

UNIVERSITÀ DI PISA



Dipartimento di Informatica
Corso di Laurea Triennale in Informatica

Sviluppo di un servizio di de-duplicazione di pubblicazioni scientifiche

Tutore Accademico:
Prof. Giovanni Manzini

Candidato:
Michele Velardita

Tutore Aziendale:
Dott. Luca De Santis

Sommario

L'obiettivo principale del tirocinio, svoltosi presso l'azienda Net7 srl, è stato quello di ideare ed implementare un'euristica efficace in grado di riconoscere duplicati di pubblicazioni scientifiche avendo a disposizione esclusivamente i metadati relativi ai documenti. Il lavoro è consistito in una prima fase di ricerca e studio del problema, ampiamente trattato in letteratura, seguito da un'analisi dei dati a disposizione. Avendo infine ideato un'euristica ragionevole, si è passati ad un primo test per validare empiricamente l'efficacia dell'approccio scelto per il riconoscimento dei duplicati (su un dataset di 240 documenti, creato appositamente per le misurazioni) e un successivo test sulla robustezza del codice su un dataset di oltre 150.000 documenti. Il lavoro proseguirà con ulteriori test con lo scopo di migliorare l'approccio ideato per far fronte al problema della deduplicazione ed essere poi in grado di riconoscere i duplicati con una buona accuratezza su un database di oltre 2 milioni di pubblicazioni scientifiche.

Indice

1	Introduzione	3
1.1	Azienda	3
1.2	GoTriple	4
2	Lo stato dell'arte	5
2.1	Presentazione del problema	5
2.2	Risultati odierni	6
3	Tecnologie utilizzate	9
3.1	ElasticSearch	9
3.1.1	Indice ElasticSearch	9
3.1.2	Query DSL	10
3.1.3	Fuzzy Search	11
3.1.4	Elasticvue extension	11
3.2	Java ES API	12
3.3	JSON.simple	12
3.4	junidecode	12
4	Metadati	14
4.1	TRIPLE Data Model	14
4.2	Analisi e scelta dei dati	15
4.2.1	Doi	16

<i>INDICE</i>	2
5 Sviluppo dell'euristica	18
5.1 Limiti degli strumenti	18
5.2 Parametri	19
5.3 Thresholds	20
5.4 Euristica	21
6 Preprocessing	23
6.1 Estrazione dei dati	23
6.2 Riconoscimento del Doi	24
6.3 Normalizzazione	24
7 Fase di implementazione	28
7.1 RestHighLevelClient	28
7.2 Estrazione e normalizzazione dei dati	29
7.3 Aggiornamento	30
7.4 Query ElasticSearch	31
7.4.1 MatchQuery	32
7.4.2 BoolQuery	32
7.4.3 FuzzySearch	33
7.5 Implementazione dell'euristica	35
7.6 Scroll e Paginazione	37
8 Testing e validazione dei risultati	40
8.1 Dataset	40
8.2 Metriche adottate	41
8.2.1 Precision	42
8.2.2 Recall	42
8.2.3 F_1	43
8.3 Osservazioni sperimentali	43
9 Conclusioni	46

Capitolo 1

Introduzione

1.1 Azienda

L'intero progetto di tirocinio è stato svolto presso la Net7 srl, un'azienda di system integration con sede a Pisa con una lunga tradizione di sviluppi di soluzioni informatiche per le Digital Humanities. Con un background di progettazione e realizzazione di collezioni digitali e raccolte di edizioni di immagini, testi e materiale multimediale in ambito umanistico, Net7 ha contribuito alla creazione di reti di collaborazione, nazionali, europee e internazionali, principalmente grazie alla partecipazione a progetti europei, quali: COST A32, che ha permesso di aggregare milioni di manoscritti digitali e pubblicarli in Open Access sul portale Europeana secondo le specifiche Linked Data; Discovery, con lo scopo di sviluppare una federazione di Biblioteche Digitali Semantiche nell'ambito delle Humanities; SemLib – SEMantic tools for digital LIBraries, che ha avuto come obiettivo lo sviluppo di un sistema di annotazione semantica di oggetti digitali e pagine web e di un sistema di recommendation basato su sorgenti Linked Data; o ancora Agora, per lo sviluppo di una federazione di digital libraries Open Access nell'ambito della filosofia Europea e la sperimentazione di tecnologie semantiche e Linked Data su corpus filosofici.

1.2 GoTriple

TRIPLE (“Transforming Research through Innovative Practices for Linked Interdisciplinary Exploration”) è un progetto Europeo lanciato nel 2019 ed è costituito da un consorzio di 19 partner provenienti da 13 paesi europei e in cui Net7 ha la responsabilità dei due workpackages di implementazione informatica.

