Ingegneria dei Dati:
Motore di Ricerca per Tabelle HTML

Mattia Micaloni, Carlo Proserpio, Alessandro Di Girolamo

Project repository: https://github.com/Xhst/de-projects/tree/main/project-3

Keywords: Ingegneria dei Dati, Apache Lucene, Motore di Ricerca, Indicizzazione, Tokenizzazione, Embedding, Database vettoriali, Ricerca su tabelle, Ricerca semantica

1. Introduzione

In questa relazione vengono presentate alcune soluzioni per la realizzazione di un motore di ricerca sulle tabelle degli articoli scientifici di *arXiv*[1]. Le tabelle di interesse sono contenute in file in formato *JSON* derivati dal Progetto 1[2] . Questi file, per ogni articolo, contengono tutte le tabelle presenti nel paper insieme alle loro informazioni rilevanti come didascalie, paragrafi che citano le tabelle e note a piè di pagina.

2. Analisi e pulizia dei dati

L'estrazione delle tabelle e delle relative informazioni dai file HMTL è per lo più risultata pulita. Nonostante questo, un numero non indifferente di errori e incongruenze erano presenti nei dati del corpus. Al fine di verificare la validità e la pulizia dei dati, è stata condotta una piccola analisi riassunta nella tabella 1.

	File HTML	File Tabelle
Numero totale	9372	9490
Numero duplicati	203	175
Numero malformati	137	457

Tabella 1: Numero di file duplicati o malformati sul totale dei file HTML e JSON.

Quando scriviamo *Numero malformati* intendiamo file il cui contenuto era corrotto, irrilevante oppure non rispettava la struttura definita dai requisiti del Progetto 1 (per

[1] https://arxiv.org/
[2] https://github.com/Xhst/de-projects/tree/main/project-1

esempio dei JSON con campi mancanti o aventi un tipo differente da quello richiesto). Per *Numero duplicati* si intendono i file con nome diverso ma riferenti lo stesso articolo (per esempio *2024.39489* e *arvix_ article_ 2024.39489*).

3. Soluzioni per la Realizzazione del Motore di Ricerca

Nello svolgimento del progetto sono state esplorate diverse soluzioni per la realizzazione di un motore di ricerca orientato a restituire risultati scientifici in maniera efficiente. Queste soluzioni possono essere suddivise in due categorie principali.
La prima categoria prevede l'utilizzo di *Apache Lucene*, una piattaforma consolidata per la gestione e la ricerca di documenti testuali.
Lucene si basa su tecniche tradizionali di information retrieval, come il calcolo di rilevanza tramite il modello *BM25*. Questo approccio sfrutta algoritmi basati su metriche come il *TF-IDF* e consente un'implementazione efficiente e personalizzabile per le operazioni di indicizzazione e ricerca.
Sebbene non faccia uso diretto di tecnologie avanzate come il machine learning, *Lucene* è estremamente scalabile e modulare, risultando particolarmente adatto per contesti dove l'efficienza e la precisione sono parametri critici.
La seconda categoria si focalizza invece sull'adozione di modelli di linguaggio avanzati, tra cui: *BERT, MPNet, T5-Large, MiniLM*.
Questa soluzione integra una varietà di modelli, ognuno con caratteristiche specifiche che comportano vantaggi e svantaggi. L'adozione di tali modelli consente di ottenere, almeno in linea teorica, una comprensione semantica più profonda delle query e dei documenti rispetto agli approcci tradizionali. In questo contesto, il motore di ricerca utilizza i modelli per generare rappresentazioni vettoriali di documenti e query, calcolandone la similarità tramite metriche come la similarità coseno.

