Relazione Homework2 ID - HZ

Github url: https://github.com/haowen3012/ID-Homerworks/tree/main/homework2

Url notion per visualizzare meglio la relazione: https://stump-rabbit-25b.notion.site/Relazione-Homework2-ID-HZ-13ac078832ad80e794cddabe7a6c1221

Introduzione

Il progetto consiste in una web application che consente all'utente di interrogare un indice ottenuto tramite l'elaborazione della raccolta di file HTML costruita durante il primo homework. Il processo di indicizzazione è realizzato estraendo campi chiave come titolo, autore, contenuto e abstract.

La web application è sviluppata in Java. Si utilizzano Jsoup per il parsing e l'estrazione di contenuti HTML, e Lucene per l'indicizzazione e la ricerca dei contenuti estratti. Il progetto è gestito tramite Maven e distribuito come file WAR su un server Apache Tomcat.

Tecnologie Utilizzate

- Linguaggio di programmazione: Java 21
- Strumento di build: Maven 3
- Librerie e Framework: Jsoup 1.14.3 & Lucene 8.11
- Web Server: Tomcat 8.0.32

> 💡 Il progetto può essere replicato utilizzando versioni diverse da quelle specificate.

Architettura

Componenti Principali:

- FieldExtractorUtil: classe di utilità progettata per estrarre vari campi dai documenti HTML utilizzando la libreria Jsoup. Questa classe è fondamentale per il processo di parsing dei documenti HTML nel progetto, poiché permette di ottenere informazioni strutturate come titolo, autori, contenuto e abstract. Di seguito sono descritti i metodi principali della classe.
  - extractAbstract:
    - Descrizione: estrare l'abstract dal documento HTML.
    - Implementazione: utilizza Jsoup per analizzare il documento e selezionare l'elemento <p> con classe `ltx_p` il cui genitore è un con classe `div.ltx_abstract`.
  - extractContent:
    - Descrizione: estrae l'intero body della pagina HTML.
    - Implementazione: utilizza Jsoup per analizzare il documento HTML e selezionare l'elemento <body>. Il contenuto testuale viene estratto dall'elemento body, rimuovendo eventuali ritorni a capo e spazi superflui.
  - extractAuthor:
    - Descrizione: estrae l'autore o gli autori dal documento HTML.
    - Implementazione: utilizza Jsoup per analizzare il documento HTML e selezionare tutti gli elementi con la classe `"span.ltx_personname"`. Il contenuto HTML di ciascun elemento viene ripulito eliminando i tag indesiderati e sostituendo i tag `<br>` con spazi. I nomi degli autori vengono separati utilizzando vari delimitatori, come virgole, punti e virgola, e spazi multipli. Ogni nome estratto viene poi ulteriormente normalizzato, rimuovendo caratteri non validi e decodificando eventuali entità HTML.
  - extractTitle:
    - Descrizione: estrae il titolo dal documento HTML.

- Implementazione: utilizza Jsoup per analizzare il documento HTML e selezionare l'elemento <h1> con la classe `"h1.ltx_title.ltx_title_document"`. Se l'elemento è presente, restituisce il testo del titolo; altrimenti, restituisce una stringa vuota.

  ```
  Document jsoupDoc = Jsoup.parse(content);
  Elements elements = jsoupDoc.select("h1.ltx_title.ltx_title_document"));
  ```

- Indexer: classe responsabile dell'indicizzazione dei documenti HTML. Utilizza Lucene per creare un indice dei contenuti estratti. Configura un IndexWriter con un PerFieldAnalyzerWrapper per analizzare i campi specifici. Ogni campo (title, authors, content, abstract) viene associato a un analizzatore appropriato.
  - Scelta degli Analyzer:
    - Custom Analyzer composito per i campi title e authors : `WhitespaceTokenizer` `LowerCaseFilter` `WordDelimiterGraphFilter` `ASCIIFoldingFilter`
      - WhitespaceTokenizer: è stato scelto un tokenizzatore basato su spazi bianchi per trattare ogni parola separata da uno spazio come un singolo token. L'approccio preserva l'integrità delle parole composte e non spezza i termini, mantenendo così la struttura originale del testo.
      - LowerCaseFilterFactory: converte tutti i termini in minuscolo, rendendo la ricerca insensibile alle maiuscole.
      - ASCIIFoldingFilterFactory: converte i caratteri accentati o speciali nei loro equivalenti ASCII, rendendo i titoli più facilmente ricercabili anche in presenza di caratteri non standard.

      Per i campi title e authors è stato deciso di evitare l'utilizzo di analyzer più aggressivi per prevenire il rischio di compromettere la precisione delle query. Ad esempio, un uso eccessivo di stopwords o una separazione eccessiva delle parole composte potrebbe generare risultati di ricerca troppo generici e meno rilevanti in quanto i titoli e i nomi di autori contengono principalmente parole chiave cruciali.

- Standard Analyzer per il campo content:
  `StandardAnalyzer(newStopwords().getStopWords())`
  - Il campo content contiene un testo che può includere vari stili di scrittura e formattazioni (es. equazioni, caratteri speciali). Secondo tale ottica, un analyzer generico come lo StandardAnalizer risulta una scelta valida. Inoltre, al fine di migliorare l'indicizzazione e la successiva fase di ricerca, l'analyzer è stato integrato con un custom StopwordFIlter , la cui blacklist contiene termini comuni che appartengono all'ambito scientificio/informatico (es. "study", "models", "methods").

- English Analyzer per il campo Abstract:

  `EnglishAnalyzer(newStopwords().getStopWords())`
  - Per il campo "abstract" è stato scelto un analyzer più mirato e preciso, sfruttando la natura più strutturata e compatta del testo contenuto nel paragrafo p dell'abstract. Anche qui è stato introdotto uno StopwoedFilter che consentisse l'eliminazione di termini comuni di stampo scientifico.

