tendenzialmente, la maggior parte degli utenti non controlla i risultati, ma preme sui primi risultati senza troppe considerazioni. Studi hanno confermato che anche invertendo la lista dei primi risultati, questa tendenza rimane.
I risultati in prima posizione vengono cliccati più di dieci volte rispetto ai risultati in sesta posizione. Questo è confermato anche da ulteriori test, sempre invertendo le liste dei risultati. C'è troppa fiducia da parte degli utenti verso le piattaforme di ranking.
Ogni click su un link ha un costo, questo implica che le aziende non prime nei ranking devono pagare N volte di più per ottenere la stessa visibilità.

Si introduce ora il concetto di **Terms Rarity**. Infatti, i termini più comuni sono meno informativi dei termini più rari. Si conclude quindi che i **termini rari** debbano avere un **peso maggiore** rispetto ai termini comuni. Si utilizza quindi la **inverse document frequency** (**idf**) per catturare questo fenomeno, che a sua volta quindi si basa sulla **document frequency** (**df**).

Si definisce **Document Frequency** come il numero di documenti che contengono la parola *t*. Misura quindi quanto è comune una parola rispetto a tutti i documenti presenti.
In questo caso, più una parola è comune, meno informativa sarà per noi. Introduciamo ora il concetto inverso.

Si definisce **Inverse Document Frequency** come l'inverso di Document Frequency, come **logaritmo in base 10 di (N / df)**, con N = numero di documenti nella collezione.
Si usa il logaritmo per arrotondare l'effetto dell'idf. Parole molto comuni come good o home possono avere una frequenza di 1 milione di volte più alta rispetto a parole più rare.
Si usa quindi il logaritmo per rimuovere il fenomeno esponenziale linguistico (**Zipf's law**), altrimenti le parole molto comuni avrebbero un idf pari a 0.

IDF non viene però utilizzato per query di una sola parola.
Inoltre, IDF non viene mai utilizzato da solo, ma sempre insieme a TF per una stima migliore del ranking.

Infatti, il peso **TF-IDF** di una parola è il **prodotto della sua term frequency con la sua inverse document frequency**. Questo è il **metodo più utilizzato** nei sistemi di information retrieval.
Il peso di una parola, quindi, aumenta all'aumentare della sua frequenza nei documenti (tf), ed aumenta all'aumentare della sua rarità (idf).

Probabilistic models

BM25

Il ranking nei modelli probabilistici viene dato dalla probabilità della rilevanza. La probabilità è stimata nella maniera più precisa possibile usando i dati disponibili.

Il modello **BM25**, spesso chiamato anche **Okapi BM25**, è un modello probabilistico sviluppato per essere sensibile a **Term Frequency**, **Term Rareness** (simile a **IDF**) e **lunghezza del documento**.

Con TF-IDF, il punteggio di similitudine può raggiungere valori molto alti, nel modello BM25 invece il **valore** è sempre tra **0 e 1**.

Okapi BM25 ha due parametri:

- **k1**: controlla quanto velocemente un aumento nella TF risulta nella sua saturazione. Il valore di default è 1.2. Con valori bassi si raggiunge una saturazione più rapida, e viceversa
- **b**: controlla quanto effetto la normalizzazione della lunghezza del documento ha. Un valore di 0 annulla completamente la normalizzazione, ed un valore di 1 lo normalizza completamente. Il default è 0.75

Language Model

Un **modello di linguaggio** statistico che assegna una probabilità ad una sequenza *P* di parole *m*, attraverso una distribuzione di probabilità.

Un language model separato è associato ad ogni documento di una collezione. I documenti vengono quindi rankati basandosi sulla probabilità della query *Q* nel language model del documento.

Due tipologie:

- **Unigram language model**: distribuzione di probabilità sulle parole di un linguaggio. Generazione del testo tirando le parole da un "secchio" basandosi sulla distribuzione di probabilità e sostituendole
- **N-gram language model**: alcune applicazioni usano language models a bigrammi o trigrammi dove la probabilità si basa sulle parole precedenti

Calcolare il punteggio di rilevanza usando modelli non booleani è molto pesante computazionalmente. I documenti vengono presi anche se non realmente utili per la query. Vengono quindi implementate **strategie per il controllo dell'efficienza**.