3.1 Soluzione con Apache Lucene

La soluzione basata su Apache Lucene si concentra sull'estrazione e l'indicizzazione delle informazioni chiave presenti nei file JSON del corpus.
In particolare, il processo prevede l'estrazione dei seguenti elementi: l'ID dei file, gli ID delle tabelle contenute nei file, le didascalie delle tabelle, i paragrafi che fanno riferimento a tali tabelle e le note a piè di pagina.
Questi dati vengono successivamente indicizzati, creando una struttura che permette di eseguire query mirate per recuperare in modo efficiente informazioni specifiche, più precisamente al fine di cercare tabelle con risultati scientifici pertinenti.
L'obiettivo di questa implementazione è migliorare la capacità del motore di ricerca di identificare e restituire risultati pertinenti, basati su relazioni e citazioni tra i vari elementi del corpus.
Per ogni campo indicizzato è stato utilizzato un analizzatore opportuno.

3.1.1 Analizzatore per il Codice Paper e Codice Tabella

Questo analizzatore è stato progettato per indicizzare il codice univoco di ciascun paper di ogni tabella.

```
CustomAnalyzer.Builder idAnalyzerBuilder = CustomAnalyzer.builder()
        .withTokenizer(KeywordTokenizerFactory.class)
        .addTokenFilter(TrimFilterFactory.class);
```

KeywordTokenizerFactory tokenizza il testo come un unico token, utile in quanto il codice del paper è univoco, mentre *TrimFilterFactory* filtra i token rimuovendo gli spazi bianchi superflui alla fine e all'inizio di quest'ultimo.

3.1.2 Analizzatore per il Didascalia e Referenze

L'analizzatore applica una serie di filtri per ridurre il testo da indicizzare, concentrandosi sui concetti principali espressi.

```
CustomAnalyzer.Builder textAnalyzerBuilder = CustomAnalyzer.builder()
         .withTokenizer(StandardTokenizerFactory.class)
         .addTokenFilter(LowerCaseFilterFactory.class)
         .addTokenFilter(TrimFilterFactory.class);
         .addTokenFilter(PorterStemFilterFactory.class)
         .addTokenFilter(StopFilterFactory.class)
         .addTokenFilter(RemoveDuplicatesTokenFilterFactory.class)
         .addTokenFilter(ASCIIFoldingFilterFactory.class)
```

Utilizza filtri come *StandardTokenizerFactory*, *LowerCaseFilterFactory*, *TrimFilterFactory* che sono molto comuni. Utilizza inoltre i seguenti filtri più specicfici: *PorterStemFilterFactory* che applica lo stemmer di Porter ai token, un algoritmo che riduce le parole alla loro radice; *StopFilterFactory* che rimuove le stop-word (es.: "the", "and", "in"), le quali non sarebbero rilevanti ai fini della ricerca; *RemoveDuplicatesTokenFilterFactory* che rimuove le parole duplicate consecutive al fine di prevedere e gestire, ad esempio, eventuali errori commessi dall'autore durante la scrittura e *ASCIIFoldingFilterFactory* utilizzato per rimuovere gli accenti dai token.

3.1.3 Analizzatore per la Tabella

Progettato per le tabelle, questo analizzatore utilizza la seguente configurazione.

```
CustomAnalyzer.Builder tableAnalyzerBuilder = CustomAnalyzer.builder()
         .withTokenizer(StandardTokenizerFactory.class)
         .addTokenFilter(LowerCaseFilterFactory.class)
         .addTokenFilter(TrimFilterFactory.class);
         .addTokenFilter(ASCIIFoldingFilterFactory.class)
```

3.2 Soluzione basata su Modelli di Linguaggio

La soluzione che sfrutta i Modelli di Linguaggio (LM) per l'estrazione e il confronto dei dati dai file JSON adotta un approccio avanzato, valorizzando la capacità di questi modelli di comprendere il contesto semantico del testo. L'intero processo si articola in diverse fasi principali:

3.2.1 Estrazione dei dati dai file JSON

In una prima fase, vengono estratti dai file JSON i seguenti elementi: gli ID dei file, gli ID delle tabelle, le didascalie delle tabelle, i paragrafi che fanno riferimento a ciascuna tabella e le note a piè di pagina. I campi estratti vengono poi opportunamente elaborati (parsing) per ridurre il rumore e ottimizzare la dimensione dei dati, in particolare per rispettare i limiti sul numero massimo di token imposti dai modelli di embedding.