L’obiettivo del progetto è lo sviluppo di un’innovativa piattaforma di scoperta multilingue e multidisciplinare per le scienze sociali e umane, chiamata GoTriple. Fornisce un unico punto di accesso che consente di esplorare, trovare, accedere e utilizzare materiali come pubblicazioni, dati, progetti e profili di ricercatori su scala Europea[8].

Capitolo 2

Lo stato dell'arte

Il problema della deduplicazione di pubblicazioni scientifiche o, più in generale, di record bibliografici, è ampiamente trattato in letteratura. In questo capitolo verrà introdotto più approfonditamente il problema in questione e verranno presentate, in maniera molto sintetica, alcune ricerche che sono state significative per lo sviluppo della soluzione sperimentata nella tesi.

2.1 Presentazione del problema

Uno dei principali problemi derivanti dalla ricerca in database bibliografici o raccolte di oggetti digitali è il riconoscimento e recupero di duplicati, un compito essenziale per garantire a revisori sistematici uno studio più efficiente delle citazioni, evitando di perdere tempo esaminando lo stesso documento più volte.

Raccogliere pubblicazioni da fonti diverse implica raccogliere documenti descritti da standard di metadati diversi. L'eterogeneità dei record riguarda sia il contenuto che la struttura, inficiando così sulla presenza di duplicati. Uno dei primi ostacoli è proprio quello di analizzare i dati disponibili, confrontando metadati importati da piattaforme diverse, per poi normalizzarli e creare uno standard univoco per tutte le pubblicazioni, predisponendosi in tal modo allo sviluppo di un'euristica affidabile.

2.2 Risultati odierni

La letteratura ci mostra che gli approcci utilizzati per far fronte al problema della deduplicazione sono molteplici, da tecniche di machine learning a euristiche non-supervisionate che non necessitano di nessun tipo di training (il quale può essere a volte dispendioso da sviluppare).

Quest'ultima è la strada intrapresa da un team di ricercatori dell'Università Federale di Rio Grande do Sul[2], che hanno raggiunto risultati statistici molto simili ad approcci basati su processi di machine learning, ideando un set di funzioni di similarità studiate soprattutto per una libreria digitale molto vasta (oltre 1.800.000 metadati relativi a paper scientifici pubblicati in diversi paesi) chiamata DBLP¹.

Per confrontare i metadati di due record, vengono utilizzati tre valori di threshold: t_Y , t_N e t_L , relativi rispettivamente alla distanza tra gli anni di pubblicazione, la similarità tra gli autori e la similarità tra i titoli (quest'ultima misurata con la distanza di Levenshtein).

Lo pseudocodice dell'algoritmo da loro proposto è riportato di seguito:

MetadatMatch(a, b, t_Y, t_N, t_L)

if ($Year(a) - Year(b) \leq t_Y$) **then**

if ($NameMatch(Authors(a), Authors(b), t_N) == 1$) **then**

if ($Levenshtein(Title(a), Title(b)) \geq t_L$) **then**

return 1

dove *Year* indica l'anno di pubblicazione e *NameMatch* un algoritmo che confronta le iniziali degli autori dei due record. Concentrandosi sulle iniziali scavalcano il problema della rappresentazione eterogenea dei nomi degli autori; vengono confrontati solo il primo, il secondo e l'ultimo carattere del nome dell'autore, considerando anche una possibile inversione tra nome e cognome.

La strategia di comparare prima l'anno di pubblicazione, seguito dalle iniziali degli autori ed infine i titoli, riduce drasticamente il tempo di esecuzione dell'algoritmo. Su 1.800.000 record circa, le chiamate alle funzioni *NameMatch* e *Levenshtein* (computazionalmente più costose) sono state rispettivamente 363.000 e 383.

¹La Digital Bibliography Library Project è un sito web che fornisce risorse bibliografiche specialistiche nell'ambito dell'informatica.

Per le loro misurazioni sul Dataset DBLP, hanno utilizzato i seguenti valori di threshold:

- $t_Y = 0$ (stesso anno) or $t_Y = 1$ (un anno di differenza)
- $0.5 \leq t_L \leq 0.7$
- $0.6 \leq t_N \leq 1.0$

ottenendo i seguenti dei valori di Precision, Recall e F_1 (metriche spiegate nel paragrafo 8.2):

t_Y	t_L	t_N	Precision(%)	Recall(%)	F_1 (%)
0	0.7	1.0	88.86	68.60	77.55
1	0.6	0.6	83.90	81.33	82.60

Se comparato con gli approcci precedenti per il riconoscimento di duplicati, questa euristica si comporta leggermente meglio con i dataset di librerie digitali, senza il fardello e il costo di qualunque processo di training. Come già è stato detto, gli algoritmi ideati per il riconoscimento di oggetti digitali deduplicati, e in particolare di pubblicazioni in librerie digitali, sono molteplici. Di seguito viene riportata una tabella[1] che riassume i principali modelli di algoritmi di classificazione sviluppati in seguito e la loro efficacia misurata in termini di Precision, Recall e F_1 (Vedi paragrafo 8.2).