  > 💡 Sarebbe stato possibile utilizzare lo StandardAnalyzer come per il campo content.

- Searcher: esegue ricerche all'interno dell'indice Lucene in base alla query fornita dall'utente tramite l'apposito form, e in relazione a un campo specificato (title, authors, content abstract o title & authors).
  - Costruzione della query: la query è costruita come una **BooleanQuery**, che combina più sotto-query per garantire flessibilità nel matching. Nella **BooleanQuery** vengono incluse diverse tipologie di query:

- **PhraseQuery**: consente di cercare una sequenza di termini nel campo, con una certa tolleranza (slop) per l'ordine dei termini.
- **FuzzyQuery**: permette di fare corrispondenze approssimative tra i termini della query e quelli indicizzati, consentendo di gestire errori di battitura o variazioni nella scrittura.
- **WildcardQuery**: consente di usare caratteri jolly (ad esempio `*` e `?`) per trovare parole che corrispondono a modelli di ricerca più generali, come parole parziali o con caratteri variabili.
- **CustomQuery**: In base al campo selezionato, viene utilizzato un analyzer specifico (vedi processo di indicizzazione) per costruire la query di ricerca tramite la classe `QueryParser` di Lucene.

Ogni sotto-query viene aggiunta alla **BooleanQuery** utilizzando l'operatore **SHOULD**, il che significa che i documenti possono essere restituiti anche se corrispondono solo a una delle sotto-query, aumentando la flessibilità della ricerca. In questo modo, la query combinata consente di eseguire ricerche più ampie e pertinenti, senza limitarsi a un solo tipo di corrispondenza. Questo approccio consente di bilanciare tra **precisione** e **recall** nella ricerca dei documenti.

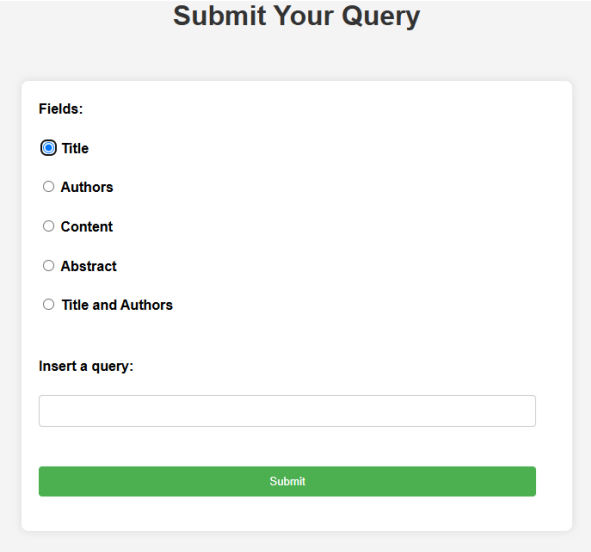
- Esecuzione della ricerca: la ricerca viene eseguita tramite il metodo `search` dell'oggetto `IndexSearcher`, che restituisce i migliori risultati in base alla query e ai parametri di configurazione (ad esempio, il numero massimo di risultati recuperati, definito nella proprietà `top.results`=10).
- Elaborazione dei risultati: una volta ottenuti i documenti Lucene, viene creato un elenco di oggetti `DocumentDTO`, ciascuno rappresentante un documento trovato nell'indice, contenente i campi "title", "authors", "content" , "abstract", “score” e “rankingPosition”.
  I documenti vengono quindi restituiti come una lista di oggetti `DocumentDTO` al Controller (Servlet).

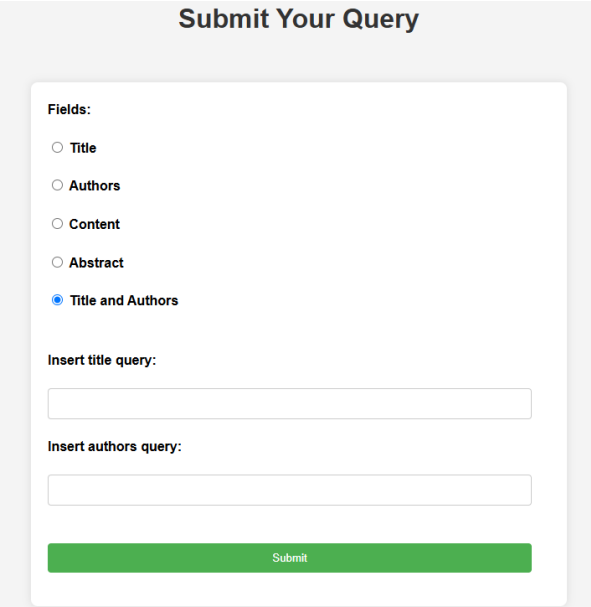
- SearchServlet: servlet Java che gestisce una richiesta POST inviata dall'utente tramite un form HTML per eseguire una ricerca all'interno dell’indice. I documenti `DocumentDTO` restituiti dall’invocazione al metodo search

della classe Searcher vengono inoltrati come attributo tramite richiesta ad un file JSP (`/result.jsp`).

- SearchInitializer: classe listener che, all'avvio dell'applicazione, esegue automaticamente l'indicizzazione dei documenti se la proprietà `indexing.enabled` è impostata a `true`. In caso contrario, non viene eseguita alcuna operazione di indicizzazione.

Interfaccia applicativa





Risultati sperimentali

Tempi di indicizzazione: 24-26 min.

Numero di documenti indicizzati: 9372.

Tempo per il deployment: ~ 34 sec.