3.2.2 Creazione degli embedding

La seconda fase consiste nella generazione degli embedding, utilizzando una selezione di modelli preaddestrati. Ciascun modello è stato addestrato su differenti tecniche e corpus, offrendo caratteristiche uniche per la rappresentazione semantica dei dati. Di seguito è riportata una breve descrizione dei modelli utilizzati:

- **BERT-base-uncased**: È la versione di base di BERT, preaddestrata su un vasto corpus di testo in inglese. Il modello uncased non fa distinzione tra maiuscole e minuscole. (110 milioni di parametri, 12 strati).
- **DistilBERT-base-uncased**: Una versione più compatta e leggera di BERT, ottenuta tramite un processo di distillazione, che riduce le dimensioni del modello mantenendo la maggior parte delle prestazioni di BERT. È progettato per essere più veloce e meno esigente in termini di risorse computazionali, pur conservando una buona accuratezza (66 milioni di parametri, 6 strati).
- **SciBERT-uncased**: Una variante di BERT preaddestrata su un corpus di testi scientifici, in particolare articoli e documenti accademici. Questo modello è ottimizzato per il dominio scientifico e include un vocabolario specifico che gli consente di trattare meglio il linguaggio tecnico e specialistico (110 milioni di parametri, 12 strati).
- **all-Mpnet-base-v2**: è uno dei migliori della famiglia Sentence Transformers per la ricerca semantica, progettato per generare rappresentazioni dense dei testi con alta accuratezza. È basato su MPNet, un'architettura più avanzata rispetto a BERT (110 milioni di parametri, 12 strati).

- **sentence-t5-large**: è una versione della famiglia T5 (Text-to-Text Transfer Transformer), adattata per generare rappresentazioni dense di testi con Sentence Transformers. Offre una delle migliori performance nei task di similarità testuale e ricerca semantica; ottimo per NLP (770 milioni di parametri, 24 strati)

- **all-MiniLM-L6-v2**: uno dei modelli più popolari e leggeri della famiglia Sentence Transformers, progettato per compiti di similarità testuale e ricerca semantica con un ottimo compromesso tra velocità, accuratezza e risorse. Basato su MiniLM (un modello derivato da BERT) (22 milioni di parametri, 6 strati)

- **all-MiniLM-L12-v2**: versione avanzata della famiglia MiniLM ed è progettato per compiti di similarità semantica, ricerca semantica e altri task NLP che richiedono la rappresentazione di testi sotto forma di vettori. È una versione migliorata rispetto a *all-MiniLM-L6-v2*, con più strati, maggiore numero di parametri e, di conseguenza, una performance migliore in compiti complessi grazie all'aumento della profondità della rete (33 milioni di parametri, 12 strati).

Come abbiamo detto, le informazioni disponibili per le tabelle sono: le didascalie, le note a piè di pagina, i paragrafi che fanno riferimento a tali tabelle, oltre che le tabelle stesse. Per ogni modello abbiamo definito diversi metodi per costruire gli embedding: il primo, più semplice, fa l'embedding soltanto delle informazioni contenute nella tabella; nel secondo abbiamo aggiunto le informazioni della didascalia; nel terzo e quarto metodo abbiamo integrato, insieme alle informazioni di tabella e didascalia, anche i testi dei paragrafi. La differenza del quarto metodo rispetto al terzo sta nel come è costruito l'embedding, invece di fare un unico embedding sul testo aggregato dei tre campi fa tre embedding separati, uno per ciascun campo, per assegnargli un peso facendo una combinazione lineare, come mostrato nell'equazione 1. Abbiamo scelto di non utilizzare le note a piè di pagina poiché poco frequenti e spesso estratte in maniera errata. Invece, per aggiungere un minimo di contesto sull'argomento, abbiamo aggiunto agli embedding, di tutti e quattro i metodi, il titolo dell'articolo.

$$E = w_t * E_t + w_r * E_r + w_d * (E_d + title_p) \tag{1}$$

Equazione 1: Funzione di aggregazione degli embedding. E e w sono rispettivamente l'embedding e il peso associato, mentre le etichette t, r, d sono rispettivamente tabella, referenze della tabella e didascalia della tabella.