Type	Algorithm	Parameters	P	R	F1	G_{F1}
function	SMO	linear kernel	82.8	95.2	88.6	5.4
function	Multilayer Perceptron	6 internal nodes, 200 epochs	88.9	91.3	90.1	4.7
Bayes	Naive Bayes	default	75.1	98.6	85.3	4.7
meta	AdaBoost.M1	Naive Bayes default	85.5	89.2	87.3	0.0
rules	RIPPER	one optimization phase	86.0	93.0	89.4	-1.5
tree	C4.5	25% confidence for pruning, at least 2 instances per leaf	86.1	95.0	90.3	-1.3

Il maggior beneficio nell'utilizzo di algoritmi di classificazione è sicuramente l'incremento della qualità del processo di deduplicazione. I risultati mostrano come questo approccio, combinato con le funzioni di similarità, migliora statisticamente la precisione e l'accuratezza nel riconoscimento di un record duplicato. Ma se da un lato si ottengono, in media, risultati statistici migliori, dall'altro si presenta il fardello della creazione di un training set per

l'addestramento del modello ed un data set studiato ad hoc per eludere il problema del sovraddattamento. Inoltre non è facile, nel nostro caso, implementare un algoritmo di classificazione a causa della limitazione degli strumenti.

Approach	P	R	F1
unsupervised heuristic-based	83.9	81.3	82.6
classification-based	88.2	95.2	91.6

Figura 2.1: Risultati ottenuti sullo stesso Dataset (Cora Dataset)

Nella figura 2.1 viene mostrato un confronto tra i due approcci (euristica e algoritmo di classificazione) in termini di precision, recall e F_1 .

Capitolo 3

Tecnologie utilizzate

3.1 ElasticSearch

Elasticsearch è un motore di ricerca e analisi distribuito e open source per tutti i tipi di dati, inclusi numerici, testuali, geospaziali, strutturati e non strutturati. Scritto in Java e basato su Apache Lucene (libreria software per motori di ricerca gratuita e open source), è conosciuto per le sue semplici API REST, la sua velocità e la sua scalabilità: Elasticsearch nasce come sistema di ricerca parallelo, perciò la capacità del database può essere aumentata semplicemente aggiungendo nodi al cluster. Inoltre gestisce autonomamente il problema dell'affidabilità, migrando dati o promuovendo dati replica a dati primari al momento del malfunzionamento di un qualsiasi nodo. Queste caratteristiche, unite alla sua capacità di indicizzare molti tipi di contenuti, fanno sì che possa essere utilizzato per una serie assai ricca di casi d'uso.

3.1.1 Indice ElasticSearch

Un indice in Elasticsearch è una raccolta di documenti correlati tra loro. Elasticsearch archivia i dati come documenti JSON, garantendo che la collezione di dati sia semplice sia da leggere che da accedere. Ogni documento ha una serie di chiavi con dei valori corrispondenti (figura 3.1).

Elasticsearch utilizza una struttura di dati chiamata indice invertito, progettata per consentire ricerche full-text molto veloci. Un indice invertito elenca ogni parola univoca che appare

year_of_publication (year_of_publication.keyword)	publish_date	authors (authors.keyword)	number_of_authors
["2011"]	2022-03-17T11:59:29.000Z	["Anne-Valérie Dulac"]	1
["2017"]	2022-03-17T11:59:29.000Z	["Justin A. Joyce"]	1
["2016"]	2022-03-17T11:59:29.000Z	["David W McIvor"]	1
["2019"]	2022-03-17T11:59:29.000Z	["Robert Kilroy"]	1
["2016"]	2022-03-17T11:59:29.000Z	["Eric K. Anderson"]	1

Figura 3.1: Esempio di mapping all'interno di un indice

in qualsiasi documento e identifica tutti i documenti in cui si trova ciascuna parola.

Durante il processo di indicizzazione, Elasticsearch archivia i documenti e crea un indice invertito per rendere i dati del documento ricercabili quasi in tempo reale.