Una grande parte del lavoro delle CPU per ogni query di ricerca si sofferma per computare il punteggio di rilevanza. Di solito, gli utenti non hanno molta pazienza nell'attendere i risultati della query, quindi una **latenza bassa è necessaria**.
L'idea per migliorare questo processo è evitare di dare un punteggio ai documenti che non rientreranno nella top *k*.

Due tecniche diverse:

- **TAAT** (**Term at a time**): assegna un punteggio a tutti i documenti in maniera parallela, con una parola alla volta
- **DAAT** (**Document at a time**): viene dato un punteggio ad un documento alla volta, senza andare al successivo prima di averlo terminato

Entrambi hanno implicazioni su come l'indice viene strutturato e salvato.

Il termine "**safe ranking**" è usato per i metodi che garantiscono che i *K* documenti restituiti sono i *K* documenti col punteggio più alto.

La domanda è la seguente: **va bene usare un ranking non safe**? La risposta è **si**. Si ottiene un risultato abbastanza giusto che soddisfa comunque l'utente, andando a fare meno computazioni.

Due metodi "**non-safe**" sono:

- **Index "elimination"**: si considerano solo i termini con IDF alta (termini rari) e i documenti che contengono molte parole della query (scartando quelli con poche parole della query)
- **Champion lists**: per ogni parola, il punteggio per le i documenti più rilevanti è computata a priori, prima di una possibile query

Web Crawling and IR

Web crawling è il processo che raccoglie le pagine dal Web. L'obiettivo è quello di ottenere tutte le pagine web contenute in un'altra pagina web in maniera veloce ed efficiente, insieme alla struttura che le collega.

Un crawler parte da un URL di base e per ogni link all'interno di quella pagina lo inserisce in una coda, ripetendo il processo in maniera ricorsiva.

Un crawler deve rispettare due requisiti principali:

- **Robustness**: deve evitare le trappole, come prendere un numero infinito di pagine generate anche dinamicamente
- **Politeness**: deve rispettare le policy dei web server, che afferma quali pagine i crawler possono visitare. Si divide in esplicita (file robots.txt che afferma le regole del web server) ed implicita (anche senza specifiche, non bisogna andare su un sito frequentemente)

I link vengono inseriti all'interno della coda seguendo due strategie:

- **Breadth first**: vengono aggiunti tutti i link collegati alla pagina prima di passare alla pagina successiva. La copertura è ampia ma superficiale
- **Depth first**: vengono aggiunti i link della pagina fino a quello più profondo senza link, prima di aggiungere i successivi

Molte pagine sul web non sono interessanti. Viene quindi introdotto il concetto di popolarità, insieme alla nozione di rilevanza discussa precedentemente.

I sistemi di Web Information Retrieval basano il proprio ranking sulla combinazione di almeno due prove di rilevanza:

- Il grado di corrispondenza di una pagina: il **punteggio del contenuto**
- Il grado di importanza di una pagina: il **punteggio di popolarità**

Il primo può essere calcolato usando uno dei metodi visti fin ora, il secondo invece può essere calcolato da un'analisi della struttura degli hyperlink delle pagine indicizzate usando uno o più modelli per l'analisi dei link.

Alcuni modelli sono:

- **Simple link analysis**: si basa sull'idea del buono, sbagliato e sconosciuto. Un nodo buono non punterà mai ad un nodo sbagliato. Se punti ad un nodo sbagliato, sei un nod sbagliato. Se un nodo buono ti punta, sei buono
- **Citation analysis**: la frequenza di citazioni è una buona stima per la popolarità di un ricercatore
- **PageRank**: tecnica che assegna un punteggio numerico tra 0 ed 1 ad ogni nodo del grafo. Il PageRank del nodo dipende dalla struttura dei link del grafo. Il calcolo del PageRank corrisponde al trovare la distribuzione di probabilità stazionaria di una

random walk in un grafo. Una random walk è un caso particolare della Markov chain, dove lo stato successivo dipende dallo stato corrente.
Il PageRank viene utilizzato come uno di tanti fattori che determinano il punteggio finale di una pagina web in Google. Ci sono più di 200 fattori (detti anche segnali) che influenzano il risultato

