Graboid RFC

Progetto Gestione dell'Informazione

2025-06-12

Menozzi Matteo, Turci Gabriele e Turci Sologni Enrico

Indice

1. Dataset

1.1. Scelta del Dataset

Abbiamo dedicato particolare attenzione alla selezione di un dataset adeguato. La scelta è ricaduta sull'insieme di documenti denominati **Request For Comments** (**RFC**), una serie di pubblicazioni tecniche che definiscono standard, metodologie e sviluppi tecnici di Internet.

La selezione di questo dataset si è basata su una serie di requisiti fondamentali e vantaggi distintivi che gli RFC soddisfano pienamente. Di seguito sono riportati i motivi principali e i benefici derivanti dall'uso degli RFC come dataset per il nostro progetto.

1. **Facilità di Recupero**: Gli RFC sono facilmente accessibili grazie alla struttura del loro sistema di identificazione. Ogni documento è identificato da un numero univoco incrementale, il che rende lo **scraping particolarmente semplice**. Per esempio, è possibile recuperare i documenti incrementando sistematicamente un indice numerico. Inoltre, esistono repository ben organizzati che facilitano il download di massa.

2. **Dimensione del Dataset**: Il corpus degli RFC comprende circa 9600 documenti, un numero perfettamente adeguato per il nostro scopo. Questo range consente di analizzare e testare algoritmi di indicizzazione e ricerca su un dataset realistico senza incorrere in problemi di scalabilità o carenza di dati.

3. **Varietà nella Lunghezza e Complessità dei Documenti**: Gli RFC presentano una significativa varietà nella lunghezza dei documenti:

   - Alcuni documenti contano poche decine di righe.
   - Altri arrivano a diverse centinaia di righe.

   Questa varietà consente di testare il motore di ricerca su casi molto diversi in termini di granularità del contenuto.

4. **Ricchezza di Termini Specifici**: Gli RFC sono documenti tecnici caratterizzati dall'uso di terminologia altamente specifica, inclusi acronimi e parole chiave che rappresentano concetti chiave nel dominio informatico. Questa caratteristica li rende ideali per:

   - Creare indici dettagliati.
   - Formulare query tecniche complesse.
   - Effettuare benchmarking di algoritmi di ricerca avanzati.

5. **Presenza di Metadati Strutturati**: Ogni RFC include metadati ben definiti, quali: *Numero identificativo*, *Titolo*, *Autori*, *Data di pubblicazione*, *Status (es. Standard, Informativo, Obsoleto)*, *Abstract e Parole chiave*, *Sezioni strutturate (Introduzione, Specifiche, Conclusione, ecc.)*.

   La presenza di questi metadati consente di implementare funzionalità avanzate, come la ricerca basata su campi specifici o il filtraggio per criteri.

6. **Compatibilità con Query Complesse**: Gli RFC coprono una vasta gamma di argomenti tecnici e standard, il che li rende perfetti per simulare query complesse e specifiche. Ad esempio, è possibile eseguire ricerche che riguardano concetti tecnici, interazioni tra standard o termini legati a specifiche versioni.

1.2. Scraping del Dataset

Abbiamo deciso di costruire un dataset personalizzato effettuando lo scraping manuale dei documenti e dei relativi metadati. Non abbiamo utilizzato dataset precostruti, poiché volevamo garantire che i dati soddisfacessero specificamente i requisiti del nostro motore di ricerca.

1.2.1. Scraping dei Metadati

La prima fase del processo prevede il recupero dei metadati associati a ciascun documento RFC.

- Il codice HTML degli RFC è carente e i metadati inclusi nei documenti stessi risultano spesso inconsistenti e difficili da estrarre in modo affidabile.
- Per ovviare a questa limitazione, abbiamo individuato una fonte alternativa: i metadati sono disponibili in formato tabellare sul sito ufficiale dell'RFC Editor, raggiungibile al seguente link: Fonte dei Metadati.
- I metadati recuperati includono campi come *numero identificativo*, *titolo*, *autori*, *data di pubblicazione*, *stato*, *estratto*, e *parole chiave*. Questi sono stati scaricati e organizzati in modo strutturato per essere utilizzati successivamente.

1.2.2. Scraping dei Corpi dei Documenti

Una volta ottenuti i metadati, il passo successivo è stato lo scraping dei corpi dei documenti.

- Gli RFC sono accessibili tramite URL che seguono una struttura numerica incrementale, come: `https://www.rfc-editor.org/rfc/rfcX.txt`, dove `X` è l'identificatore numerico del documento (es. 1, 2, 3, ...).
- Uno script iterativo automatizza il download di ogni documento a partire dal numero 1 fino all'ultimo RFC disponibile. Per velocizzare il processo, viene impiegato un sistema basato su threadpool, che consente il download parallelo di più documenti, migliorando le prestazioni.

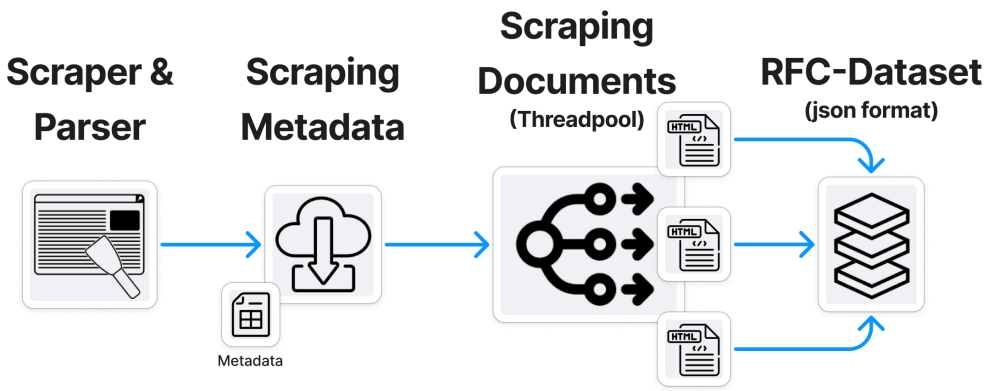


Figura 1: Il diagramma illustra il flusso completo del processo di scraping e costruzione del dataset.

1.2.3. Formattazione del Dataset

Una volta completato il recupero dei metadati e dei corpi dei documenti. Tutte le informazioni sono state consolidate in un unico file in formato **JSON**, dove ogni documento rappresenta un record strutturato contenente sia i metadati sia il testo completo del documento.

Esempio:


```
[
    {
        "Number": "9009",
        "Date": "2021-04",
        "Status": "Proposed Standard",
        "More Info": "Does not obsolete any RFC.",
        "Title": "Efficient Route Invalidation",
        "Authors": [
            "R.A. Jadhav", "P. Thubert", "R.N. Sahoo", "Z. Cao"
        ],
        "Files": [
            "HTML", "TEXT", "PDF", "XML"
        ],
        "Keywords": [
            "NPDAO", "DCO", "no-path", "route", "cleanup"
        ],
        "Abstract": "This document explains ... optimized route invalidation
messaging.",
        "Content": "A directed graph having the property ... timer value of 1
second."
    },
    ...
]
```


1.3. Locazione dei File Interessati

Script di Parsing:

Lo script responsabile per lo scraping dei metadati e dei corpi dei documenti si trova al seguente percorso:

`".../gestione-info/graboidrfc/core/modules/engines/myParser/"`

- Il file di interesse è: **`myParser.py`**

Dataset Generato:

Il dataset prodotto dallo script, contenente i metadati e i corpi dei documenti RFC, è situato nella directory:

`".../gestione-info/graboidrfc/core/data/dataset/"`

- Il file interessati sono: **`dataset.json`**, **`example.json`**

2. Tipologia di Utente e Query Language

2.1. Definizione della Tipologia di Utente

Il nostro motore di ricerca è progettato per soddisfare le esigenze di utenti con poca esperienza o conoscenza del dominio tecnico degli RFC, nello specifico:

- **Studenti universitari**: Studenti che potrebbero necessitare di informazioni specifiche sugli standard e sulle tecnologie di Internet per i loro studi o progetti.
- **Non esperti del settore**: Utenti che non hanno familiarità con gli RFC o con il loro contenuto tecnico ma che hanno necessità di cercare informazioni specifiche.
- **Utenti generici**: Chiunque voglia ottenere informazioni dettagliate presenti negli RFC ma non sappia come localizzarle o in quali documenti siano contenute.

L'obiettivo principale è rendere l'esperienza di ricerca intuitiva e accessibile anche per chi non ha conoscenze approfondite del dominio tecnico.

2.2. Scelta del Query Language

Per soddisfare le esigenze di questa tipologia di utenti, abbiamo optato per un approccio di ricerca basato su:

1. **Keyboard-Based Search**

   Abbiamo scelto di adottare un approccio basato su una ricerca libera (**keyboard-based search**), che consente agli utenti di inserire termini di ricerca nella barra principale in modo diretto e intuitivo. Questo metodo è ideale per chi non ha familiarità con il linguaggio tecnico o con la struttura specifica degli RFC.

2. **Keyboard-Based Search su Campi Specifici**

   Abbiamo inoltre integrato la possibilità di effettuare **ricerche su campi specifici**, come titoli, parole chiave e estratti, per consentire una maggiore precisione nella localizzazione delle informazioni.

2.2.1. Funzionalità di Supporto alla Ricerca

A supporto della ricerca libera, sono state implementate funzionalità avanzate per migliorare ulteriormente l'esperienza utente. Tra queste:

1. **Filtri di Stato**: La possibilità di filtrare i risultati in base allo stato degli RFC, come ad esempio Standard Track, Best Current Practice, Informational, Experimental o Historic.
2. **Filtri Temporali**: Gli utenti possono inoltre limitare i risultati a un determinato periodo temporale, specificando un anno preciso o un intervallo di date, rendendo più facile trovare documenti aggiornati o rilevanti per contesti storici.

La scelta di queste funzionalità e approcci è stata motivata dalla volontà di creare un sistema accessibile, che riduca le barriere tecniche per gli utenti inesperti, senza però sacrificare le capacità avanzate per chi ha esigenze più specifiche.

3. Benchmark

Per valutare le prestazioni del nostro motore di ricerca, abbiamo implementato un sistema di benchmarking basato sui risultati di tre noti motori di ricerca accessibili tramite web: *Google*, *DuckDuckGo* e *Bing*. Questo metodo ci consente di costruire un benchmark costituito da documenti RFC rilevanti per determinate query, associando a ciascun documento un punteggio di rilevanza.