3.1.2 Query DSL

Elasticsearch utilizza un particolare linguaggio basato su JSON per codificare la struttura di una ricerca. A differenza del linguaggio SQL, il query DSL è componibile a piacere in modo da garantire la massima libertà di ricerca. Tuttavia se da una parte questa libertà permette di effettuare ricerche a qualsiasi livello, dall'altra spesso il modo di interrogare il database risulta più complesso (figura 3.2).

La potenza del query DSL risiede nella possibilità di effettuare ricerche di tipo full-text, ovvero il risultato di una ricerca non conterrà soltanto i documenti che soddisfano determinati criteri, ma anche quelli che si avvicinano di una certa percentuale al risultato.

```

{
  "query": {
    "match": {
      "year_of_publication.keyword": "2011"
    }
  },
  "size": 10,
  "from": 0,
  "sort": [
    {
      "index_timestamp": {
        "unmapped_type": "keyword",
        "order": "asc"
      }
    }
  ]
}
```

Figura 3.2: MatchQuery su un indice di Elastic

3.1.3 Fuzzy Search

Dal momento che i computer non sono in grado di comprendere il linguaggio naturale, ci sono una miriade di approcci alla ricerca testuale, ognuno con i propri vantaggi e svantaggi. La Fuzzy Query di Elasticsearch è uno strumento molto potente per una moltitudine di situazioni: ricerche di nomi di persona, errori di ortografia e altri problemi legati al linguaggio naturale possono spesso essere trattati con questa query non convenzionale. Una Fuzzy Query restituisce termini simili al termine di ricerca (misurato in distanza di Levenshtein). Per trovare stringhe simili, la Fuzzy Search crea un set di parole con delle possibili variazioni, a partire dalla stringa di ricerca (figura 3.3).

```
"query": {  
  "fuzzy": {  
    "normalized_title.keyword": {  
      "value": "editorial",  
      "max_expansions": 5  
    }  
  }  
}
```

Figura 3.3: Fuzzy Query sul termine "editorial"

Una ricerca sulla parola "editorial", ad esempio, assegnando gli opportuni valori ai parametri di ricerca, potrà restituire come risultato campi contenenti parole simili o con errori ortografici (editoriale, edtoriale, editoriall, etc...).

3.1.4 Elasticvue extension

Per interfacciarsi al database di Elastic è stata utilizzata l'estensione browser Elasticvue, una gui semplice e open-source. Grazie a questa estensione è molto più facile creare e modificare gli indici Elastic, scrivere delle query per interrogare il database e usare le REST API per gestire i documenti e mandare in esecuzione degli script su un indice o più indici all'interno dello stesso database Elastic.

3.2 Java ES API

È l'API Java per interfacciarsi con Elasticsearch che semplifica la scrittura di richieste complesse e l'elaborazione di risposte. Questa API è molto vasta ed è organizzata in gruppi suddivisi per funzionalità, come si può vedere dalla documentazione di Elasticsearch[4]. I gruppi di funzionalità sono chiamati "namespace", e ogni namespace si trova in un sottopacchetto di *co.elastic.clients.elasticsearch*. Fanno l'unica eccezione le API *search* e *document*, contenuti nel sottopacchetto principale e accessibili dall'oggetto principale *client* di Elasticsearch. Ci sono due tipi di metodi:

- I metodi che fanno parte dell'API (come *ElasticSearch.search()*) derivano dalle rispettive API Json di Elasticsearch e usano lo standard di Java camelCaseNaming.
- I metodi che fanno parte del framework (come *Query.kind*) sono preceduti da un trattino basso per evitare conflitti e distinguere l'API dal framework.

3.3 JSON.simple

Quando si interroga un database Elastic o si recuperano dei documenti, i risultati sono degli oggetti di tipo Json. Nella fase di implementazione è stato necessario eseguire il parsing dei dati restituiti prima di poter effettuare eventuali operazioni su di essi. A tale scopo, è stata utilizzata la libreria Java *Json.simple*[6].

3.4 junidecode

Per la normalizzazione dei dati, in particolare per la conversione di stringhe scritte nello standard Unicode in stringhe scritte nello standard ASCII, è stata utilizzata una libreria Java esterna chiamata *junidecode*[7]. È stata utilizzata principalmente per convertire titoli che non erano scritti usando la tradizionale codifica ASCII a 7-bit (figura 3.4).

Armenian	Հայաստան	Hayastan
Cyrillic	Москва	Moskva
Greek and Coptic	Ελληνικά	Ellenika
Ethiopic	አዲስ አበባ	'aadise 'aababaa
Arabic	هاتف	htf

Figura 3.4: esempio dell'applicazione della libreria junidecode su alcuni alfabeti

Capitolo 4

Metadati

Come è stato già accennato precedentemente nel paragrafo 2.1, quando si raccolgono pubblicazioni da diverse piattaforme e librerie digitali ci si trova a dover lavorare con metadati descritti da standard diversi (che differiscono quindi sia nel contenuto che nella struttura). In questo capitolo verrà brevemente presentato lo standard adottato nel progetto TRIPLE per descrivere i metadati, per poi passare alle osservazioni e analisi effettuate sugli stessi metadati all'interno di ElasticSearch.