- **Hyperlink-Induced Topic Search** (**HITS**): in risposta ad una query, piuttosto di una lista ordinate di pagine che corrispondono alla query, trova 2 insiemi di pagine collegate. Utile per topi campii piuttosto che query per pagine
- **Semantic Search**: effettua una ricerca su grafi su conoscenza strutturata piuttosto che una ricerca testuale tradizionale

IR Evaluation

L'obiettivo è quello di **misurare la rilevanza**. Troviamo tre elementi principali:

- Un benchmark di collezione di documenti
- Un benchmark di insiemi di query
- Una valutazione umana di rilevante o meno per ogni query ed ogni documento

Bisogna premettere che ogni utente trasforma una sua necessità in una query. Quindi la **rilevanza non** va **valutata sulla query**, ma sulla **necessità dell'utente**.

Il modo più semplice è quello **binario** (rilevante o meno). Va però valutato che l'insieme di query e documenti da valutare è enorme, portando a costi altissimi. Vengono quindi considerati soltanto top-k documenti di N sistemi di information retrieval, andando a limitare il numero di valutazioni da fare.

Il metodo più semplice è quello di permettere a piattaforme online di valutare il tutto, tagliando i costi. Questo però compromette leggermente i risultati ottenuti.

Per valutare al meglio l'efficacia di un sistema di IR ci sono due parametri da tenere in considerazione:

- **Precisione**: quanti documenti restituiti sono rilevanti per le informazioni richieste
- **Recall**: quanti documenti rilevanti sono stati restituiti dal sistema

I documenti **rilevanti** sono quelli collegati alla query, mentre quelli **restituiti** sono tutti quelli restituiti dal sistema.

La **precisione** viene calcolata dividendo i documenti rilevanti restituiti dal sistema con i documenti restituiti dal sistema.

Il **recall** viene calcolato dividendo i documenti rilevanti restituiti dal sistema con i documenti davvero rilevanti.

Si introduce ora il concetto di **Precisione@K** e **Recall@K**.
Viene posto un limite K del rank e vengono calcolate le percentuali solo in quel top K, ignorando il resto dei documenti.

La **Average Precision** è il calcolo aggregato per i risultati rankati. Viene calcolato come segue:

- Piuttosto di impostare un K arbitrario, ci fermiamo quando vengono recuperati tutti i documenti rilevanti
- Questo coincide con la prima K dove il valore di Recall@K è uguale ad 1
- Andiamo a calcolare la Precisione@K solo per le K dove i risultati rilevanti sono stati recuperati
- La media di questa precisione è la Average Precision

La **Mean Average Precision** invece serve per misurare la Average Precision di più query, andando a fare la media di tutte le Average Precision di ogni query.
Questo metodo assegna un peso uguale ad ogni query e sottintende che l'utente è interessato ad ottenere molti documenti per ogni query.

In tutti questi casi abbiamo usato nozioni di rilevanza binarie. Alcuni documenti possono essere meno rilevanti di altri, ma comunque rilevanti. Esistono quindi anche **metodi non binari**, anche se quelli binari rimangono comunque i più utilizzati per sistemi di IR.

Advanced methods for IR

In molte collezioni, lo stesso concetto può essere espresso usando parole diverse (**sinonimi**). Il problema è che la presenza di sinonimi ha un impatto sui sistemi di IR.

Una idea per risolvere questo problema è quella di **espandere la query** con tutti i sinonimi di una parola (car -> car cars automobile ecc.) e con parole annesse.
Sono stati quindi sviluppati diversi metodi automatici o semi automatici per l'espansione delle query, cercando di migliorare l'efficienza dei sistemi di IR.

Come trovare i termini relativi alla query per espanderla? Tramite **vocabolari** o **collezione di testo**.