3.1. Obiettivi del Benchmark

L'obiettivo principale è confrontare i risultati dei nostri motori di ricerca con un benchmark che rappresenta uno standard di riferimento. Il benchmark viene costruito raccogliendo i primi risultati restituiti dai tre motori di ricerca per specifiche query e calcolando un punteggio di rilevanza per ciascun documento basato su parametri specifici.

3.2. Costruzione del Benchmark

3.2.1. Fase 1: Recupero dei Risultati

Abbiamo sviluppato uno script, denominato **`autoUrlExtractor.py`**, che utilizza l'API *SerpApi* per effettuare scraping delle pagine di ricerca di *Google*, *DuckDuckGo* e *Bing*. Lo script esegue i seguenti passaggi:

1. Esegue una query predefinita su ciascun motore di ricerca, limitando i risultati al dominio degli RFC (**`site:https://www.rfc-editor.org/rfc/`**).
2. Recupera i primi risultati in formato URL, come ad esempio **`https://www.rfc-editor.org/rfc/rfcNUMBER.html`**, dove **`NUMBER`** rappresenta il numero identificativo dell'RFC.
3. Salva i risultati in un file JSON strutturato, associando ogni query ai relativi URL estratti dai motori di ricerca.

3.2.2. Fase 2: Calcolo della Rilevanza

Il secondo script, denominato **`createBenchMark.py`**, legge il file JSON generato nella fase precedente e calcola la Rilevanza di ciascun documento RFC. Questo processo si basa su due parametri principali: la **Frequenza del Documento** e il **Punteggio basato sulla Posizione** nei risultati.

3.2.2.1. Parametri Iniziali

Per ogni documento RFC, calcoliamo i seguenti parametri:

1. **Frequenza del Documento**: La frequenza indica in quanti motori di ricerca il documento compare tra i risultati. È definita come:

$$\text{Frequenza}_{\text{doc}} = \#\ \text{Numero di motori di ricerca in cui doc compare}$$

2. **Punteggio in base alla Posizione**: Per ciascun motore di ricerca, calcoliamo il punteggio di un documento in base alla sua posizione nei risultati. Il punteggio diminuisce con il crescere della posizione, secondo la seguente formula di decremento logaritmico:

$$\text{PunteggioSingolaPosizione}_{\text{doc,engine}} = \frac{1}{\log_2\left(\text{Posizione}_{\text{engine}}(\text{doc}) + 1\right)}$$

   Qui, la posizione $\mathbf{Posizione_{engine}(doc)}$ rappresenta l'indice del documento nei risultati di un motore di ricerca. Il punteggio totale del documento basato sulle posizioni viene calcolato sommando i singoli punteggi di tutti i motori di ricerca in cui il documento compare:

$$\text{PunteggioTotale}_{\text{doc}} = \sum_{i=1}^{E} \text{PunteggioSingolaPosizione}_{\text{doc},i}$$

3.2.2.2. Punteggio Finale

Successivamente calcoliamo per ciascun documento la Rilevanza combinando il Punteggio Totale e la Frequenza ottenute in precedenza, utilizzando la formula:

$$\text{Rilevanza}_{\text{doc}} = \text{PunteggioTotale}_{\text{rfc}} * (1 + \alpha * \text{Frequenza}_{\text{rfc}})$$

Dove α è un fattore di peso configurabile tra **`0`** e **`1`**, che determina l'importanza della frequenza.

3.2.2.3. Normalizzazione

Le Rilevanze vengono normalizzate tra 1 e 3 per rendere i risultati comparabili e più interpretabili:

$$\text{RilevanzaNormalizzata}_{\text{doc}} = \left\lfloor 2 \cdot \frac{\text{Rilevanza}_{\text{doc}} - \text{Min}}{\text{Max} - \text{Min}} \right\rfloor + 1$$

Dove **Min** e **Max** sono rispettivamente la minima e la massima rilevanza calcolata per i documenti.

3.2.3. Risultato del Benchmark

Alla fine del processo, otteniamo un benchmark strutturato che associa a ogni query una lista di documenti RFC con i relativi punteggi di rilevanza. Questo benchmark fornisce un riferimento oggettivo per confrontare le prestazioni dei nostri motori di ricerca (*Whoosh*, *PyLucene*, *PostgreSQL*) rispetto ai motori di ricerca di riferimento (*Google*, *Duckduckgo*, *Bing*).

- Di seguito un esempio di **output strutturato** dello script:

```
Testo della Query: 'Prompt query 2 site:rfc-editor.org'
Rfc: 9308, Rilevanza: 4.89279, Normalizzata: 3.00000, Arrotondata: 3
Rfc: 9001, Rilevanza: 4.00000, Normalizzata: 2.58616, Arrotondata: 3
⋮
Rfc: 9287, Rilevanza: 2.79203, Normalizzata: 2.02622, Arrotondata: 2
⋮
Rfc: 9297, Rilevanza: 0.60206, Normalizzata: 1.01109, Arrotondata: 1
Rfc: 9443, Rilevanza: 0.57813, Normalizzata: 1.00000, Arrotondata: 1
```

3.2.4. Revisione Manuale dei Risultati

Come fase finale del processo di benchmarking, i risultati generati automaticamente dallo script verranno sottoposti a un controllo manuale. Questa revisione ha l'obiettivo di garantire che i punteggi di rilevanza assegnati ai documenti siano coerenti e rispecchino accuratamente la loro importanza rispetto alle query formulate.

3.3. Locazione dei File Interessati

Gli script responsabili per lo scraping dei risultati ed il calcolo della rilevanza si trovano al seguente percorso:

`".../gestione-info/graboidrfc/core/modules/engines/myBenchmark/"`

- Gli script: **`benchmark.py`**, **`extractor_online.py`**

`".../gestione-info/graboidrfc/core/data/benchmark/"`

- I file: **`benchmark.json`**, **`extracted_online.json`**, **`extracted_online_revisited.json`**

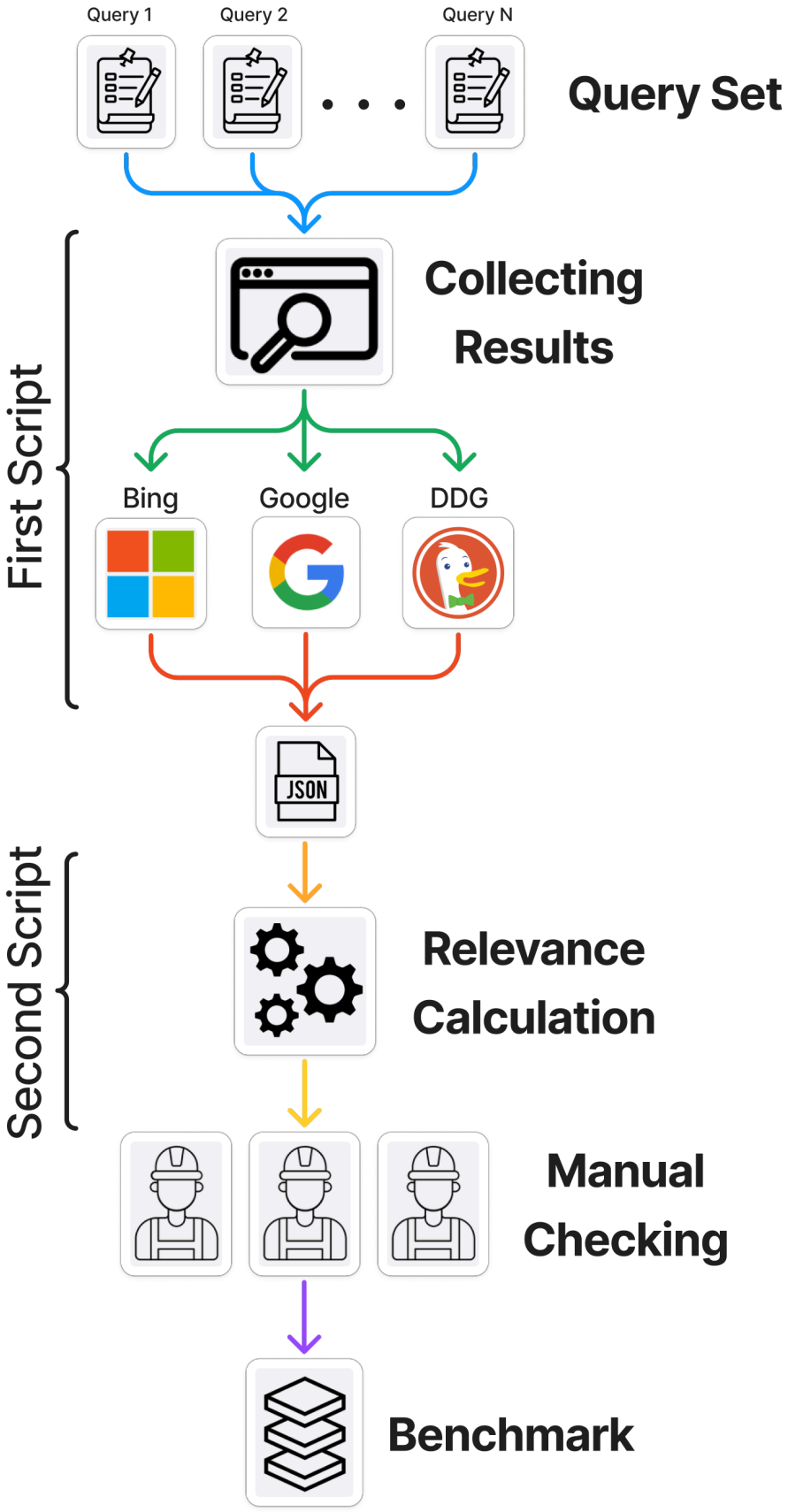


Figura 2: Il diagramma illustra il flusso completo del processo di creazione del benchmark.

4. Queries

Le seguenti query sono state selezionate per mettere alla prova i sistemi da noi sviluppati, permettendo un confronto finale.