4.1 TRIPLE Data Model

Per potere analizzare i vari dati relativi alle pubblicazioni importate da piattaforme diverse è stato adottato un modello ben preciso per raggrupparli e descriverli (riportato in figura 4.1). Tra questi, alcuni sono stati molto più rilevanti ai fini dello sviluppo dell'euristica per il riconoscimento dei duplicati: *Author*, *Identifier*, *Publication date* e *Title*.

	A	B	C	D
1	Data model	Schema.org	Description	Format
3	Abstract	schema:abstract	Abstract or description of the document	{detected_lang, translated, lang, text}
4	Author	schema:author	Authors of the document and link to profile(s). List of author ids.	{family_name, given_name, id, identifier}
5	Contributor	schema:contributor	List of contributors	
6	Document type	schema:additionalType	Type of the document (given by aggregators)	{id, label}
7	Funding reference	schema:funder	List of project ids	
8	Identifier	schema:identifier	List of identifiers	
9	Keywords	schema:keywords	List of producer's keywords	{lang, text}
10	Language	schema:inLanguage	Language of the content	
11	License	schema:license	A license or a type of license	
12	Primary producer	schema:producer	Primary producer of the doc (eg: HALSHS)	{id, label}
13	Publication date	schema:datePublished	Date of publication or creation	
14	Publisher	schema:publisher	List of publishers	
15	Sources information	schema:mentions	Source (free text) from: dc:source, dcterms:source (e.g. journal issue)	
16	Sources URL	schema:isBasedOnURL	Source (HTTP) from: dc:source, dcterms:source (e.g. URL from a publishing platform)	
17	Spatial location of dataset	schema:spatialCoverage	Content's spatial location of collection (list)	{id, label}
18	Subject (MORESS categories)	sioc:topic	List of MORESS Categories (disciplines)	{id, confidence, label}
19	Temporal period of dataset	schema:temporalCoverage	Content's temporal period of collection (list)	{label}
20	Title	schema:headline	Title of document	{detected_lang, translated, lang, text}
21	TRIPLE thesaurus	schema:knownAbout	TRIPLE thesaurus entries	{id, label}
22	URL of full document	schema:url	URL of full text	
23	URL of the landing page	schema:mainEntityOfPage	URL of landing page to the document.	
24	Format	schema:encodingFormat	File format (multiple values)	
25	Aggregator	schema:provider	Aggregator of the doc (eg: Isidore)	

Figura 4.1: TRIPLE Data Model

4.2 Analisi e scelta dei dati

Alcuni metadati sono molto suscettibili a vari tipi di variazioni. Ne è un esempio l'abstract delle pubblicazioni, che può facilmente cambiare tra un articolo e la sua ripubblicazione e può contenere, data la sua lunghezza superiore, più imprecisioni, errori ortografici e abbreviazioni rispetto al titolo. Per sviluppare un algoritmo affidabile ci si è dunque soffermati su campi più "affidabili"; Rispetto all'abstract, titoli di potenziali duplicati possono facilmente essere comparati tra loro, anche tenendo conto di possibili variazioni lessicografiche, grazie all'aiuto

della FuzzySearch , potente strumento di Elasticsearch che permette di esaminare titoli molto simili tra loro (che differiscono tra loro, ad esempio, per l'omissione di una lettera o diversi segni di punteggiatura);

Non è raro, inoltre, trovare variazioni nella rappresentazione del nome di autori. Il nome dello stesso autore può presentarsi in vari modi (figura 4.2): nome e cognome possono essere intercambiati, abbreviati con le iniziali (specialmente se presenta più di un cognome), possono essere preceduti da una sigla (come Jr.) o omessi, o ancora avere errori di ortografia (quest'ultimo è un caso ben più raro).

<code>["Erika Matias Silva","Érik José Ferreira da Silva","Edjane Vieira...</code>	<code>6</code>
<code>["Erika Matias da Silva","Érik José Ferreira da Silva","Edjane Vie...</code>	<code>6</code>

Figura 4.2: Caso di omissione di una parte del cognome trovato in un duplicato in Elastic

Di conseguenza ai fini dell'euristica si è preferito dare più rilevanza al numero degli autori piuttosto che ai nomi. La variazione del numero di autori è un fenomeno assai più raro rispetto a tutte le altre possibili variazioni citate prima.