Un primo metodo è chiamato **Thesaurus Query Expansion**. **Non** è molto **efficace** perché si basa sul controllare un vocabolario generale, senza considerare il contesto.

Piuttosto di usare questo metodo, parole chiave associate possono essere estratte da collezioni di testo. Vengono utilizzate diverse misure di **co-occurence** per trovare parole chiave associate:

- **Coefficiente di Dice**
- **Mutual Information**

- **Expected Mutual Information**
- **Pearson's Chi-squared**

Queste misure si basano su interi documenti o su parti più piccole di documenti. Per ora andiamo a considerare **solo interi documenti** per semplicità.

Coefficiente di Dice

Supponiamo di voler trovare parole collegare alla parola "fish". Come facciamo a calcolare il livello di collegamento di una seconda parola rispetto a "fish"?
Una misura può essere ottenuta dalla formula: **quante volte le parole appaiono insieme diviso quante volte le parole compaiono da sole**. Più alto il punteggio, più sono collegate le parole.

Una ulteriore formula afferma che date due parole *a* e *b*: $\mathbf{2n_{ab} / (n_a + n_b)}$, dove le due n di a e b sono il numero di documenti che contiene quelle parole, mentre n di ab è il numero di documenti che contiene entrambe le parole.

Mutual Information

Simile a Dice, **basato su probabilità**. Date due parole *a* e *b*: $\mathbf{\log (P(a,b) / P(a)P(b))}$, dove P(a) è la probabilità che la parola *a* sia presenta in una finestra di testo, P(b) è la probabilità che la parola *b* sia presenta in una finestra di testo, e P(a,b) è la probabilità che la parola entrambe le parole siano presenti nella stessa finestra di testo.

Questo metodo però **favorisce parole con bassa frequenza**.

Expected Mutual Information

Risolve il problema del favorire parole con bassa frequenza assegnando un peso alla mutual information. Si usa la seguente formula. Date due parole *a* e *b*: $\mathbf{P(a,b) \times \log (P(a,b) / P(a)P(b))}$.

Pearson's Chi-squared

Confronta il numero di co-occorrenze delle due parole con il numero previsto di co-occorrenze se le due parole fossero indipendenti. Il confronto viene poi normalizzato per il numero atteso.

Tra queste misure, la Mutual Information è quella che favorisce parole molto rare o scritte male. Chi-squared cattura anche parole inusuali. Il Coefficiente di Dice ed Expected Mutual Information sono le misure più adatte per l'espensione di query in sistemi di IR.

Un altro metodo per l'espansione di query è chiamato **Relevance Feedback** (**RF**), ed è basato sul feedback degli utenti.

L'idea di base è quella di far dare una query breve all'utente per poi restituire un primo insieme di risultati. L'utente identifica alcuni documenti restituiti come rilevanti (o non) ed il sistema restituisce una migliore rappresentazione delle informazioni basata proprio su quel feedback. Viene quindi restituito un nuovo insieme di risultati, si spera più rilevante.

Esiste anche uno **Pseudo Relevance Feedback**, anche detto **blind relevance feedback**, che funziona in maniera **automatica** senza che l'utente debba attivamente dare un parere. Questo viene ottenuto andando ad assumere che i top k documenti rankati siano rilevanti, e ricalcolando il Relevance Feedback basandosi su questo concetto.

È possibile poi usare Machine Learning per istruire modelli sull'imparare a rankare le pagine. Questa idea è stata analizzata solo di recente.
In questo caso, Relevance Feedback è un semplice esempio di come si possa utilizzare il machine learning nei sistemi di IR per migliorare le performance.

Data Analytics

Lo scopo dell'analisi dei dati è quello di estrarre informazioni dalle collezioni di dati. Le informazioni estratte possono essere informazioni riassunte o associazioni di informazioni. Tutto si basa sulla **statistica descrittiva**.

Vengono definiti i seguenti concetti di base.