4.1. Elenco

1. Prima Query
   - **UIN**: L'utente vuole trovare informazioni dettagliate su TCP Fast Open, un'estensione di TCP che riduce la latenza della connessione iniziale.
   - **MOTIVO**: Questa query testa la capacità del motore di ricerca di recuperare documenti che trattano un'estensione specifica di un protocollo ben noto. La difficoltà sta nel fatto che TCP è molto generico e potrebbe portare a molti risultati irrilevanti.
   - **QUERY**: `"TCP Fast Open"`

2. Seconda Query
   - **UIN**: L'utente vuole trovare informazioni sulle implementazioni dei protocolli di autenticazione EAP e RADIUS in reti Wi-Fi sicure.
   - **MOTIVO**: Questa query stressa il motore perché richiede di trovare documenti che trattano EAP e RADIUS specificamente nel contesto delle reti Wi-Fi, filtrando documenti generici sulla sicurezza di rete.
   - **QUERY**: `"EAP RADIUS Wi-Fi security"`

3. Terza Query
   - **UIN**: L'utente cerca informazioni dettagliate sulla sicurezza di TLS 1.3 e sulle principali differenze rispetto a TLS 1.2.
   - **MOTIVO**: Query specifica che richiede distinzione tra più versioni di un protocollo, valutando la gestione di documenti con termini simili.
   - **QUERY**: `"TLS 1.3 security differences from TLS 1.2"`

4. Quarta Query
   - **UIN**: L'utente vuole un elenco di tutti gli RFC che trattano autenticazione e crittografia nei protocolli di posta elettronica come SMTP, POP3 e IMAP.
   - **MOTIVO**: Questa query stressa il motore di ricerca perché deve collegare concetti di sicurezza (autenticazione e crittografia) con protocolli email specifici. Inoltre, diversi RFC trattano ciascun protocollo separatamente.
   - **QUERY**: `"SMTP POP3 IMAP authentication encryption"`

5. Quinta Query
   - **UIN**: L'utente vuole esplorare la relazione tra sicurezza nei protocolli di routing e il protocollo BGP, cercando RFC che trattino entrambi i temi.
   - **MOTIVO**: Questa query combina due concetti correlati ma trattati in RFC separati. Stressa il motore di ricerca nel riconoscere documenti che parlano di sicurezza nel routing senza escludere BGP.
   - **QUERY**: `"BGP security routing"`

6. Sesta Query
    - **UIN**: L'utente vuole confrontare IPv4 e IPv6 in termini di gestione della frammentazione dei pacchetti.
    - **MOTIVO**: La query sfida il motore nel trovare documenti che menzionano sia IPv4 che IPv6, ma in un contesto specifico come la frammentazione, evitando documenti che parlano di IPv6 in generale.
    - **QUERY**: `"IPv4 IPv6 fragmentation"`

7. Settima Query
    - **UIN**: L'utente vuole capire come i protocolli di controllo della congestione hanno influenzato lo sviluppo di QUIC.
    - **MOTIVO**: La query è difficile perché lega concetti di performance e ottimizzazione della rete con QUIC, richiedendo di trovare documenti che parlano di controllo della congestione e del loro impatto su QUIC.
    - **QUERY**: `"QUIC congestion control impact"`

8. Ottava Query
    - **UIN**: L'utente vuole approfondire il funzionamento del protocollo TCP con particolare attenzione ai meccanismi di controllo della congestione utilizzati per ottimizzare le prestazioni di rete.
    - **MOTIVO**: Query multi-campo che testa la capacità del motore di identificare documenti specifici sul controllo della congestione in TCP. Il motore deve bilanciare la rilevanza tra contenuto, titolo e abstract.
    - **QUERY**:`"TCP protocol and congestion control title:TCP abstract:Congestion Control"`

9. Nona Query
    - **UIN**: L'utente è interessato alla gestione degli indirizzi IPv6 e alla sua evoluzione rispetto alle versioni precedenti del protocollo IP, con particolare riferimento agli RFC.
    - **MOTIVO**: Query multi-campo che valuta la capacità del motore di recuperare documenti focalizzati sia sulla gestione degli indirizzi IPv6 che sulla sua evoluzione nel tempo. Il campo abstract fornisce un contesto più generale sulla rete.
    - **QUERY**:`"IPv6 address management and evolution title:IPv6 abstract:network"`

10. Decima Query
    - **UIN**: L'utente vuole analizzare le diverse versioni del protocollo TLS, le differenze tra loro e il livello di sicurezza offerto da ciascuna.
    - **MOTIVO**: Query multi-campo che verifica la capacità del motore di classificare documenti sulle versioni di TLS e i loro miglioramenti in termini di sicurezza, sfruttando il campo delle keyword per migliorare la pertinenza dei risultati.
    - **QUERY**: `"TLS protocol versions and security keywords:security"`

5. Motori di Ricerca

L'obiettivo è progettare, sviluppare e confrontare diverse varianti di tre motori di ricerca, analizzandone il comportamento in relazione agli UIN utente precedentemente enunciati.

I tre motori di ricerca presi in esame si basano sui seguenti sistemi: *Whoosh*, *PyLucene* e *PostgreSQL*. Questi sistemi consentono l'indicizzazione e la ricerca efficiente dei documenti.

Struttura e differenze dei sistemi

- **Interfaccia comune**: I tre sistemi condividono la stessa interfaccia web, garantendo un'esperienza uniforme e il supporto per tutti i filtri elencati nella sezione Scelta del Query Language.
- **Preprocessing indipendente**: Ogni sistema indicizza i documenti del dataset applicando il proprio meccanismo di preprocessing predefinito, senza una fase di normalizzazione comune tra i tre motori.

Varianti sviluppate

Per ciascuno dei tre sistemi, abbiamo implementato due varianti basate su altrettanti modelli di ranking. Per ciascuno dei modelli di ranking abbiamo progettato due varianti:

- Una **prima variante** basata sull'algoritmo di ranking standard.
- Una **seconda variante** dello stesso modello standard, modificata attraverso personalizzazioni della formula o dei parametri di ranking.

  Ad esempio, nel caso di *Whoosh*, abbiamo scelto il modello di ranking *TF-IDF* (VSM), che calcola la rilevanza di un documento sommando i contributi dei pesi dei singoli termini, determinati tramite la formula *TF-IDF*.

  Successivamente, abbiamo sviluppato una versione modificata moltiplicando la formula per un parametro aggiuntivo, il Freshness Factor, che favorisce i documenti più recenti rispetto a quelli meno recenti. Abbiamo denominato questa variante *TF-IDF-FF*.

Successivamente, analizzeremo e confronteremo il comportamento delle varianti rispetto agli UIN utente, impiegando metriche appropriate e rappresentandone i risultati attraverso grafici intuitivi.

Nei prossimi paragrafi presenteremo e discuteremo le caratteristiche delle varianti sviluppate.

5.1. Motore Whoosh

Per il motore di ricerca *Whoosh*, abbiamo selezionato due modelli di ranking implementati di default nel sistema: *BM25* e *TF-IDF (VSM)*.

Locazione dello Script

Lo script che gestisce l'integrazione con *Whoosh* è situato al seguente percorso:

`".../gestione-info/graboidrfc/core/modules/engines/myWhoosh/myWhoosh.py"`

5.1.1. Modello BM25F

Whoosh implementa la versione BM25F del BM25. La sua formula è:

$$\text{BM25F}(\text{tf}, \text{idf}, \text{fl}, \text{avgfl}, B, \text{K1}) = \text{idf} \cdot \frac{\text{tf} \cdot (K_1 + 1)}{\text{tf} + K_1 \cdot \left(1 - B + B \cdot \frac{\text{fl}}{\text{avgfl}}\right)}$$

dove:

- ***tf***: (*term frequency*) frequenza del termine nel documento;
- ***idf***: (*inverse document frequency*) misura della rarità del termine nella collezione;
- ***fl***: (*field length*) lunghezza del campo (es. numero di termini nel documento);
- ***avgfl***: (*average field length*); lunghezza media del campo nei documenti (es. lunghezza media documenti);
- ***B***: bilanciamento della normalizzazione della lunghezza del documento (tipicamente tra 0 e 1);
- ***K1***: costante di regolazione per la saturazione del tf (tipicamente tra 1.2 e 2.0).

L'*inverse document frequency* è definita dalla seguente formula:

$$\text{idf}(\text{dc}, n) = \log\left(\frac{\text{dc}}{n + 1}\right) + 1$$

dove:

- ***dc***: (*document count*); numero totale di documenti nella collezione;
- ***n***: numero di documenti che contengono il termine interessato.

L'implementazione di queste formule è disponibile nel modulo Scoring di Whoosh, in particolare nelle classi «WeightingModel», «BM25F», «BM25FScorer» e nella funzione «bm25».

Per questo modello abbiamo lasciato i valori di default impostati da Whoosh.

- $k_1 = 1,2$
- $b = 0,75$

5.1.2. Modello BM25F Custom

Il modello di ranking BM25F Custom è stato configurato con i seguenti parametri:

- $k_1 = 1,5$
- $b = 0,5$

5.1.3. Modello VSM

Whoosh implementa una versione standard del modello **VSM**, calcolando *TF-IDF* come il prodotto tra la *term frequency* (frequenza del termine nel documento) e l'*inverse document frequency* (misura della rarità del termine nella collezione).

Quando si seleziona il modello TF-IDF in Whoosh, il motore utilizza la formula convenzionale della **cosin-similarity**, tipica del **Vector Space Model**, per calcolare la similarità tra query e documenti:

$$\text{cosine-similarity(q,d)} = \frac{q \cdot p}{\|q\| \cdot \|p\|}$$

L'*inverse document frequency* è definita dalla seguente formula:

$$\text{idf}(\text{dc}, n) = \log\left(\frac{\text{dc}}{n+1}\right) + 1$$

dove:

- ***dc***: (*document count*) numero totale di documenti nella collezione;
- ***n***: numero di documenti che contengono il termine considerato.

La formula per il calcolo dello score TF-IDF di un termine t è:

$$\text{TF-IDF}(t, \text{dc}, n) = \text{tf}(t) \times \text{idf}(\text{dc}, n)$$

dove:

- ***tf***: è la *term frequency* del termine t nel documento, ottenuta nel codice tramite: «`matcher.weight()`».

L'implementazione di queste formule è disponibile nel modulo Scoring di Whoosh, in particolare nelle classi «WeightingModel», «TF_IDF» e «TF_IDFScorer».

Questa è l'implementazione predefinita del modello TF-IDF (VSM) in Whoosh, che nel nostro caso è stata utilizzata senza modifiche.

5.1.4. Modello VSM Custom (TF-IDF-FF)

Questo modello rappresenta una variante del classico modello VSM fornito da Whoosh. L'estensione consiste nell'introduzione di un parametro aggiuntivo durante il calcolo del punteggio assegnato a un termine: il **freshness factor**, un coefficiente che tiene conto della *recentezza* del documento. Più un documento è recente, maggiore sarà il suo freshness factor; al contrario, documenti più datati avranno un contributo ridotto al punteggio finale.