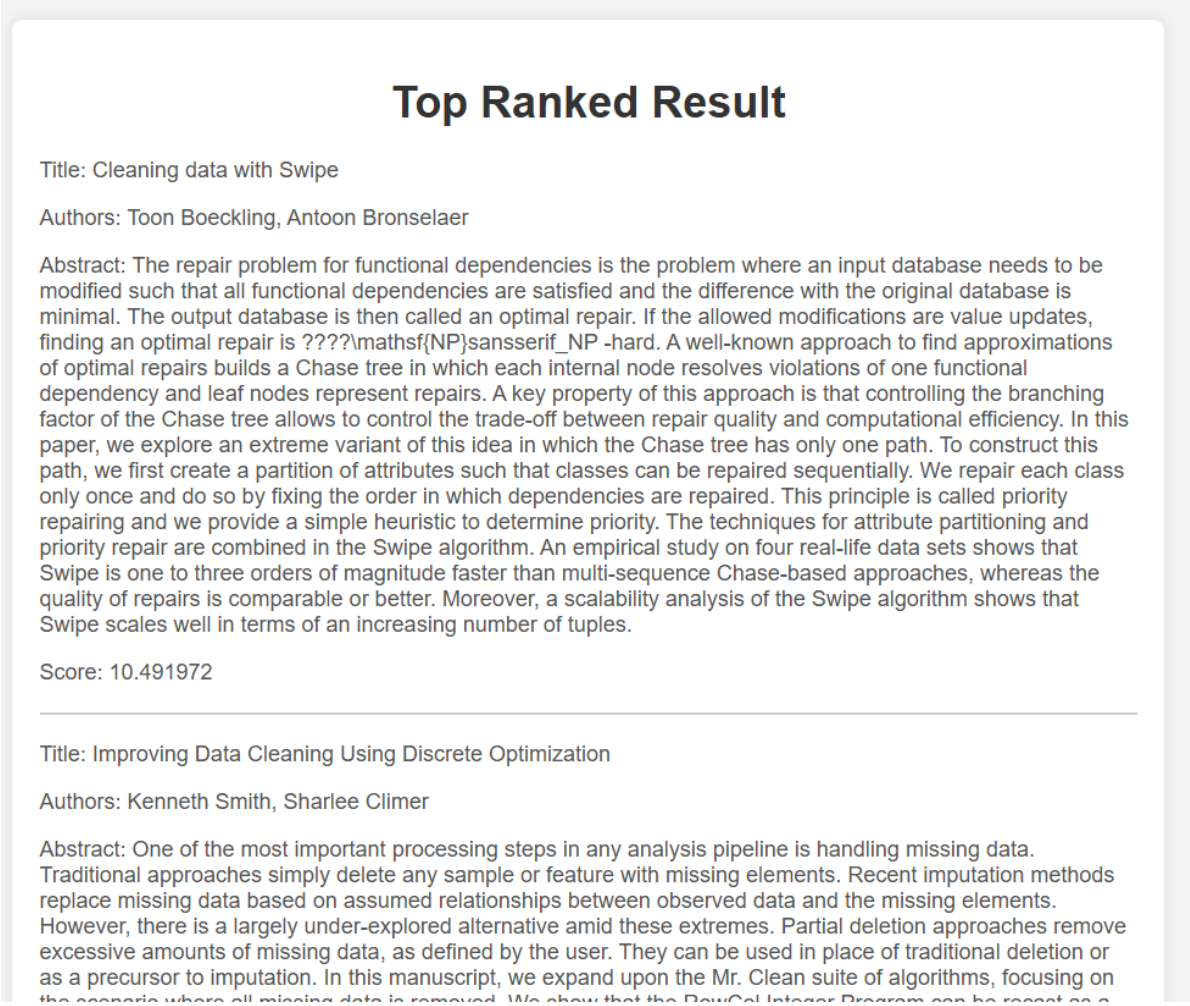
```
Indexed 9372 HTML files
Elapsed time: 26.418326303333334 minutes
[2024-11-09 06:40:42,314] Artifact homework2_ID.war: Artifact is deployed successfully
[2024-11-09 06:40:42,314] Artifact homework2_ID.war: Deploy took 1,587,626 milliseconds
```

```
Connected to server
[2024-11-12 10:12:44,071] Artifact homework2_ID.war: Artifact is being deployed, please wait...
12-Nov-2024 10:12:45.957 INFO [RMI TCP Connection(2)-127.0.0.1] org.apache.jasper.servlet.TldSc
[2024-11-12 10:13:18,396] Artifact homework2_ID.war: Artifact is deployed successfully
[2024-11-12 10:13:18,396] Artifact homework2_ID.war: Deploy took 34,325 milliseconds
```