Altro campo significativo è stato quello degli identificatori, contenente id (univoci e non) relativi alle pubblicazioni. Si è data più importanza al Doi (Digital Object Identifier), codice univoco e persistente associato ad una pubblicazione.

4.2.1 Doi

Il Digital Object Identifier (DOI) è un identificatore univoco digitale, cioè una stringa di caratteri alfanumerici utilizzata per identificare un documento digitale. Il DOI consente di identificare direttamente un oggetto, e non attraverso un suo attributo (come per esempio la sua posizione URL). È costituito da un prefisso ed un suffisso separati dal carattere '/' e non c'è alcun limite alla sua lunghezza. È case insensitive (10.42/quarantadue e 10.42/QUARANTADUE sono considerati uguali) e può contenere ogni tipo di carattere dello standard Unicode. Il prefisso inizia sempre con la stringa '10.' seguita da un codice alfanumerico che indica il registrante. Così come il prefisso, anche il suffisso è formato da una serie indefinita di caratteri alfanu-

merici. Queste caratteristiche relative alla sua struttura sono state cruciali per sviluppare un regex (vedi paragrafo 6.2) in grado di riconoscere e separare il Doi dagli altri identificatori, a volte poco significativi e non univoci (come l'ISSN, che è invece associato ad una collezione di oggetti digitali e non ad un singolo oggetto).

Da un'analisi sui metadati presenti in Elastic è emerso che due duplicati potrebbero comunque avere Doi diversi; ad esempio, uno stesso articolo può essere pubblicato in due repository differenti e avere così Doi diversi. Inoltre, questo identificatore spesso non è presente tra i metadati di un record.

Capitolo 5

Sviluppo dell'euristica

Riconoscere duplicati può essere un compito arduo a causa dell'eterogeneità dei metadati con cui possono essere descritti gli oggetti digitali.

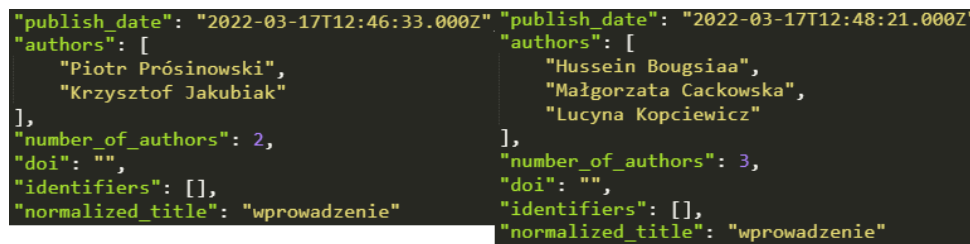
Una volta analizzati i dati presenti all'interno di Elasticsearch, si sono presi in considerazione solo quei campi contenenti metadati significativi allo sviluppo di un'euristica, escludendo dati poco affidabili e spesso soggetti a variazioni di contenuto (come abstract, nome degli autori o sorgente URL).

5.1 Limiti degli strumenti

Java deve interfacciarsi tramite delle Query ad un indice di Elasticsearch, e ciò comporta una certa limitazione nelle scelte di progettazione di un approccio per far fronte al problema della deduplicazione. Non c'è un modo facile e intuitivo per implementare un algoritmo di classificazione simile a quelli proposti precedentemente. L'implementazione di tale tipo di algoritmo necessita dello sviluppo di un insieme di dati di addestramento adeguato e ben ponderato per evitare il problema del sovradattamento (oltre che di un codice più complesso e non facile da gestire e inserire all'interno di tutto il progetto TRIPLE).

5.2 Parametri

All'interno del dominio di biblioteche digitali, il problema della deduplicazione è generalmente basato sulla semantica di specifici metadati. Per esempio, i campi che specificano il nome degli autori sono tra i più discriminatori di un record, di conseguenza possono essere utilizzati come prova di somiglianza per il riconoscimento di duplicati. Possono esistere titoli di libri o pubblicazioni molto simili anche se si riferiscono a due oggetti digitali differenti (figura 5.1);



```

"publish_date": "2022-03-17T12:46:33.000Z" "publish_date": "2022-03-17T12:48:21.000Z"
"authors": [ "authors": [
  "Piotr Prósiniowski", "Hussein Bougsiaa",
  "Krzysztof Jakubiak" "Małgorzata Cackowska",
], "Lucyna Kopciwicz"
"number_of_authors": 2, ],
"doi": "", "number_of_authors": 3,
"identifiers": [], "doi": "",
"normalized_title": "wprowadzenie" "identifiers": [],
"normalized_title": "wprowadzenie"

```

Figura 5.1: record con titoli uguali ma autori diversi

Il nome degli autori può in questo caso fare la differenza per decidere se ci troviamo di fronte ad un falso positivo o meno. Se due pubblicazioni hanno titoli molto simili ma autori diversi, molto probabilmente si riferiscono a due record non duplicati. Ma come è stato detto precedentemente (vedi paragrafo 4.2), un altro problema relativo ai metadati che descrivono un oggetto digitale è la variazione nella rappresentazione dei nomi degli autori (abbreviazioni, inversioni di ordinamento dei nomi e omissioni di suffissi come Jr). Per tale motivo si è preferito dare più importanza al numero degli autori piuttosto che ai nomi veri e propri.