Il calcoli dell'*inverce document frequency* e della *cosine-similarity* rimangono invariati rispetto al modello TF-IDF (VSM) standard.

Per determinare la freschezza di un documento, abbiamo scelto come unità di misura la differenza in **mesi** tra la data corrente e la data di pubblicazione del documento. Tale differenza viene calcolata con la formula:

$$\text{months\_diff} = (\text{current.year} - \text{publication.year}) * 12 + (\text{current.month} - \text{publication.month})$$

A partire da questo valore, il **freshness factor** viene definito come:

$$\text{ff}(\lambda_{\text{freshness}}, \text{md}) = \exp(-\lambda_{\text{freshness}} \cdot \text{months\_diff})$$

dove:

- $\boldsymbol{\lambda_{\text{freshness}}}$: è un parametro che controlla quanto velocemente la freschezza decresce nel tempo. Nella nostra implementazione è fissato a $0,1$.

Integrando il freshness factor nel calcolo TF-IDF, la formula finale del punteggio diventa:

$$\text{TF-IDF-FF}(t, \text{dc}, n, \lambda_{\text{f}}, \text{md}) = \text{tf}(t) \times \text{idf}(\text{dc}, n) \times \text{ff}(\lambda_{\text{f}}, \text{md})$$

dove:

- ***tf(t)***: term frequency del termine ***t***;
- ***idf(dc, n)***: $\log\left(\frac{\text{dc}}{n+1}\right) + 1$;
- ***ff***: funzione esponenziale decrescente in base alla differenza in mesi ***md*** e al parametro $\boldsymbol{\lambda_{\text{f}}}$;

Il modulo customizzato che implementa il modello TF-IDF-FF (VSM) è situato al seguente percorso:

`".../gestione-info/graboidrfc/core/modules/engines/myWhoosh/custom_scorer.py"`

5.2. Motore PyLucene

Per il motore di ricerca *PyLucene*, abbiamo selezionato due modelli di ranking implementati di default nel sistema: *BM25* e *VSM*.

Locazione dello Script

Lo script che gestisce l'integrazione con *PyLucene* è situato al seguente percorso:

`".../gestione-info/graboidrfc/core/modules/engines/myPylucene/myPylucene.py"`

5.2.1. Modello BM25

PyLucene implementa la versione classica del modello BM25, utilizzando una riscrittura equivalente della formula standard. Dalla documentazione ufficiale e dai commenti nel codice emerge la seguente espressione:

$$\text{BM25}(t, d, \text{dl}, \text{avgdl}, B, K_1) = \text{idf}(t) \cdot \frac{\text{tf}(t,d) \cdot (K_1 + 1)}{\text{tf}(t,d) + K_1 \cdot \left(1 - B + B \cdot \frac{\text{dl}}{\text{avgdl}}\right)}$$

dove:

- ***tf (t)***: (*term frequency*) frequenza del termine nel documento;
- ***idf (d)***: (*inverse document frequency*) misura della rarità del termine nella collezione;
- ***dl***: (*document length*) lunghezza del documento (es. numero di termini nel documento);
- ***avgdl***: (*average document length*); lunghezza media dei documenti;
- ***B***: bilanciamento della normalizzazione della lunghezza del documento (tipicamente tra 0 e 1);
- ***K_1***: costante di regolazione per la saturazione del tf (tipicamente tra 1.2 e 2.0).

Nel codice sorgente di Lucene, questa formula è riformulata come:

$$\text{BM25}(t, d) = \text{weight} - \frac{\text{weight}}{1 + \text{tf}(t,d) + K_1 \cdot \text{norminverse}})$$

dove:

- $\mathbf{weight = idf * boost}$, dove boost è un fattore moltiplicativo per aumentare la rilevanza di termini specifici;
- $\mathbf{normInverse} = \frac{1}{\boldsymbol{K_1} \cdot \left(\mathbf{1} - \boldsymbol{B} + \boldsymbol{B} \cdot \frac{\text{dl}}{\text{avgdl}}\right)}$, cioè l'inverso della normalizzazione per la lunghezza del documento.

L'*inverse document frequency* è definita dalla seguente formula:

$$\text{idf}(N, n) = \log\left(1 + \frac{N - n + 0.5}{n + 0.5}\right)$$

dove:

- ***N***: numero totale di documenti nella collezione;
- ***n***: numero di documenti che contengono il termine interessato.

Le implementazioni di queste formule sono disponibili nel sorgente del modulo BM25Similarity di Lucene, in particolare nelle classi «BM25Similarity», «BM25Scorer» e «BM25Scorer::score».

Per questo modello abbiamo lasciato i valori di default impostati da Lucene, descritti nella documentazione e riscontrabili nel codice:

- $k_1 = 1,2$
- $b = 0,75$

5.2.2. Modello BM25 Custom

Il modello di ranking BM25 Custom è stato configurato con i seguenti parametri:

- $k_1 = 1,5$
- $b = 0,5$

5.2.3. Modello VSM

In Lucene, il modello **Vector Space Model** utilizza come metrica predefinita la **TF-IDF** per calcolare i pesi dei termini relativi ai documenti. Successivamente, per calcolare la similarità tra query e documenti, viene impiegata una versione rifinita della similarità coseno, come descritto nella documentazione ufficiale di Lucene.

Concettualmente, la similarità coseno è calcolata tramite la formula:

$$\text{cosine-similarity(q,d)} = \frac{V(q) \cdot V(d)}{|V(q)| \cdot |V(d)|}$$

dove:

- $\boldsymbol{V(q) \cdot V(d)}$ è il prodotto scalare dei vettori pesati della query q e del documento d_i;
- $\boldsymbol{|V(q)| \cdot |V(d)|}$ sono le rispettive norme Euclidee dei vettori.

In realtà, Lucene utilizza una formula equivalente ma rifinita, data da:

$$\text{cosine-similarity(q,d)} = \sum_{t \in q} (\text{tf}(t \in d) \cdot \text{idf}(t)^2 \cdot \text{t.getBoost}() \cdot \text{norm}(t, d))$$

dove:

- $\boldsymbol{\text{tf}(t \in d) = \sqrt{\text{frequenza del termine } t \text{ in } d_i}}$
- $\boldsymbol{\text{idf}(t) = 1 + \log\left(\frac{\text{docCount} + 1}{\text{docFreq} + 1}\right)}$, con *docCount* numero totale di documenti e *docFreq* il numero di documenti contenenti t_i;
- $\boldsymbol{\text{t.getBoost}()}$ è un fattore di potenziamento specifico del termine;
- $\boldsymbol{\text{norm}(t, d)}$ è una normalizzazione legata alla lunghezza del documento.

L'implementazione di queste formule è disponibile nel modulo ClassicSimilarity e nel modulo TFIDFSimilarity di Lucene, in particolare nelle classi «ClassicSimilarity», «TFIDFSimilarity» e «TFIDFScorer».

Questa è l'implementazione predefinita del modello TF-IDF (VSM) in Lucene, e nel nostro caso viene utilizzata senza modifiche.

5.2.4. Modello VSM Custom (TFLN-PIDF)

Questo modello rappresenta una variante del classico modello TF-IDF (VSM) fornito da Lucene. L'estensione consiste nell'applicazione di formule alternative per il calcolo della *term frequency* e dell'*inverse document frequency*. Nello specifico: abbiamo calcolato la TF normalizzando con il logaritmo; e abbiamo adottato la variante probabilistica dell'IDF.

La *Term Frequency Log Normalization* è data da:

$$\mathrm{TFLN}(\mathrm{rawFreq}) = 1 + \log_{10}(\mathrm{rawFreq})$$

La *Probabilistic Inverse Document Frequency* è data da:

$$\mathrm{PIDF}(\mathrm{numDocs},\ \mathrm{docFreq}) = 1 + \log_2\left(\frac{\mathrm{numDocs} - \mathrm{docFreq} + 1}{\mathrm{docFreq} + 1}\right)$$

Entrambe le formule si trovano nella pagina di wikipedia relativa al TF-IDF.

Il modulo customizzato che implementa il modello VSM (TFLN-PIDF) è situato al seguente percorso:

`".../gestione-info/graboidrfc/core/modules/engines/myPylucene/custom_scorer.py"`

5.3. Motore PostgreSQL

Per il motore di ricerca *PostgreSQL*, abbiamo selezionato due modelli di ranking implementati di default nel sistema: *CD* e *SW*.

Locazione dello Script

Lo script che gestisce l'integrazione con *PostgreSQL* è situato al seguente percorso:

`"../gestione-info/graboidrfc/core/modules/engines/myPostgres/myPostgres.py"`

Document Lenght Normalization (DLN)

Poiché un documento più lungo ha maggiori probabilità di contenere un termine della query, è ragionevole **tenere conto della lunghezza del documento**. Le funzioni (*ts_rank*, *ts_rank_cd*) accettano un'opzione di normalizzazione intera che specifica se e come la lunghezza del documento debba influire sul punteggio.

Questa opzione intera è una **bitmask**, cioè un valore in cui ciascun bit rappresenta un comportamento specifico. È possibile combinare più comportamenti usando | (ad esempio, *2|4*). L'elenco completo delle opzioni disponibili si trova nella relativa documentazione.

5.3.1. Modello SW (Statistical Weighting → ts_rank)

Questa funzione calcola il **ranking basato sulla frequenza** dei termini della query nel documento. Più volte un termine appare, maggiore sarà il punteggio. La formula considera:

- Numero di occorrenze;
- Posizione dei termini (più vicini all'inizio → punteggio maggiore);
- Eventuali pesature dei termini A, B, C, D (usate per dare più peso a certi campi).

Consultare la relativa documentazione. Nel nostro caso viene utilizzata senza apportare modifiche.

5.3.2. Modello SW (Statistical Weighting → ts_rank) Custom (DLN)

La combinazione di opzioni utilizzata è **2|8**, in particolare:

- **2**: divide il punteggio per la lunghezza del documento;
- **8**: divide il punteggio per il numero di parole uniche nel documento.

5.3.3. Modello CD (Coverage Density → ts_rank_cd)

Questa variante non si concentra solo sulla frequenza, ma su **quante parti diverse del documento sono «coperte» dalla query** e **quanto sono vicine tra loro le corrispondenze**. In altre parole:

- Premia i documenti che coprono più parti della query;
- Premia documenti in cui i termini della query appaiono vicini;
- Non aumenta il punteggio solo perché un termine si ripete molte volte.