Query Testing

1. Ricerca per title parziale

   Query: Cleaning data

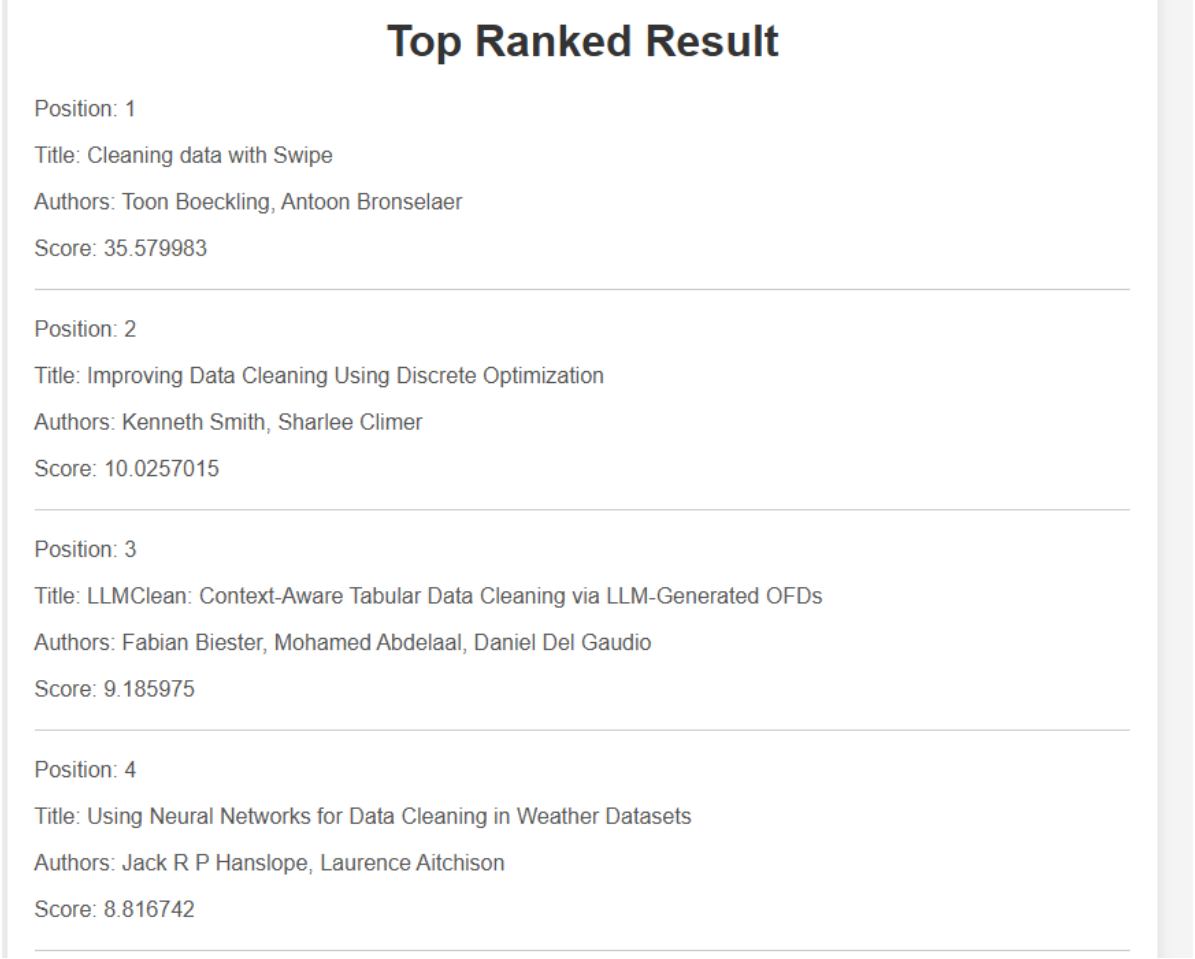


Il titolo del documento recuperato con il punteggio più alto contiene la sottostringa "Cleaning Data", il che è coerente con le aspettative. Lo score è abbastanza elevato : ~10.

2. Ricarca per title completo

   Query: Cleaning data with Swipe

   Documenti restituiti: 10

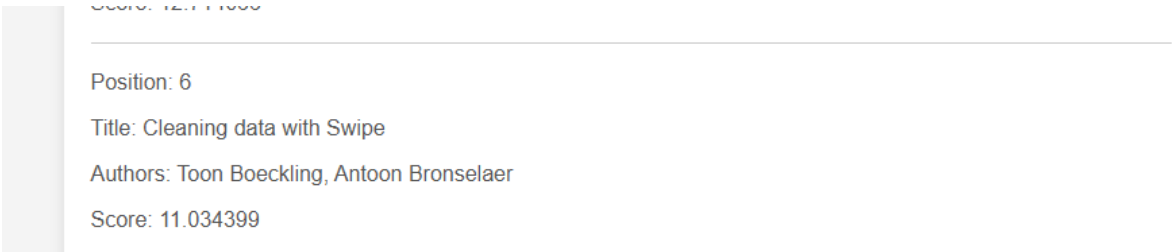


La stringa "Cleaning data with Swipe" corrisponde esattamente al titolo del documento in prima posizione, che ottiene quindi uno score elevato, come previsto. Tuttavia, grazie all'utilizzo della BooleanQuery contenente diverse sotto-query, vengono recuperati anche documenti con titoli che non corrispondono esattamente alla query di input.

3. Ricerca per title con termini in ordine invertito

   Query: Data Cleaning

   Documenti restituiti: 10

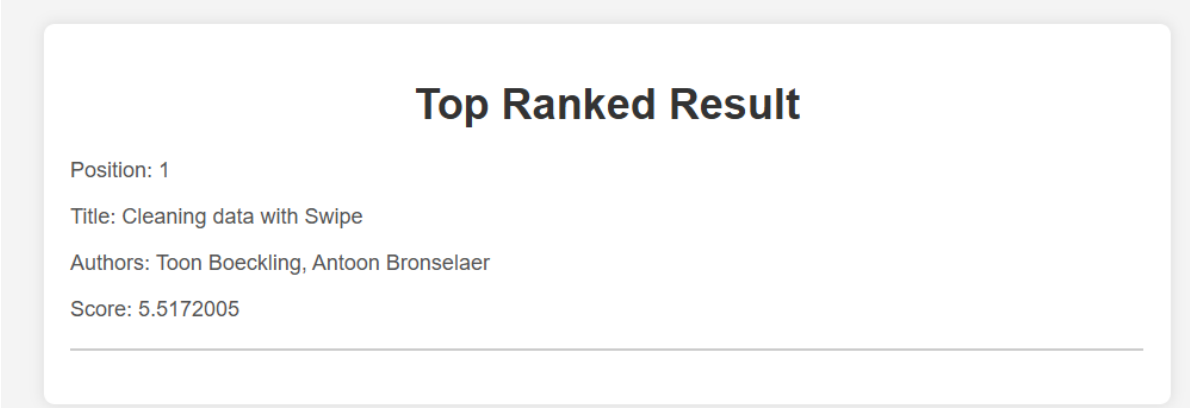


Invertendo l'ordine dei termini nella query da "Cleaning Data" a "Data Cleaning", il documento che in precedenza occupava la prima posizione scende ora al sesto posto.