Per rendere l'algoritmo meno restrigente e non incorrere nel problema di avere numerosi falsi negativi, oltre che al numero di autori si è considerato come parametro significativo (in aggiunta al titolo e al Doi) anche l'anno di pubblicazione. Quindi, per ideare una buona euristica ci si è basati su dati che possono essere facilmente analizzati e confrontati tra loro. Di seguito vengono riportati i metadati utilizzati per la deduplicazione:

- **Doi.** Identificatore univoco della pubblicazione. Non sempre presente né uguale tra duplicati dello stesso articolo.
- **Titolo.** Dato molto significativo per il riconoscimento di duplicati. Contrariamente al Doi, un record avrà sempre questo dato.

- **Numero di autori.** Semplice e veloce da confrontare, e difficilmente soggetto ad errori e variazioni (a differenza del nome degli autori).
- **Anno di pubblicazione.** Insieme al numero di autori, dato discriminatorio per la prova di somiglianza tra due record; sono fondamentali in quei casi in cui due record hanno titoli molto simili ma non si riferiscono allo stesso oggetto. Potrebbe però non essere univoco (ad esempio in caso di ripubblicazioni).

5.3 Thresholds

I valori di soglia (o thresholds) servono per gestire errori e problemi nell'acquisizione dei dati (come ad esempio titoli troppo corti per poter essere confrontati), e per "adattare" le funzioni che confrontano i dati di due record; si potrebbero ammettere come duplicati due record con lo stesso titolo ma che differiscono tra loro per l'anno di pubblicazione (entro una certa soglia). Non è difficile trovare, ad esempio, la ripubblicazione di uno stesso articolo da parte di un editore diverso o in una piattaforma diversa con un differente anno di pubblicazione.

Lo stesso ragionamento può essere applicato al numero di autori, anche se l'omissione di uno tra questi è un fenomeno molto meno frequente. Di seguito gli iperparametri utilizzati per l'euristica:

- t_Y : Differenza massima consentita tra due anni di pubblicazione.
- t_A : Differenza massima consentita tra il numero di autori di due record.
- t_{length} : Lunghezza minima del titolo, sotto la quale il campo contenente il titolo del record non viene considerato attendibile per essere utilizzato come parametro dell'euristica.
- t_T : Si riferisce alla "minima somiglianza" necessaria tra due titoli per poter essere considerati uguali.

L'iperparametro t_T verrà poi utilizzato all'interno della FuzzySearch in fase di implementazione. All'aumentare del valore di t_T diminuisce la somiglianza necessaria affinché due titoli siano considerati uguali, e viceversa. Ad esempio, avendo due record che hanno come stringhe,

in un qualche campo, "presentacion" e "presentazione" (troppo brevi per poter essere considerati dati attendibili nell'euristica, ma solo a titolo esemplificativo), con un valore di "minima somiglianza" pari a 5, possono essere considerate identiche.

I threshold sono stati molto utili durante la fase di testing per misurare, al variare dei valori, il comportamento dell'euristica, la sua precisione e la sua accuratezza, tenendo quindi in considerazione falsi positivi (non duplicati riconosciuti dall'algoritmo come duplicati) e falsi negativi (duplicati che non vengono riconosciuti). Nel paragrafo 8.2 verranno spiegate più dettagliatamente le metriche di misurazione.

In aggiunta viene utilizzata una variabile di tipo booleano b_e , per limitare la ricerca di duplicati per quei record molto "poveri" di metadati.

Assume valore *true* se la lunghezza del titolo supera una certa soglia (t_{length}) e se è presente almeno un dato tra anno di pubblicazione e numero di autori. Serve dunque per non incorrere nel problema di riconoscere come duplicati troppi falsi positivi. Non è affatto raro che una pubblicazione che presenta un titolo molto corto (come una rivista o il nome di una collana di pubblicazioni) e manca di metadati quali anno di pubblicazione e autore, venga uguagliata ad altre pubblicazioni con titolo uguale o simile e mancanti anche loro di metadati significativi. Il booleano b_e aiuta a gestire questi casi, diminuendo così il numero di falsi positivi.