Consultare la relativa documentazione. Nel nostro caso viene utilizzata senza apportare modifiche.

5.3.4. Modello CD (Coverage Density → ts_rank_cd) Custom (DLN)

La combinazione di opzioni utilizzata è **2|8**, in particolare:

- **2**: divide il punteggio per la lunghezza del documento;
- **8**: divide il punteggio per il numero di parole uniche nel documento.

6. Performance

6.1. Metriche

Le metriche descritte in seguito sono implementate all'interno del modulo:

`"../gestione-info/graboidrfc/core/modules/engines/myComparator/myComparator.py"`

6.1.1. Precision e Recall

La precision indica la qualità dei risultati recuperati, cioè quanti documenti sono effettivamente rilevanti tra quelli restituiti dal sistema. La recall, invece, misura la capacità del sistema di recuperare tutti i documenti rilevanti per una determinata query.

$$\text{Precision} = \frac{|\text{Ra}|}{|\text{A}|} \qquad\qquad \text{Recall} = \frac{|\text{Ra}|}{|\text{R}|}$$

Dove:

- ***Ra***: è l'insieme dei documenti rilevanti tra quelli recuperati;
- ***A***: è l'insieme totale dei documenti recuperati;
- ***R***: è l'insieme dei documenti rilevanti per la query.

6.1.2. Average Precision

L'average precision al livello r di recall è ottenuta calcolando la precisione al livello di recall r per ciascuna query e facendo poi la media di queste precisioni su tutte le query:

$$\text{AP}(r) = \frac{1}{N_q} \cdot \sum_{i=1}^{N_q} P_i(r)$$

Dove:

- $\boldsymbol{N_q}$ é il numero totale di query.
- $\boldsymbol{P_i(r)}$ è la precisione interpolata al livello r rispetto alla query q.

L'average precision viene anche calcolata su ogni singola query q in questo modo:

$$\text{AP}(q) = \frac{1}{n} \cdot \sum_{r=0}^{n} P_q(r)$$

Dove:

- $\boldsymbol{n}$ é il numero totale di livelli di recall.
- $\boldsymbol{P_q(r)}$ è la precisione interpolata al livello r per la query q.

Detto ciò, l'average precision è utile perchè misura la capacità del motore di ricerca di restituire documenti rilevanti in cima ai risultati.

6.1.3. Mean Average Precision

La Mean Average Precision (MAP) si ottiene normalizzando la somma delle precisioni medie (delle query) per il numero totale di query:

$$\text{MAP} = \sum_{q \in Q} \frac{\text{AP}(q)}{|Q|}$$

È utile per avere una valutazione complessiva delle prestazioni di un sistema su tutte le query.

6.1.4. F-Measure

L'F-Measure è la media armonica tra precision e recall, ed è utile quando si vuole un compromesso tra le due: se un sistema è molto preciso ma non recupera molti documenti, avrà una precision alta ma un recall basso, e viceversa. L'F-Measure bilancia quindi entrambi i parametri per valure le prestazioni complessive di un sistema.

$$F = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

6.1.5. Normalized Discounted Cumulative Gain

La Normalized Discounted Cumulative Gain (NDCG) tiene conto sia della rilevanza dei documenti che della loro posizione nei risultati. Quindi, più gli elementi rilevanti sono in alto, maggiore è il punteggio.

$$\text{NDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p} \qquad\qquad \text{DCG}_p = \text{rel}_1 + \sum_{j=2}^{p} \frac{\text{rel}_j}{\log_2(j)}$$

Dove:

- $\mathbf{DCG_p}$ ***(Discounted Cumulative Gain)***: misura la qualità del ranking considerando la posizione degli elementi rilevanti.
- $\mathbf{IDCG_p}$ ***(Ideal DCG)***: rappresenta il miglior ranking possibile.

È molto utile per valutare l'efficacia con cui il motore di ricerca ordina i risultati, normalizzando il punteggio per permettere confronti equi tra diversi ranking.

6.2. Grafici

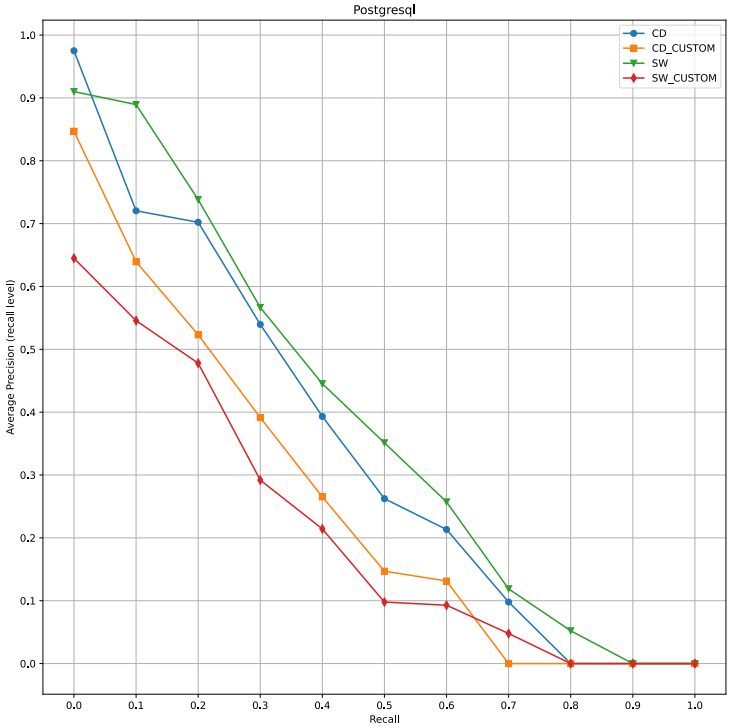
I grafici generati hanno l'obiettivo di fornire un confronto visivo tra le prestazioni dei diversi motori di ricerca e dei vari modelli di ranking utilizzati, sulla base di un set di query di valutazione.

La generazione dei grafici è stata implementata all'interno del modulo:

`"../gestione-info/graboidrfc/core/modules/engines/myComparator/myGraphs.py"`

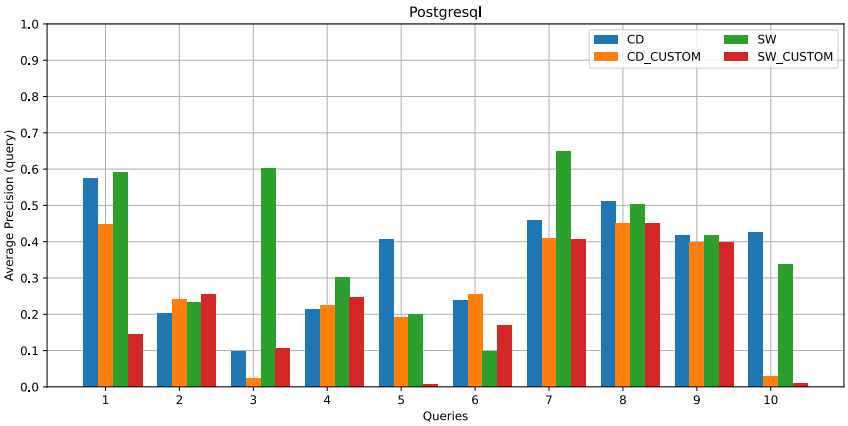
6.2.1. Average Precision per Modello (Average Precision Level)

Per ogni motore di ricerca viene calcolata la media dei valori di precision ottenuti su tutte le query, per ciascun modello di ranking. Questo consente di confrontare le prestazioni medie dei modelli in termini di precision.



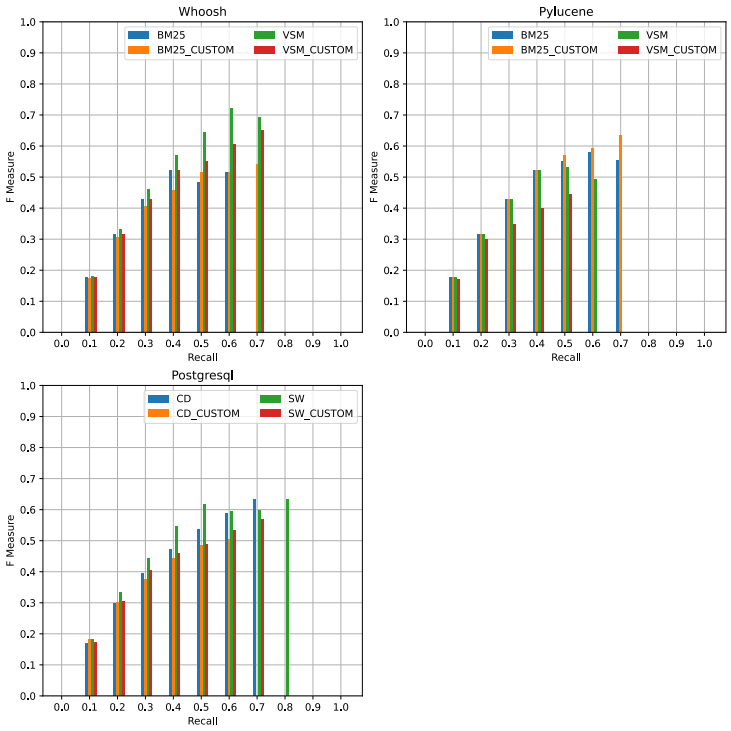
6.2.2. Average Precision per Query (Average Precision Query)

Per ogni singola query si calcola la media dei valori di precision ai diversi livelli di recall. Questo permette di analizzare l'andamento della precision rispetto al recall per ogni query.