4. Ricarca per title usando esclusivamente una PhraseQuery (slop = 0)

   Query: Cleaning Data

   Documenti restituiti: 1



Il documento intitolato "Cleaning data with Swipe" è l'unico restituito dal processo di ricerca. Supponendo che l'indicizzazione dei titoli sia corretta per tutti i documenti, si può dedurre che tra quelli indicizzati solo questo documento contenga la stringa "Cleaning data" con i termini nell'ordine specificato. Si ricorda che lo slop impostato a 0 richiede una corrispondenza esatta, senza scarti o termini tra le parole della query.

5. Ricarca per title usando esclusivamente una PhraseQuery (slop = 1)

   Query: Data Cleaning

   Documenti restituiti: 6

Position: 1

Title: Improving Data Cleaning Using Discrete Optimization

Authors: Kenneth Smith, Sharlee Climer

Score: 5.012851

Position: 2

Title: LLMClean: Context-Aware Tabular Data Cleaning via LLM-Generated OFDs

Authors: Fabian Biester, Mohamed Abdelaal, Daniel Del Gaudio

Score: 4.592988

Position: 3

Title: Using Neural Networks for Data Cleaning in Weather Datasets

Authors: Jack R P Hanslope, Laurence Aitchison

Score: 4.408371

Position: 4

Title: OTClean: Data Cleaning for Conditional Independence Violations using Optimal Transport

Authors: Alireza Pirhadi, Mohammad Hossein Moslemi, Alexander Cloninger, Mostafa Milani, Babak Salimi

Score: 4.238022

Position: 5

Title: [Experiments & Analysis] Deep Clustering for Data Cleaning and Integration

Authors: Hafiz Tayyab Rauf, Andre Freitas, Norman W Paton

Score: 4.238022

Position: 6

Title: Cleaning data with Swipe

Authors: Toon Boeckling, Antoon Bronselaer

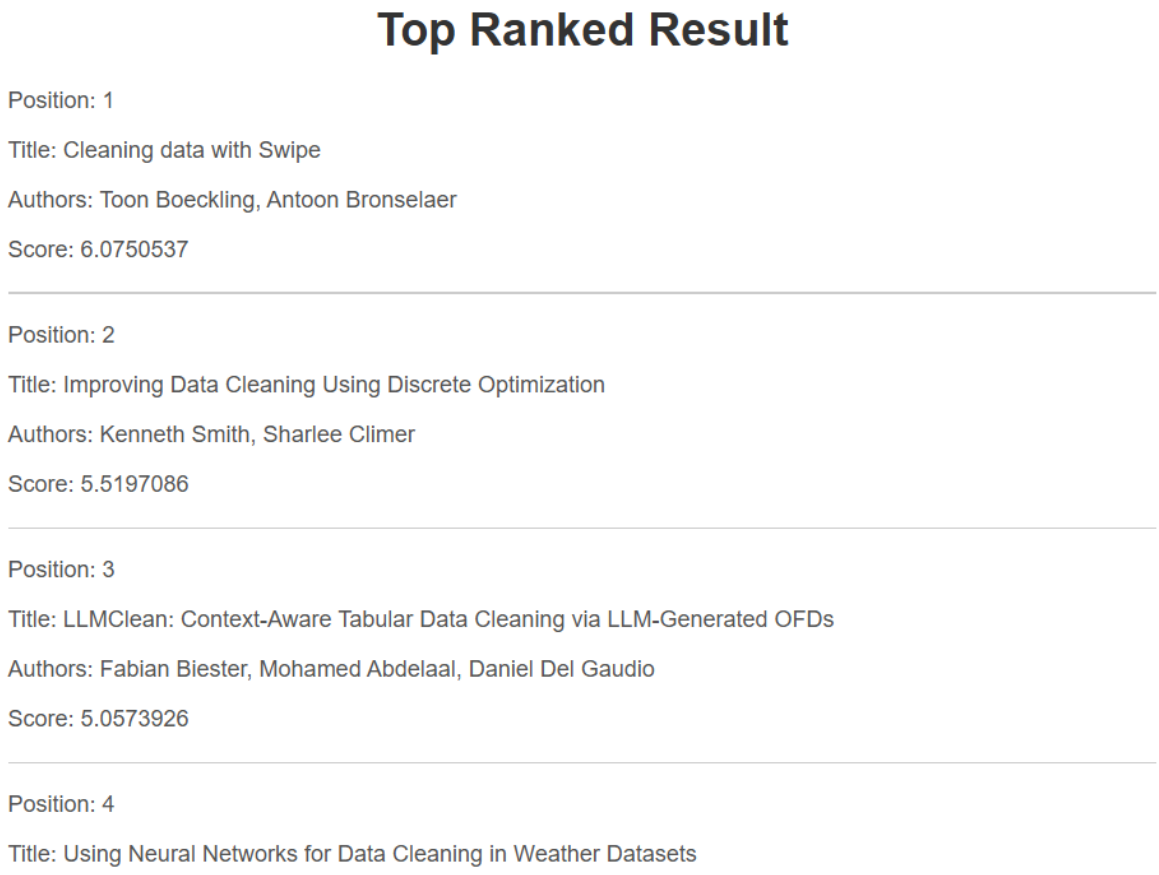
Score: 3.115106

L'impostazione dello slop a 1 consente una maggiore flessibilità nel matching, permettendo di ottenere risultati con una variabilità, seppur limitata, nell'ordine dei termini. Come si può evincere dalla figura allegata, i documenti che contengono la sottostringa "Data Cleaning" ottengono punteggi più elevati. Tuttavia, la ricerca recupera anche il documento intitolato "Cleaning data with Swipe", in cui i termini "Data" e "Cleaning" appaiono nell'ordine inverso rispetto a quello richiesto dalla query, ma che vengono comunque considerati validi grazie alla flessibilità offerta dallo slop.