- **Popolazione**: insieme di oggetti di interesse
- **Record**: tupla di valori che caratterizza un elemento della popolazione
- **Variabile**: nome del valore del record, ha significato e tipo comune per tutti i record della popolazione. Può essere di tipo numerico (discreto o continuo) o categorico (ordinale o nominale).
- **Dataset**: collezione di record che prende la forma di una tabella

La statistica descrittiva permette di trovare proprietà statistiche all'interno di una popolazione, basandosi su una sola variabile (**Indicatori di centralità** e **Indicatori di variazione**) o su più variabili (**Covarianza** e **Correlazione**).

Indicatori di centralità

Devono essere valori numerici e non categorici. In questo caso si parla di:

- **Media**: può essere usata per riempire campi mancanti senza influire sulla media generale. Anomalie influiscono molto su questo valore
- **Mediana**: indicatore robusto, permette di evitare le anomalie. Ha bisogno di dati ordinati

- **Moda**: permette di considerare anche valori categorici, usata anche per casi dove non c'è un ordine lineare nei possibili valori

Indicatori di variazione

- **Scarto quadratico**: misura la differenza tra ogni valore x e la media di tutti i valori. Più il valore è distante dalla media, più sarà alto questo valore
- **Varianza**: normalizzazione dello scarto quadratico (che viene compromesso in caso di un grande numero di valori) andando a dividere per il numero di valori presenti
- **Deviazione standard**: radice quadrata della varianza (per evitare problemi con valori grandi)

Covarianza

Misura la **forza della relazione tra due variabili** (la media dei prodotti delle deviazioni dei valori). Una covarianza positiva indica che le due variabili "deviano" nella stessa direzione (direttamente proporzionali), una covarianza negativa indica che le due variabili "deviano" in direzioni opposte (inversamente proporzionali). Viene **molto influenzata dall'unità di misura**.

Correlazione

Misura la **forza della relazione tra due variabili con valori tra -1 ed 1**. Risolve il problema della covarianza, tenendo in considerazione la deviazione standard delle due variabili.
Con correlazione vicina a -1 le due variabili vanno in direzioni opposte (inversamente proporzionali). Con correlazione vicino a 1 le due variabili vanno nella stessa direzione (direttamente proporzionali). Con correlazione vicino a 0 non c'è nessuna correlazione.

Data Warehouse

I database visti fin ora sono operazionali, ovvero permettono aggiornamenti in tempo reale, hanno bisogno di query complesse per effettuare analisi ed esplorazione dei dati, hanno bisogno di competenze tecniche sullo schema e non contengono dati storici.

Questo tipo di database è molto utile nel contesto software, ma in un contesto business i dati devono essere fonte di informazioni, permettendo di prendere decisioni più precise.

Spesso si punta ad unire diversi dati provenienti da più fonti. Fare questa cosa con database operazionali è estremamente difficile, avendo diverse definizioni e contenuti.

Nascono quindi i **Data Warehouse**, collezioni di dati non volatili, integrati, specifici e varianti nel tempo che aiutano nel processo decisionale. Più nel dettaglio:

- **Specifici**: vengono presi come riferimento uno o più campi di analisi in base alle esigenze

- **Integrati**: tutti i dati presenti all'interno vengono dall'unione di diversi sistemi di dati
- **Non volatili**: vengono accumulati dati da database operazionali per lunghi periodi di tempo. Le modifiche non sono permesse (solo dati obsoleti possono essere rimossi)
- **Varianti nel tempo**: viene tenuta traccia di come i dati variano nel tempo, per osservare l'evoluzione dei dati stessi

Il paradigma relazionale, quindi, non è adatto per i Data Warehouse, dove c'è bisogno di prestazioni ottime per query complesse. C'è necessità di sistemi più comprensibili per utenti non esperti. Viene quindi usato il paradigma del **multidimensional modeling** (anche detto **dimensional modeling**, **DM**).

Il **multidimensional modeling** permette di vedere i dati come fatti collegati alle dimensioni.