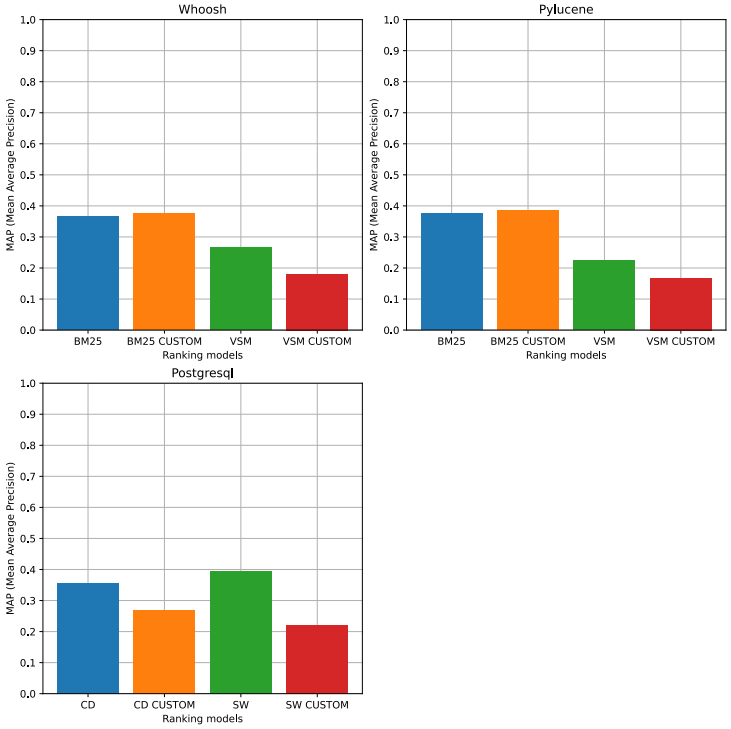
6.2.3. F-Measure

Questa serie di tre grafici consente di confrontare i valori di F-Measure (che bilancia precision e recall) ai vari livelli di recall, per ogni motore di ricerca su una specifica query.



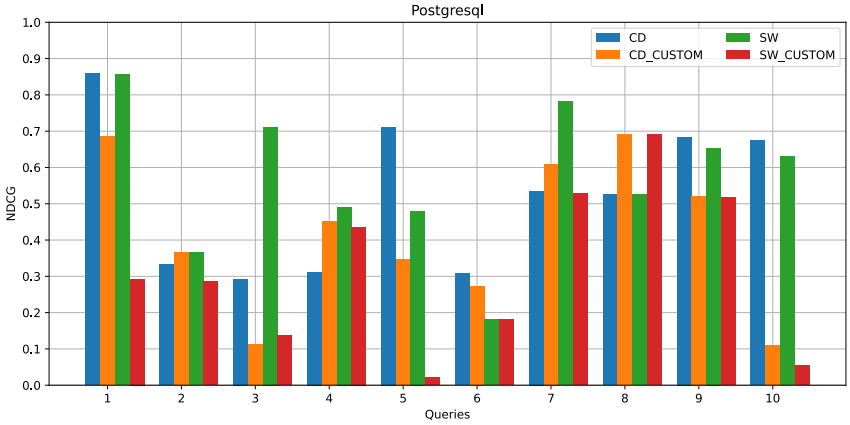
6.2.4. Mean Average Precision (MAP)

Questa composizione di tre grafici mostra la Mean Average Precision per ciascun motore di ricerca, evidenziando la performance media globale su tutte le query.



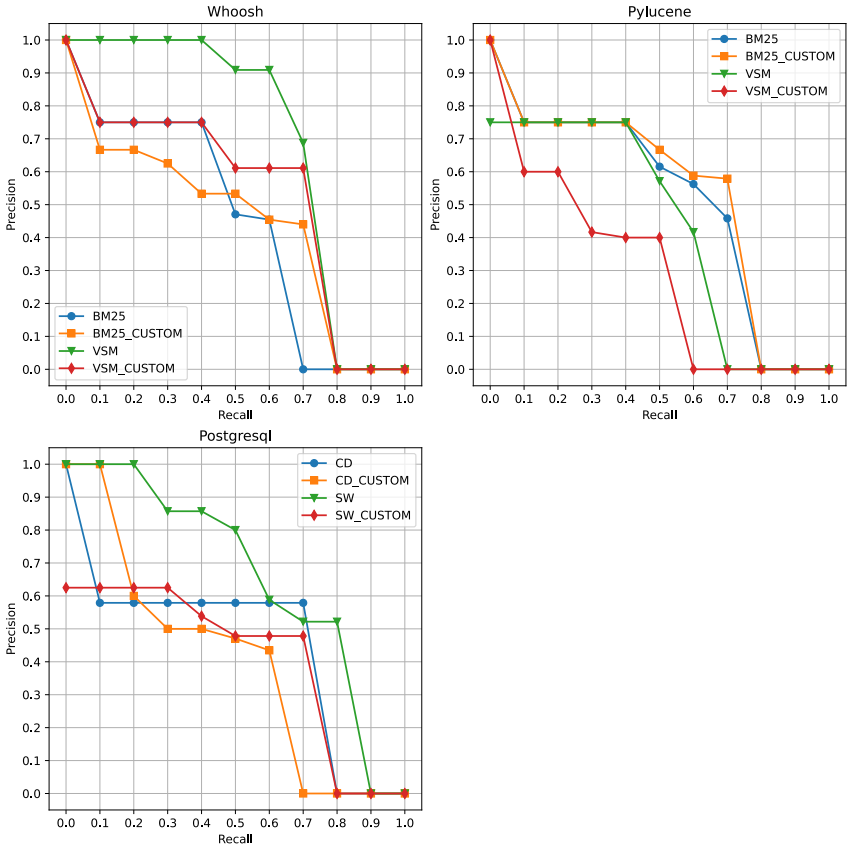
6.2.5. Normalized Discounted Cumulative Gain (nDCG)

Questo grafico rappresenta il guadagno cumulativo scontato normalizzato per ciascuna query, offrendo un'indicazione della qualità dell'ordinamento dei documenti restituiti, tenendo conto della posizione e della rilevanza.



6.2.6. Precision-Recall

La composizione di tre grafici Precision-Recall consente di confrontare, per ogni livello di recall, i valori di precision ottenuti dai vari motori di ricerca su una specifica query.



6.3. Analisi

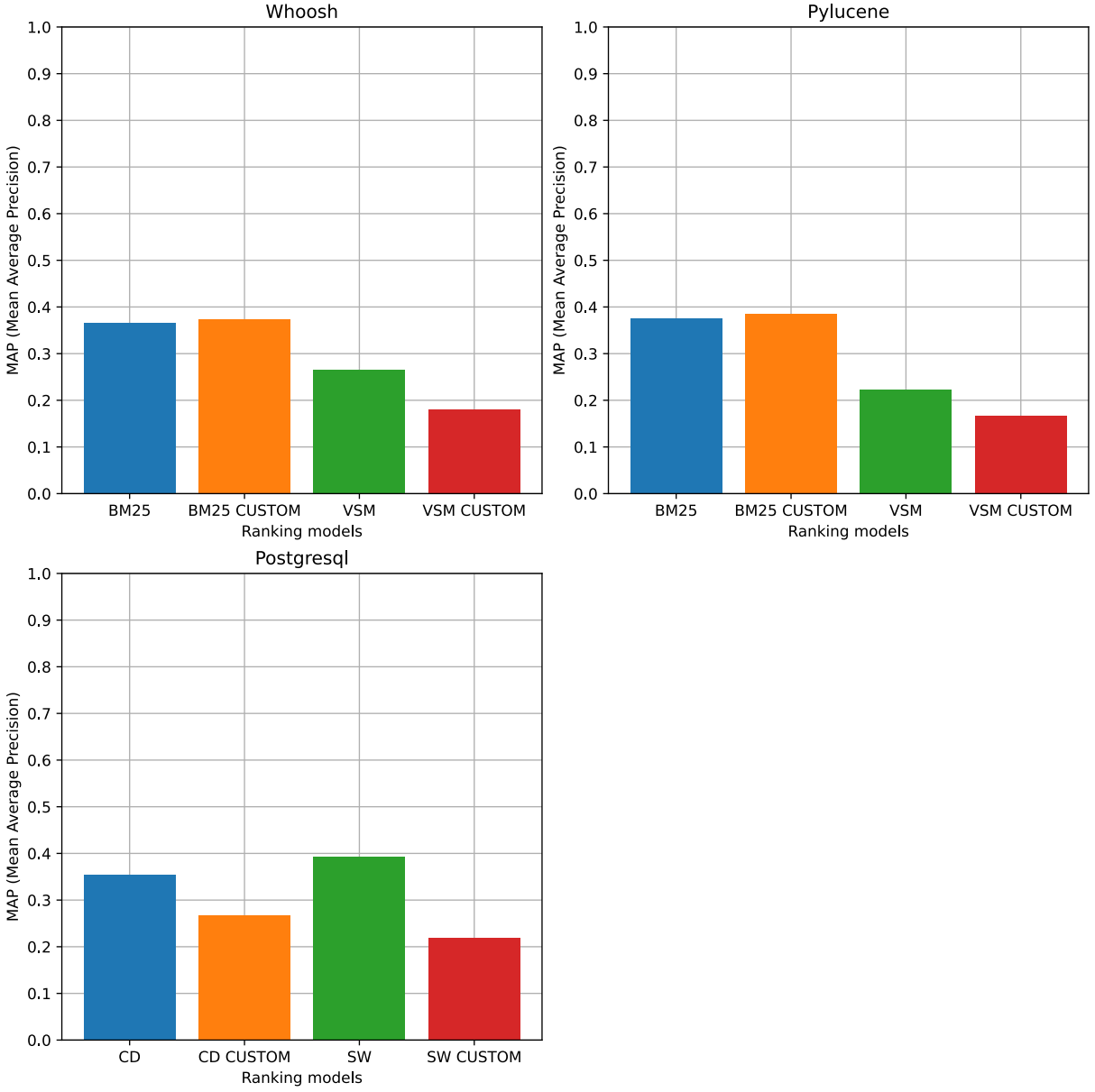
L'analisi ha l'obiettivo di confrontare alcuni casi notevoli delle query e identificare in quali casi le prestazioni dei search engine peggiorano/migliorano.

6.3.1. Comportamento dei search engine

Com'è possibile osservare dal grafico della MAP (Mean Average Precision), Whoosh e PyLucene si comportano in modo simile mentre (sia in questo grafico che negli altri) il modello SW di PostgreSQL risulta avere una precisione media maggiore rispetto a tutti gli altri modelli.

Inoltre le versioni custom del BM25 per Whoosh e PyLucene si comportano in modo molto simile a quello di default. Mentre per PostgreSQL differisce molto anche a causa della personalizzazione dei valori diversa rispetto a quella di Whoosh e PyLucene.