6. Ricerca per title con errori di battitura

   Query: Claning Data

   Documenti restituiti: 10

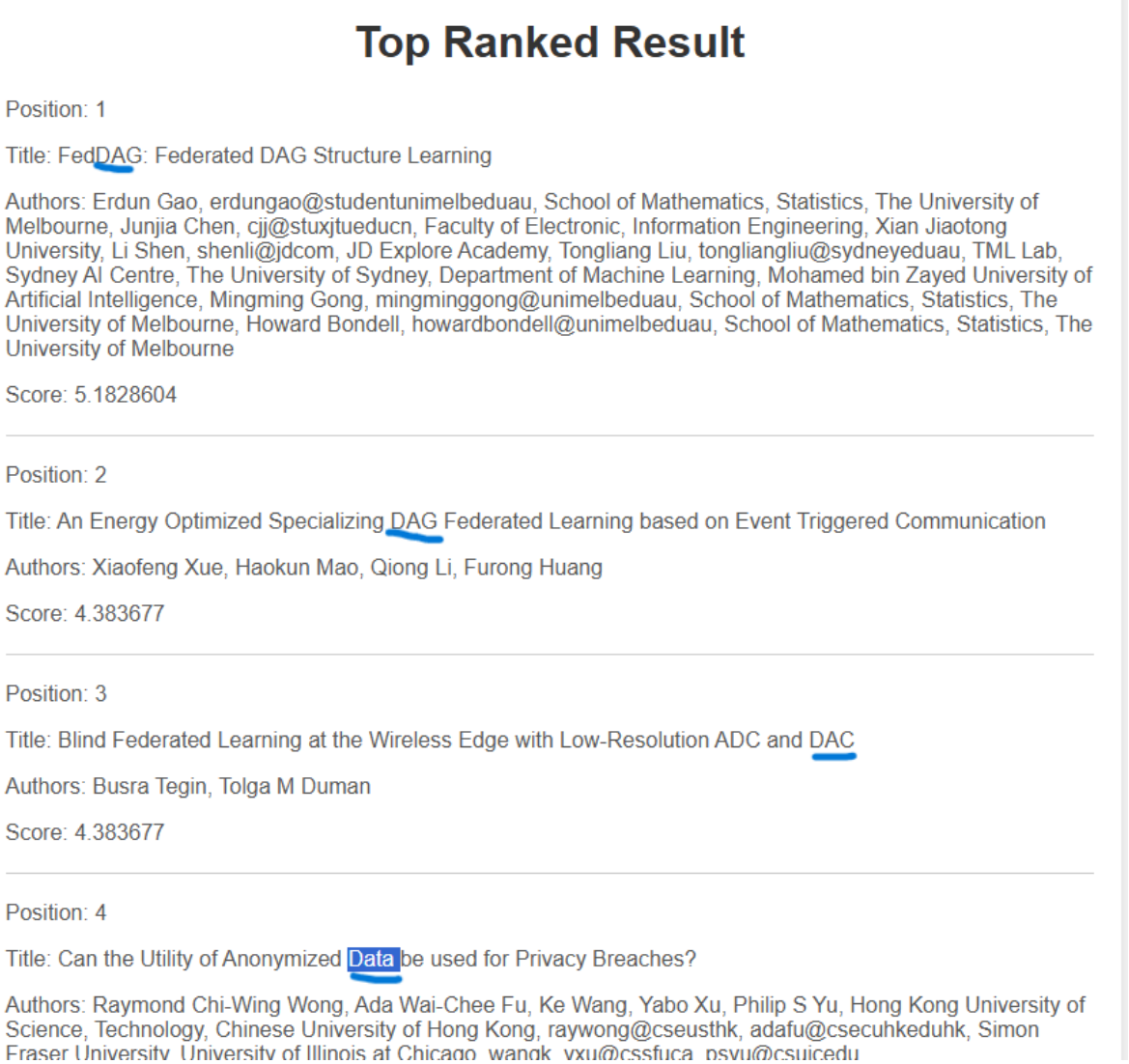


Nonostante la presenza di un errore di battitura nel termine "Cleaning", il processo di ricerca ha restituito risultati che corrispondono alla query "pensata dall'utente". Questo è stato possibile grazie all'inclusione della FuzzyQuery tra le sottoquery della BooleanQuery, che ha consentito di tollerare piccole imprecisioni nei termini ricercati.

7. Ricerca per title con carattere jolly (*)

   Query: Da*

   Documenti restituiti: 10



Top Ranked Result

Position: 1

Title: FedDAG: Federated DAG Structure Learning

Authors: Erdun Gao, erdungao@studentunimelbeduau, School of Mathematics, Statistics, The University of Melbourne, Junjia Chen, cjj@stuxjtueducn, Faculty of Electronic, Information Engineering, Xian Jiaotong University, Li Shen, shenli@jdcom, JD Explore Academy, Tongliang Liu, tongliangliu@sydneyeduau, TML Lab, Sydney AI Centre, The University of Sydney, Department of Machine Learning, Mohamed bin Zayed University of Artificial Intelligence, Mingming Gong, mingminggong@unimelbeduau, School of Mathematics, Statistics, The University of Melbourne, Howard Bondell, howardbondell@unimelbeduau, School of Mathematics, Statistics, The University of Melbourne

Score: 5.1828604

Position: 2

Title: An Energy Optimized Specializing DAG Federated Learning based on Event Triggered Communication

Authors: Xiaofeng Xue, Haokun Mao, Qiong Li, Furong Huang

Score: 4.383677

Position: 3

Title: Blind Federated Learning at the Wireless Edge with Low-Resolution ADC and DAC

Authors: Busra Tegin, Tolga M Duman

Score: 4.383677

Position: 4

Title: Can the Utility of Anonymized Data be used for Privacy Breaches?