Una considerazione generale è che la scelta di concatenare i tre campi è pensata al fine di dare la possibilità al modello di linguaggio di capire il contesto, riuscendo a determinare la semantica più efficientemente. Nonostante ciò, troppo contesto potrebbe distrarre il modello dalla specificità richiesta dalla query.

3.3 Indicizzazione e calcolo della similarità

Dopo aver creato gli embedding, per ciascun modello e metodo, abbiamo salvato i vettori su degli indici di un database vettoriale per poter fare ricerche efficienti. Il database vettoriale utilizzato è Milvus, un sistema open-source progettato per l'archiviazione e la ricerca di vettori ad alte prestazioni.
Gli indici sono stati creati con IVF_FLAT (Inverted File with Flat), un algoritmo di indicizzazione e ricerca vettoriale utilizzato nella ricerca di similarità. Per limitare la ricerca ai cluster più promettenti (utilizza *k-means* per il clustering), e *FLAT*, per la ricerca all'interno di quei cluster al fine di ottenere una maggiore precisione.
Questo approccio, generalmente, ottimizza le prestazioni riducendo la quantità di dati da analizzare, mantenendo al contempo un buon livello di accuratezza nei risultati. La ricerca nell'indice è guidata dal calcolo della similarità, nel nostro caso coseno, rispetto ad un vettore rappresentante la query.

3.3.1 Vettorizzazione della query

Quando l'utente invia una query, questa viene anch'essa passata attraverso il modello in uso, generando un embedding per tale query. L'obiettivo è quello di rappresentare la query in uno spazio vettoriale che consenta di confrontarla in modo significativo con gli embedding delle tabelle e degli altri elementi del corpus.

3.4 Soluzione Ibrida

Abbiamo deciso di testare anche un approccio ibrido che comprende l'utilizzo di Lucene, in particolare la ricerca sugli articoli implementata nel Progetto 2[3], per ottenere un primo filtraggio dei documenti rispetto all'argomento richiesto dalla query. Successivamente, il sottoinsieme di articoli selezionati viene passato ai modelli che lavorano sulle tabelle.

4. Valutazione

In questa sezione verranno descritte tutte le azioni svolte per valutare il sistema e i relativi risultati (opportunamente commentati) inerenti alle query sottomesse.

4.1 Metriche utilizzate

La valutazione è stata effettuata mediante quattro metriche comunemente adottate nell'ambito della valutazione dei sistemi di Information Retrieval: *Mean Reciprocal Rank*

[3] https://github.com/Xhst/de-projects/tree/main/project-2

(MRR), *Normalized Discounted Cumulative Gain* (NDCG) e *Mean Average Precision* (MAP).

- **Mean Reciprocal Rank**: La metrica MRR è utilizzata per valutare la posizione del primo risultato rilevante per ogni query. È particolarmente utile quando l'obiettivo principale è ottenere il risultato corretto quanto più in alto possibile nella lista dei risultati. La formula per calcolare l'MRR è la seguente:

$$MRR = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{rank_i}$$

  dove Q è il numero totale di query e $rank_i$ è la posizione del primo risultato rilevante per la query i.

- **Normalized Discounted Cumulative Gain**: La metrica NDCG è una misura più complessa che tiene conto non solo della posizione dei risultati rilevanti, ma anche della rilevanza relativa di ogni risultato in base alla sua posizione nella lista. La formula generale per calcolare il Discounted Cumulative Gain (DCG) di una query è:

$$DCG@K = \sum_{i=1}^{K} \frac{rel_i}{log_2(i+1)}$$

  dove K è il numero di risultati considerati, rel_i è la rilevanza del risultato i (valutazione numerica) e $log_2(i+1)$ è un fattore di sconto che penalizza i risultati meno rilevanti a mano a mano che si scende nella lista.
  Il NDCG normalizza il DCG dividendolo per il DCG ideale (IDCG, ottenuto mediante il ground truth), che rappresenta il massimo punteggio possibile per quella query (ovvero quando i risultati sono ordinati perfettamente in base alla loro rilevanza):