- Un fatto si concentra sull'analisi
- Le misure quantificano i fatti (tipicamente misure numeriche)
- Le dimensioni sono usate per osservare le misure da più prospettive (ad esempio dimensioni temporali o di luogo). Le dimensioni includono attributi a livello gerarchico, dove gli attributi permettono di ottenere diversi livelli di dettaglio (anno -> quarter -> mese)
- L'aggregazione di diverse misure avviene ogni volta che un livello gerarchico viene attraversato (spostandosi da mese ad anno i valori delle vendite vengono aggregati da mesi ad anni)

Il multidimensional modeling è un modello concettuale. Al **livello logico** invece, viene rappresentato da tabelle collegate in schemi a **star** o a **snowflake**. Questi schemi mettono la tabella dei fatti al centro dello schema, collegandola a tutte le dimensioni della tabella:

- **Star**: lo schema usa un'unica tabella per ogni dimensione, anche in presenza di gerarchie (tabelle non in forma normale)
- **Snowflake**: lo schema usa molte tabelle normalizzate in base alle gerarchie

Per riempire un DW si usa il metodo **ETL**.
ETL significa **Extraction, Transformation e Loading**. Processo cruciale per il riempimento dei DW. Vengono prima estratti i dati da diverse fonti.
Poi i dati vengono puliti e trasformati per entrare nel modello del DW.
Successivamente vengono caricati all'interno del DW.
Questo step copre circa **l'80% del costo di un DW**.

Vediamo gli step nel dettaglio:

- **Extraction**: tutti i dati rilevanti vengono presi dalle loro fonti. Una estrazione statica viene fatta la prima volta per popolare il DW (snapshot del db operazionale). Successivamente avvengono estrazioni incrementali per aggiornare il DW con tutte le modifiche ai dati. I dati da estrarre vengono scelti in base alla loro qualità

- **Cleaning**: lo scopo è quello di migliorare la qualità dei dati. Vengono rimossi duplicati, dati inconsistenti o mancanti, o dati semplicemente errati
- **Transformation**: vengono convertiti i dati dal formato operazionale a quello del DW. Processo molto lungo, dove bisogna passare dallo schema operazionale precedente al nuovo del DW

Si introduce ora il concetto di sistema **OLTP** (**OnLine Transaction Processing**). Indica tutti quei sistemi (database relazionali) che garantiscono accesso veloce ai dati, con transazioni sicure e controlli sulla concorrenza.

Per l'analisi dei dati c'è bisogno di un nuovo paradigma, ovvero i sistemi **OLAP** (**OnLine Analytical Processing**). Sistemi che si concentrano su query analitiche, con schemi che non rispettano le forme normali, con alti carichi di query. Questi sistemi hanno portato ai DW.

I Data Warehouse possono quindi essere visti come grandi repository che prendono dati da diverse fonti, con aggiornamenti offline, che seguono un modello multidimensionale, che supportano query OLAP e che vengono implementi con schema a star o snowflake.
Ci andremo ora a soffermare sulla loro architettura, design e query.

Architettura
Viene usata un'architettura a **Tier**:

- **Data sources**: non un proprio livello dei DW, più una collezione di dati di diversa tipologia
- **Backend tier**: composto da tool per ETL e da una data staging area (dove tutte le integrazioni e le trasformazioni sui dati vengono processate prima di essere caricati)
- **Data warehouse tier**: composto da un enterprise data warehouse e/o diversi data mart (raccoglitori di dati specifici) ed un metadata repository che contiene le informazioni sui dati contenuti all'interno del DW (diviso in metadati business e tecnici)
- **OLAP tier**: composto da un OLAP server che permette di eseguire viste multidimensionali sui dati, a prescindere dal tipo di dati salvati all'interno del DW
- **Frontend tier**: permette di visualizzare ed analizzare i dati, usando i servizi messi a disposizione dall'OLAP. Permette visualizzazioni interattive dei dati, con tool statistici e di data mining

Design
Viene utilizzato il **modello multidimensionale** discusso precedentemente. I dati vengono visti in uno **spazio n-dimensionale** (un **data cube**), composto da **dimensioni** e **fatti**.

Una **istanza** di una dimensione è detta **membro**. Un data cube contiene molte dimensioni. Al suo interno c'è anche una suddivisione gerarchica, che permette di vedere i dati con diversa granularità.