Authors: Raymond Chi-Wing Wong, Ada Wai-Chee Fu, Ke Wang, Yabo Xu, Philip S Yu, Hong Kong University of Science, Technology, Chinese University of Hong Kong, raywong@cseusthk, adafu@csecuhkeduhk, Simon Fraser University, University of Illinois at Chicago, wangk, yxu@cssfuca, psyu@csuicedu

il carattere wildcard `*` viene utilizzato in una query per sostituire uno o più caratteri in una parola.
L'asterisco (
`*`) dopo la stringa "Da" permette di trovare parole che iniziano con tale sequenza di termini .

8. Ricerca per authors con carattere jolly (?)

   Query: Lu?a

   Documenti restituiti: 10

Position: 1

Title: Privacy Preserving Record Linkage via \ttlitgrams Projections

Authors: Luca Bonomi, Li Xiong, Rui Chen, Benjamin C M Fung

Score: 1.0

Position: 2

Title: Robust Group Linkage

Authors: Pei Li, Xin Luna Dong, Songtao Guo, footnotemark, Andrea Maurino, Divesh Srivastava

Score: 1.0

Position: 3

Title: Generating Synthetic Data for Real World Detection of DoS Attacks in the IoT

Authors: Luca Arnaboldi, Charles Morisset

Score: 1.0

Position: 4

Title: Accelerating Wireless Federated Learning via Nesterov?s Momentum and Distributed Principle Component Analysis

Authors: Yanjie Dong, nbsp, Luya Wang, Yuanfang Chi, Jia Wang, nbsp, Haijun Zhang, nbsp, Fei Richard Yu, nbsp, Victor C M Leung, nbsp, Xiping Hu, nbsp

Score: 1.0

Position: 5

Title: Bent & Broken Bicycles: Leveraging synthetic data for damaged object re-identification

Authors: Luca Piano, Filippo Gabriele Pratticò, Alessandro Sebastian Russo, Lorenzo Lanari, Lia Morra, Fabrizio Lamberti, Department of Control, Computer Engineering, Politecnico di Torino, Torino, Italy, lucapiano, filippogabrieleprattico, alessandrosebastianrusso@politoit, lorenzolanari@studentipolitoit, liamorra, fabriziolamberti@politoit

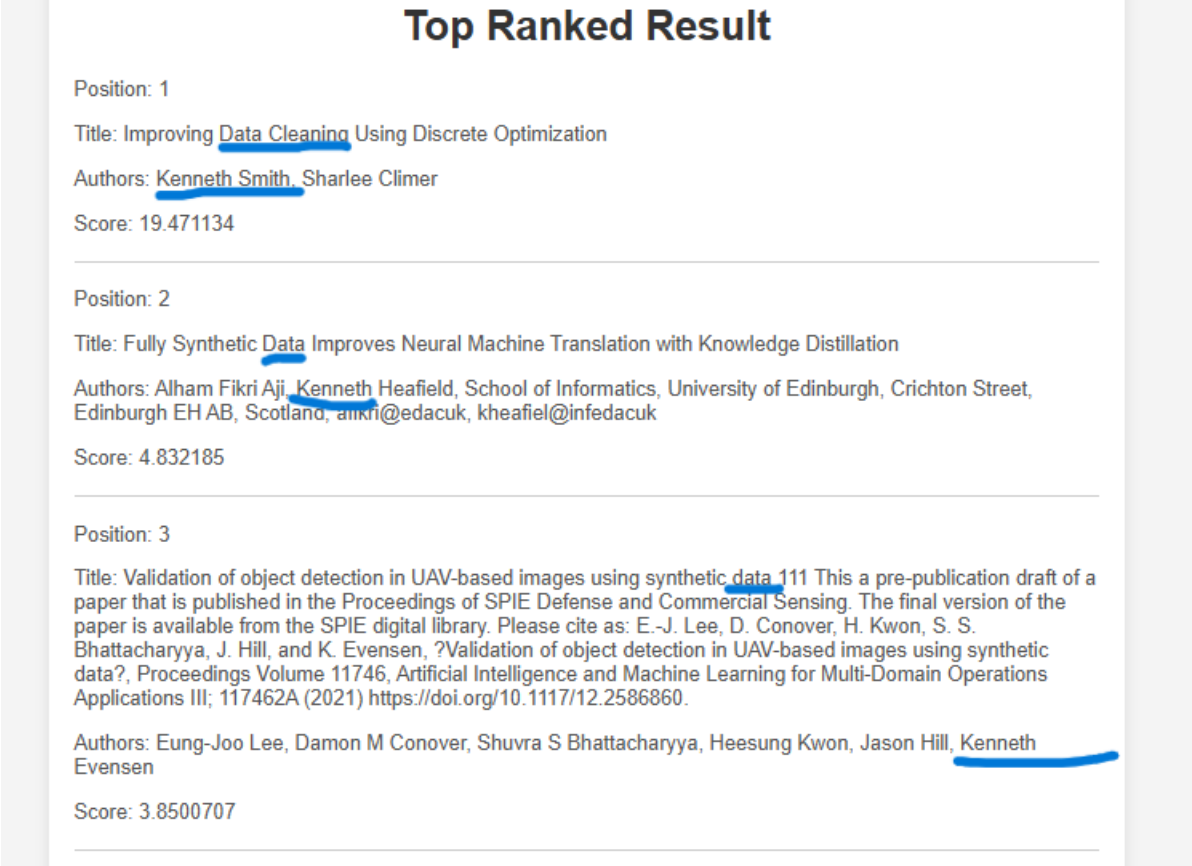
La stringa "Lu?a" rappresenta una query che sfrutta il carattere wildcard `?`, il quale in Lucene viene utilizzato per sostituire un singolo carattere in una parola. La ricerca restituisce tutti i documenti che contengono almeno un autore il cui nome inizi con "Lu" e termini con "a", con un carattere qualsiasi al posto del simbolo `?`. Tutti i documenti che soddisfano questa condizione hanno ricevuto lo stesso punteggio di rilevanza.