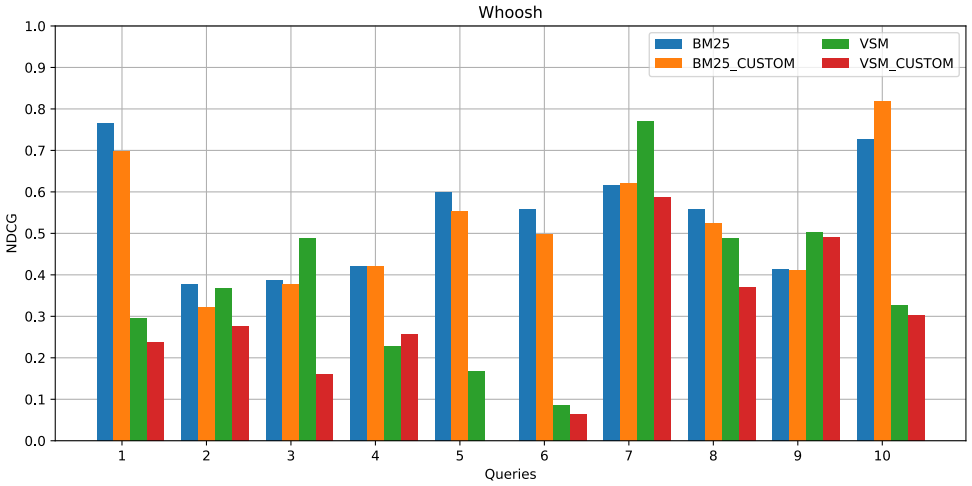
In generale le modifiche apportate ai valori non hanno portato miglioramenti significativi alle prestazioni rispetto a quelli di default.

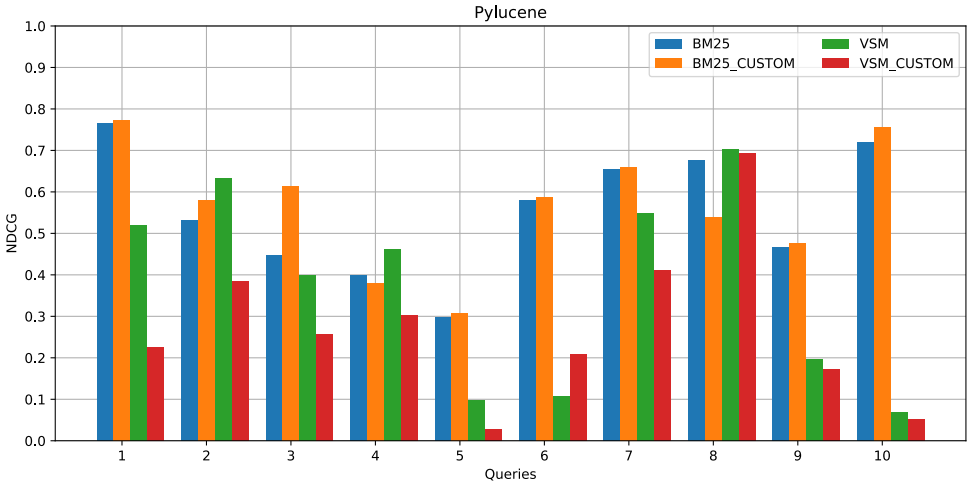


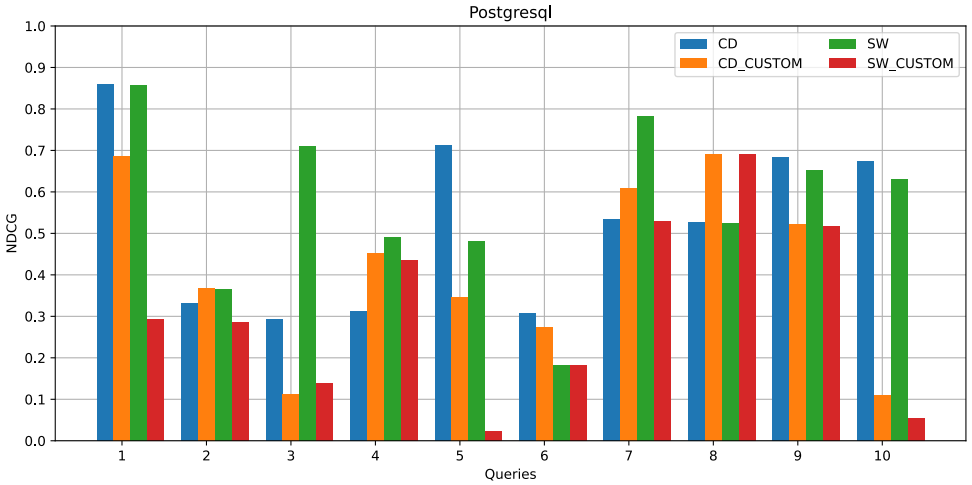
6.3.2. Comportamento delle query

Sfruttando i grafici di *precision recall* è possibile identificare quali sono le query che si comportano in modo migliore.

6.3.2.1. Analisi NDCG





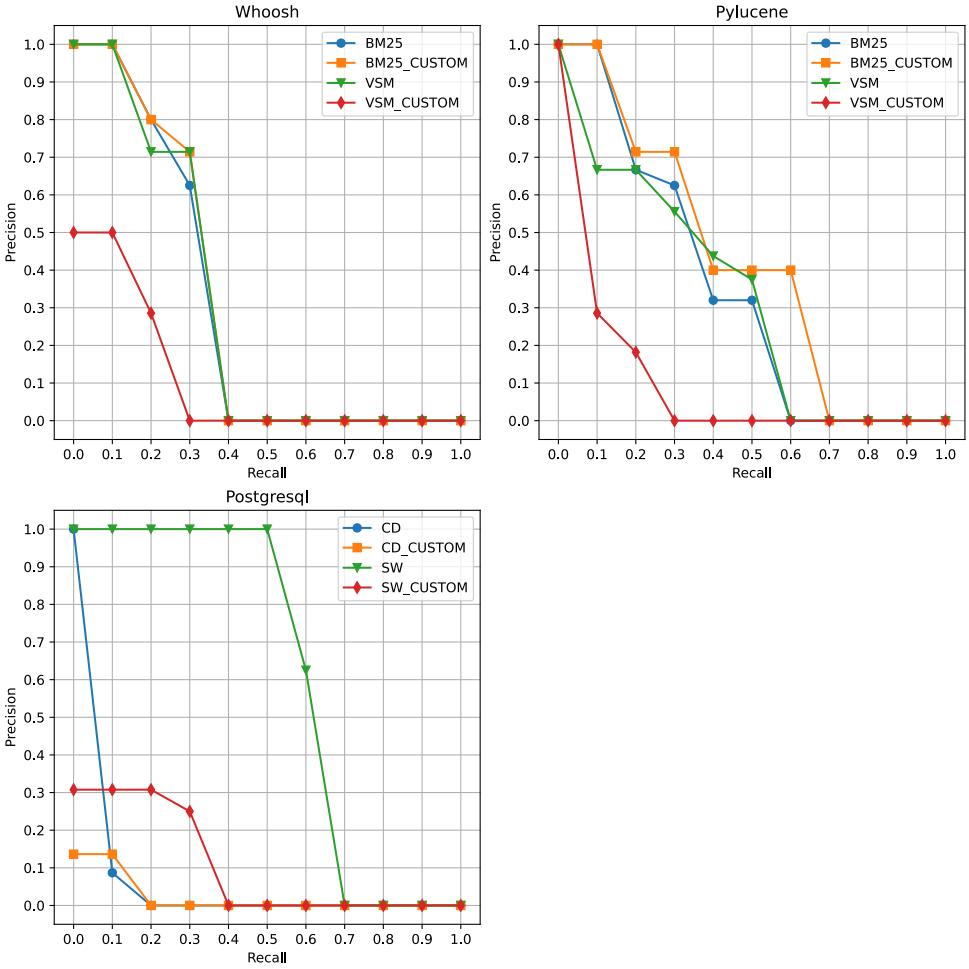


6.3.2.2. Analisi query 3

La query 3 richiede distinzione tra più versioni di un protocollo, valutando la gestione di documenti con termini simili.

Il grafico ci mostra comportamenti molto diversi, il più interessante è PostgreSQL che mantiene una precision molto elevata solamente per il modello SW, mentre gli altri modelli partono da precision basse o hanno un brusco calo.

Questo comportamento può essere dovuto alla presenza di molti documenti che contengono frequentemente i termini della query, anche se non in modo coerente o rilevante. In particolare, il modello CD è influenzato dalla distribuzione dei termini all'interno del documento penalizzando i documenti in cui le parole chiave sono troppo distanti tra loro anche se presenti.



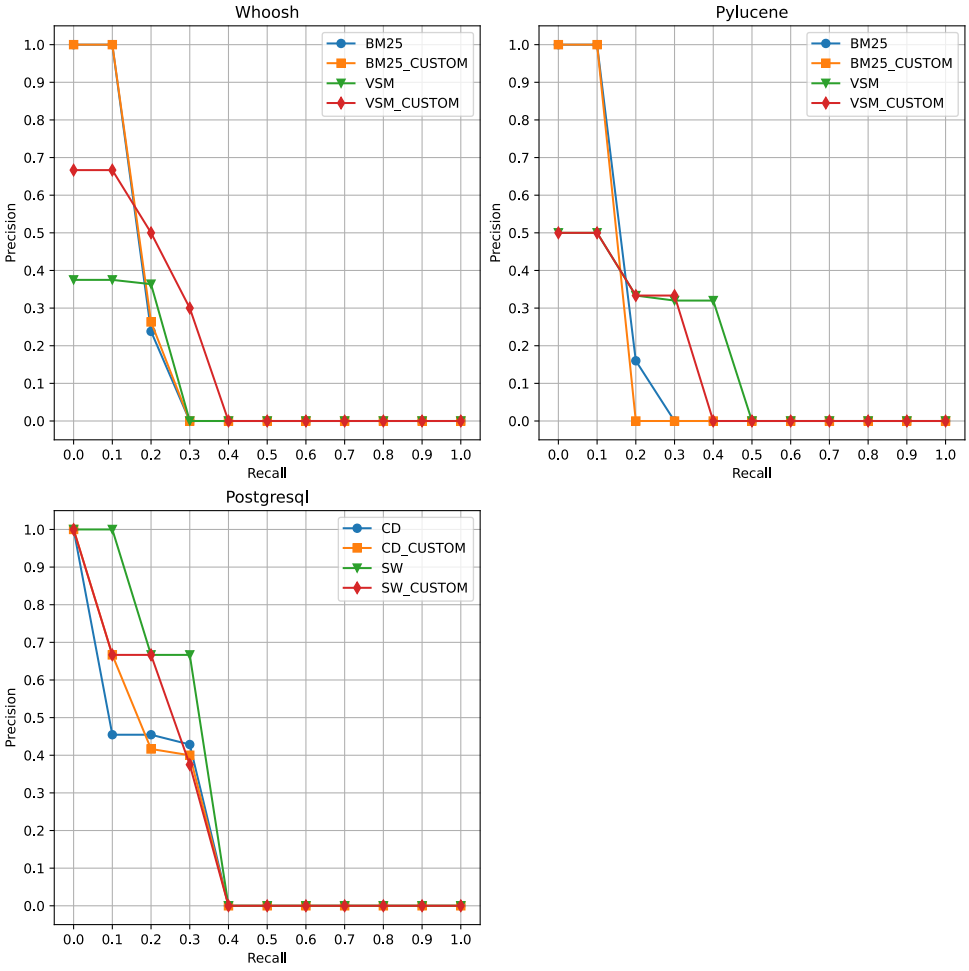
6.3.2.3. Analisi query 4

La query 4 è pensata per stressare il motore di ricerca collegando vari concetti con protocolli specifici. Inoltre diversi documenti trattano ciascun protocollo separatamente.