Diverse aggregazioni di misure cambiano il livello di astrazione al quale i dati nel cubo vengono visualizzati.
Le misure possono essere additive, semi additive e non additive.
Inoltre, le misure possono essere distributive, algebriche e olistiche.

OLAP Querying
Esistono diverse tipologie di **operazioni OLAP** che possiamo applicare al cubo di dati:

- **Roll-up**
- **Drill-down**
- **Sort**
- **Pivot**
- **Slice**
- **Dice**

Data Mining

L'obiettivo è quella di **estrarre pattern e conoscenza** da **grandi quantità di dati**.

Per **Pattern** si parla di regolarità nei dati, che si ripetono in maniera predicibile all'interno di osservazioni. Alcuni esempi di pattern sono: latte e cereali vengono spesso acquistati insieme, oppure: in molti casi il prezzo delle case in euro è circa 2500 volte i suoi metri quadri.

Una volta scoperto un pattern, si cerca di capire la logica che c'è dietro, in modo da descrivere il pattern e poter predire un fenomeno del genere.

La **conoscenza**, quindi, è il capire uno specifico argomento. Esempio: come i prezzi delle case vengono definiti partendo dai metri quadri.

Per trovare pattern e conoscenza, bisogna prima partire da diverse **osservazioni**.
Nel data mining le **osservazioni** vengono fatte in forma di **esempi di dati omogenei**. Grandi quantità di dati sono necessarie per estrarre pattern o conoscenza in maniera significativa.
Gli step del data mining sono i seguenti:

1. **Definire il problema**
2. **Identificare i dati necessari**
3. **Preparare e pre-processare**
4. **Modellare l'ipotesi**
5. **Train e test**
6. **Verificare e deployare**

Ci soffermeremo solo sui punti **3-4-5**, dando per scontato che il problema è stato definito, i dati sono stati già definiti e la fase di verifica e deploy è specifica ai diversi casi d'uso.

Si introduce quindi il **Machine Learning**, che è al centro del data mining, perché permette di **passare dall'astrazione dei dati ai pattern**. L'idea di base è quella di basarsi su molte osservazioni per ottenere una funzione con un obiettivo specifico.
Viene utilizzato il ML e non lavoro umano per via della grande quantità di dati da analizzare e delle relazioni non lineare tra le variabili.

Basandosi sulla statistica, i pattern e la conoscenza estratta non sono mai 100% precisi; infatti, il ML si basa sul principio della **PAC theory** (**Probably, Approximately Correct**).
Il modello, quindi, dovrebbe essere preciso con alte probabilità, con bassi errori di generalizzazione (deve capire bene il caso generale e essere preciso con casi mai visti prima).

Vengono utilizzati metodi **Supervised**: dato un training set di dati all'algoritmo di ML, diciamo all'algoritmo cosa dovrebbe essere l'output per quel set di esempi. In base al tipo di variabile possiamo avere Regressione (output con variabili di tipo numerico) o Classificazione (output con variabili classe, classificare qualcosa).

Esistono anche metodi **Unsupervised**: non vengono dati valori per l'output, o non c'è proprio una variabile in output. Serve anche per trovare pattern o relazioni nei dati. Alcuni metodi possono essere Clustering (si divide il dataset in gruppi, dove i dati simili tra loro appartengono allo stesso gruppo) e Association rules (serve per scoprire relazioni tra variabili).

Passando ora al **pre-processing dei dati**. Step molto utile perché i dati sono spesso incompleti, contengono errori o valori outlier, o semplicemente inconsistenti.
Bisogna quindi fare **data cleaning** (gestire valori mancanti, rimuovere outliers, risolvere inconsistenze), **data integration** (unire diversi fonti di dati) e **data transformation** (valori estremi vanno normalizzati).

I consigli dati in precedenza negli appunti (parte dei Data Warehouse) per gestire questa fase valgono anche in questo contesto.

Tutti i dati che passano la fase di pre-processing verranno **dati in pasto** ai vari algoritmi di ML.