9. Ricerca per title e authors(`BooleanClause.Occur.` *`MUST`*)

   Query1 title: Data Cleaning

   Query2 authors: Kenneth Smith

Documenti restituiti: 3



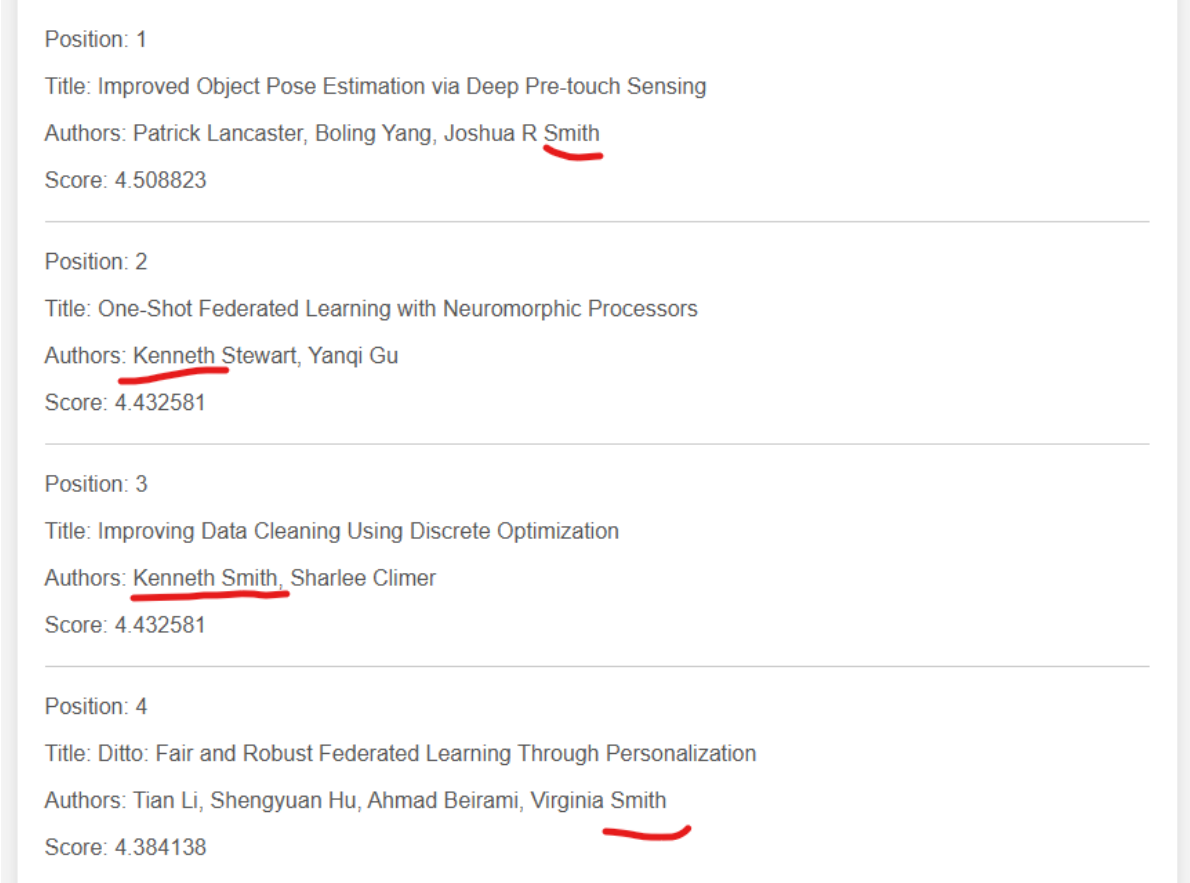
La BooleanQuery relativa al campo title viene combinata con la BooleanQuery associata al campo authors. Per entrambi è impostato l'operatore MUST. Questo implica che, affinché un documento venga incluso nei risultati, deve soddisfare le condizioni di ricerca imposte da entrambe le query.

10. Ricerca per titolo e autore (title: `BooleanClause.Occur.SHOULD`, authors: `BooleanClause.Occur.` *`MUST`*)

    Query1 title: Haowen

    Query2 authors: Kenneth Smith

    Documenti restituiti: 10



La ricerca è configurata per restituire documenti che contengano almeno uno dei termini "Kenneth" o "Smith" nel campo authors. Sebbene non siano stati trovati titoli che includano la stringa "Haowen", ciò non ha penalizzato il punteggio né impedito la recuperabilità di documenti che soddisfano la condizione di ricerca sugli authors.

Conclusioni

I test effettuati su questo "piccolo" motore di ricerca forniscono risultati soddisfacenti e in linea con le aspettative. Con più tempo a disposizione, sarebbe possibile approfondire ulteriormente l'analisi per ottimizzare l'indicizzazione e il processo di ricerca, applicando vincoli più stringenti su alcuni campi e meno su altri, al fine di migliorare la pertinenza dei risultati.