Il grafico mostra un calo netto della precision nei primi livelli di recall, in alcuni casi si passa da precision 1 a 0 dopo pochi documenti. Questo è segno che non riesce a distinguere con precisione i contenuti rilevanti dopo i primi risultati.

Questo comportamento è dato dal fatto che sono presenti molti termini tecnici nella query (3 protocolli e 2 concetti di sicurezza), la precision viene abbassata perché i documenti che contengono tutti i termini rilevanti sono pochi.

Tenendo in considerazione anche i grafici dell'NDCG, si nota che i valori per la query 4 sono bassi, quindi anche se i documenti rilevanti ci sono sono mal posizionati.



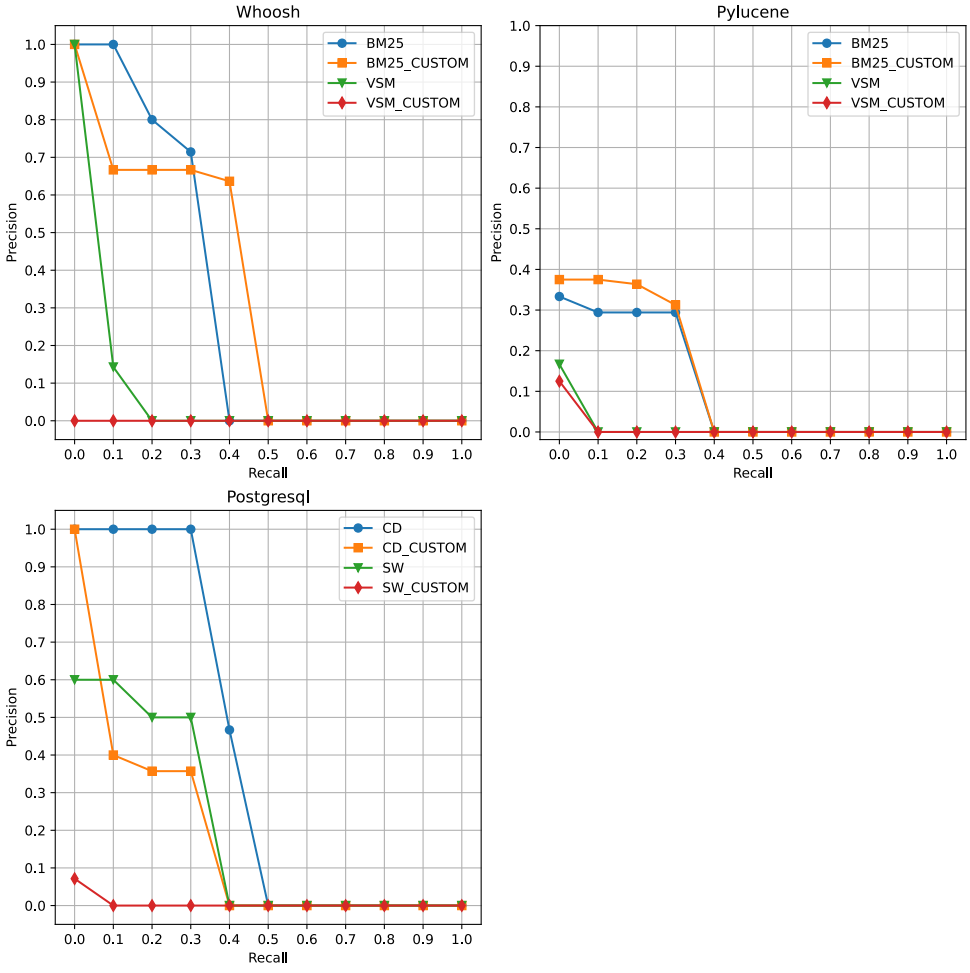
6.3.2.4. Analisi query 5

La query 5 combina due concetti correlati ma trattati in RFC separati. Stressa il motore di ricerca nel riconoscere documenti che parlano di sicurezza nel routing senza escludere BGP.

Qui si osserva un comportamento interessante perché i sistemi di PostgreSQL e Whoosh partono da livelli alti di precision ma calano dopo pochi passi di recall, questo significa che trovano pochi documenti completamente rilevanti mentre il resto dei documenti restituiti potrebbero contenere uno dei termini della query ma non essere completamente rilevante.

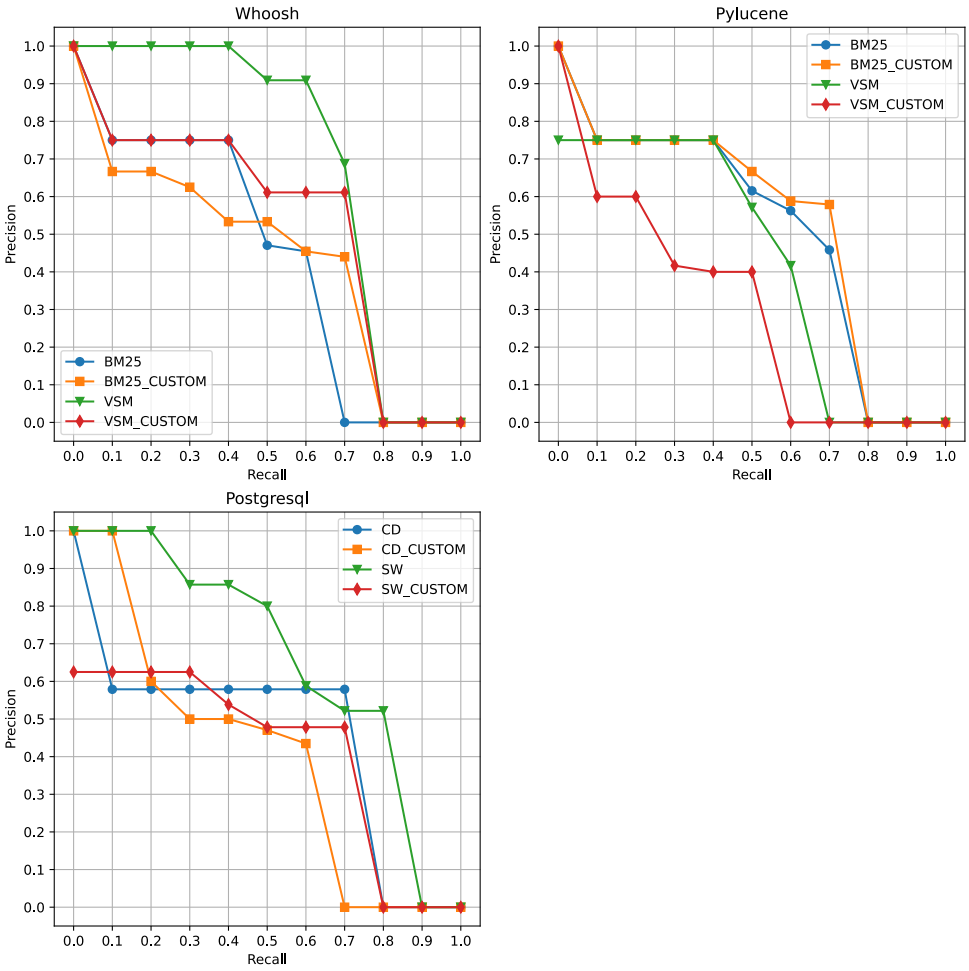
Mentre PyLucene parte subito da livelli molto bassi di precision, questo indica che nessuno dei documenti in cima al ranking viene considerato rilevante. Inoltre si nota che modificando i parametri di default per il modello BM25 riducendo l'impatto sulla lunghezza del documento e aumentando il peso della frequenza dei termini otteniamo risultati leggermente migliori.

Questo significa che per documenti tecnici, un modello BM25 più sensibile alla frequenza dei termini e meno penalizzante sulla lunghezza del documento offre risultati più precisi.



6.3.2.5. Analisi query 7

Nel nostro caso è notevole la query 7: è difficile perché lega concetti di performance e ottimizzazione della rete con QUIC, richiedendo di trovare documenti che parlano di controllo della congestione e del loro impatto su QUIC. Nonostante sia classificata come "difficile" questa query ha dei risultati soddisfacenti: tutti i motori e modelli presentano un calo graduale della precision, sono anche presenti dei plateau (zone piatte) questo indica che il motore di ricerca riesce a distinguere documenti rilevanti e non.
In questo caso i modelli migliori sono il VSM di Whoosh e il SW di PostgreSQL dato che mantengono alta la precision più a lungo.



6.3.3. Complessità della query e impatto sui risultati

È evidente che i modelli di ricerca incontrano maggiori difficoltà quando le query richiedono di collegare più argomenti all'interno di un contesto specifico (come nelle query 2, 3, 4, 5, 6 e 9).
Al contrario, mostrano prestazioni migliori quando la ricerca è formulata in termini più generali, senza vincoli semantici troppo stretti (1, 7, 8 e 10).

Inoltre si può notare come i risultati del BM25 Custom di Whoosh e PyLucene sono simili a quello di default se non migliori, modificando i valori (PyLucene BM25 Custom, Whoosh BM25 Custom), riducendo l'impatto sulla lunghezza e aumentando il peso della frequenza si favorisce la precision a causa della natura dei documenti (molto tecnici e spesso molto lunghi).