$$NDCG@K = \frac{DCG@K}{IDCG@K}$$

- **Precision at K (P@K)**: Questa metrica valuta la precisione dei primi K risultati restituiti dal sistema, calcolando la proporzione di risultati rilevanti tra i primi K. È particolarmente utile quando si vuole limitare l'attenzione alle prime posizioni della lista. La formula per P@K è:

$$P@K = \frac{1}{K} \sum_{i=1}^{K} rel_i$$

  dove rel_i assume valore 1 se il risultato i è rilevante, 0 altrimenti.

- **Mean Average Precision (MAP)**: La metrica MAP fornisce una misura aggregata delle performance calcolando la precisione media per ogni query e quindi effettuando una media su tutte le query. La precisione media per una query è calcolata

come:

$$AP = \frac{1}{R} \sum_{k=1}^{n} P(k) \cdot rel_k$$

dove R è il numero totale di risultati rilevanti per quella query, $P(k)$ è la precisione calcolata considerando i primi k risultati e rel_k indica se il risultato in posizione k è rilevante (1) o no (0). Il MAP è quindi:

$$MAP = \frac{1}{Q} \sum_{i=1}^{Q} AP_i$$

dove Q è il numero totale di query e AP_i è la precisione media per la query i.

4.2 Costruzione del Ground Truth

Per valutare le prestazioni dei motori di ricerca è stato necessario creare manualmente un ground truth. Sono state effettuate delle query al sistema utilizzato per il motore di ricerca del Progetto 2, abbiamo selezionato alcuni articoli tra diversi argomenti che utilizzassero diversi dataset e metriche di valutazione. Abbiamo deciso di selezionarli in base anche alla possibile difficoltà nell'interpretare le tabelle (con un contenuto non facilmente interpretabile senza un contesto aggiuntivo oltre a quello dato dalla sola tabella) Sono state scelte un totale di 56 tabelle contenute in 16 articoli su argomenti diversi. Abbiamo scelto 6 query su queste tabelle e per ciascuna è stato definito un ranking fino alla quindicesima posizione. Il tempo totale stimato che è stato speso per la creazione di questo ground truth è di 11 ore. Le tempistiche comprendono ogni passo della creazione, dalla scelta delle tabelle, al ranking finale delle tabelle sulle query.

4.3 Query

In questa sezione vengono presentate le query di test utilizzate per valutare le prestazioni del sistema di ricerca implementato. Le query sono state progettate per rappresentare scenari realistici e diversificati, simulando richieste di informazioni scientifiche specifiche.

1. Query 1: `NDCG on MovieLens dataset`
2. Query 2: `Recommender systems Recall on dataset Goodbook`
3. Query 3: `Recommender systems MRR`
4. Query 4: `Deep Learning dataset Apple Flower`
5. Query 5: `Deep Learning GPT-3 precision f1`
6. Query 6: `Deep Learning GPT-3 precision f-measure`

4.4 Risultati

Nelle tabelle 2 e 3 sono riportati i risultati delle metriche. Per Lucene sono stati riportati i valori delle metriche utilizzando l'algoritmo BM25, mentre per i modelli abbiamo i quattro metodi descritti nella sezione 3.2.2: con **T** riportiamo il metodo che usa solo la tabella, con **T+C** il metodo che usa tabella e didascalia, con **T+C+R** il metodo che usa tabella, didascalia e referenze e con **W** il metodo che usa la combinazione lineare.

		Standard			
		MRR	***NDCG@5***	***NDCG@15***	***MAP@15***
Lucene BM25		0.52	0.93	0.68	0.74
Bert	***T***	0.1	0.63	0.45	0.5
	T+C	0.1	0.59	0.42	0.47
	T+C+R	0.11	0.56	0.39	0.46
	W	0.17	0.69	0.5	0.55
Distil Bert	***T***	0.07	0.67	0.48	0.53
	T+C	0.11	0.61	0.44	0.49
	T+C+R	0.12	0.46	0.32	0.38
	W	0.13	0.7	0.5	0.54
SciBert	***T***	0.03	0.5	0.36	0.38
	T+C	0.06	0.46	0.33	0.38
	T+C+R	0.06	0.34	0.24	0.29
	W	0.09	0.5	0.36	0.39
MPNet base	***T***	0.22	0.91	0.67	0.71
	T+C	0.25	0.8	0.59	0.67
	T+C+R	0.48	0.79	0.61	0.66
	W	0.28	0.9	0.67	0.7
T5 Large	***T***	0.17	0.997	0.74	0.75
	T+C	0.31	0.92	0.66	0.7
	T+C+R	0.3	0.994	0.76	0.79
	W	0.34	0.99	0.73	0.75
MiniLM L6	***T***	0.24	0.96	0.73	0.75
	T+C	0.45	0.83	0.59	0.71
	T+C+R	0.43	0.93	0.69	0.71
	W	**0.6**	0.97	0.71	0.74
MiniLM L12	***T***	0.32	**1.00**	0.78	0.8
	T+C	0.36	0.8	0.59	0.65
	T+C+R	0.29	**1.00**	**0.8**	**0.82**
	W	0.36	0.98	0.75	0.76

Tabella 2: Valori di MRR, NDCG@5, NDCG@15 e MAP@15 per Lucene BM25 e per modelli. In arancione sono riportati i valori migliori di Lucene, in verde i valori migliori.

		Ibrido			
		MRR	***NDCG@5***	***NDCG@15***	***MAP@15***
Lucene BM25		0.75	0.92	0.71	0.74
Bert	***T***	0.1	0.62	0.46	0.5
	T+C	0.18	0.6	0.43	0.5
	T+C+R	0.21	0.56	0.4	0.46
	W	0.12	0.57	0.4	0.46
Distil Bert	***T***	0.11	0.68	0.51	0.54
	T+C	0.15	0.68	0.49	0.54
	T+C+R	0.16	0.51	0.36	0.42
	W	0.10	0.58	0.42	0.48
SciBert	***T***	0.05	0.51	0.37	0.38
	T+C	0.07	0.46	0.33	0.36
	T+C+R	0.05	0.38	0.27	0.29
	W	0.06	0.52	0.38	0.44
MPNet base	***T***	0.26	0.89	0.67	0.68
	T+C	0.37	0.84	0.62	0.66
	T+C+R	0.45	**0.93**	0.69	0.74
	W	0.24	0.9	0.64	0.65
T5 Large	***T***	0.28	0.81	0.64	0.67
	T+C	0.57	**0.93**	0.71	0.76
	T+C+R	0.38	0.9	0.71	**0.78**
	W	0.55	0.86	0.65	0.68
MiniLM L6	***T***	0.33	0.89	0.7	0.74
	T+C	0.53	0.84	0.61	0.65
	T+C+R	0.58	0.9	**0.74**	0.76
	W	0.53	0.89	0.68	0.68
MiniLM L12	***T***	0.23	0.89	0.67	0.66
	T+C	0.36	0.83	0.6	0.65
	T+C+R	0.25	0.89	0.69	0.65
	W	0.32	0.9	0.69	0.7

Tabella 3: Valori di MRR, NDCG@5, NDCG@15 e MAP@15 per Lucene BM25 e per modelli utilizzando il sistema ibrido. In arancione sono riportati i valori migliori di Lucene, in verde i valori migliori.

Lucene e MiniLM-L6 risultano i migliori per quanto riguarda i valori di MRR. Per la NDCG risulta che i modelli T5 Large, MiniLM L6 e MiniLM L12 riescono a restituire valori più rilevanti rispetto a Lucene. In particolare si può notare come MiniLM 12 sia riuscito a raggiungere un valore massimo sia usando solo l'embedding T, sia con l'embedding T+C+R. Questi tre modelli sono valide alternative all'utilizzo di Lucene.