5.4 Euristica

Di seguito viene riportato lo pseudocodice dell'euristica ideata. L'algoritmo prende con input i due record a e b da confrontare e gli iperparametri descritti nel paragrafo precedente t_Y , t_A , t_T e t_{length} .

Euristica (a , b , t_Y , t_A , t_T , t_{length})

if ($Doi(A) == Doi(B)$) **then return** 1

$b_e \leftarrow Check(A, t_{\text{length}})$

if ($b_e == true$) **then**

if ($Fuzzy(Title(a), Title(b), t_T) == 1$) **then**

if ($\|Year(a) - Year(b)\| \leq t_Y \vee \|Authors(a) - Authors(b)\| \leq t_A$) **then**

return 1

Il primo step è il confronto del Doi (se presente in entrambi i record). Un risultato positivo rende superfluo il confronto degli altri metadati (essendo il doi un codice univoco). In caso contrario viene effettuato un controllo sugli altri metadati. Se sono presenti dati significativi la variabile booleana di controllo b_e assumerà valore *true* e si procederà quindi al confronto tra i metadati *Title*, *Year* e *Authors* dei due record a e b .

Con $Fuzzy(s_1, s_2, t)$ si indica l'esecuzione della FuzzySearch di Elasticsearch sulle stringhe s_1 e s_2 . Se la stringa s_1 si "avvicina abbastanza" (somiglianza dettata dall'iperparametro t_T) alla stringa s_2 allora darà esito positivo, e si proseguirà con il confronto tra gli anni di pubblicazione e il numero di autori. In caso contrario darà esito negativo e il confronto termina.

Con $Year(a)$ e $Authors(a)$ si indicano rispettivamente l'anno di pubblicazione e il numero di autori del record a . Se, superato il confronto tra i titoli, almeno uno tra gli ultimi due controlli (somiglianza tra anno di pubblicazione e tra numero di autori) ha esito positivo, l'algoritmo termina restituendo 1, ovvero riconosce a e b come duplicati.

L'algoritmo che decide se ci sono dati sufficienti affinché possa essere eseguito un controllo su di essi è il seguente:

Check (A, t_{length})

if ($TitleLength(a) \geq t_{length}$) **then**

if ($Authors(A) > 0 \vee Year(A) \neq null$) **then return** *true*

return *false*

restituisce *true* se la lunghezza del titolo supera una determinata soglia (t_{length}) ovvero se il titolo può essere considerato un dato affidabile per poter essere confrontato, e se è presente almeno un dato tra anno di pubblicazione ($Year(a)$) e numero di autori ($Authors(a)$).

Capitolo 6

Preprocessing

Prima dell'implementazione dell'euristica in Java, c'è stata una fase di preprocessing sui dati. Per poter sfruttare al meglio le potenzialità della FuzzySearch, ad esempio, è stata necessaria una fase di normalizzazione dei titoli. Molti dati significativi (come il Doi), sono stati riconosciuti ed estratti da altri campi di metadati sul database di Elasticsearch. In questo capitolo verranno spiegate tutte le procedure e gli accorgimenti adottati sui dati per poter eseguire in tal modo l'euristica su di un indice di Elasticsearch.

6.1 Estrazione dei dati

Il primo step è stato quello di estrarre da alcuni campi i dati più significativi. Nella figura 6.1 viene riportato il mapping di partenza dei dati memorizzati nell'indice di Elasticsearch.

All'interno del campo *raw_data* erano contenuti gli identificatori e indirizzi url che potevano nascondere possibili Doi (come detto precedentemente tra i dati più importanti in quanto codice univoco associato all'oggetto digitale). Di conseguenza è stato creato un nuovo campo *identifiers* per poter raccogliere in un unico vettore tutti gli identificatori associati ad una pubblicazione. Al momento dell'estrazione degli identificatori sono stati esclusi a priori gli ISSN (presenti in numero considerevole) in quanto non è un codice univoco e un dato affidabile e può anzi generare, se usato all'interno dell'euristica, molti falsi positivi.

Inoltre per semplificare il confronto tra i titoli è stato necessario estrarre dal campo *title_triple* la stringa contenente il titolo vero e proprio. Infatti questo campo conteneva attributi superflui

per la deduplicazione (come la lingua).

6.2 Riconoscimento del Doi

Una parte cruciale della fase di preprocessing dei dati è stata quella di riconoscere e separare il Doi dagli altri identificatori. Conoscendone la struttura regolare, è stato quindi ideato un regex¹ in grado di riconoscere i codici alfanumerici Doi. Il regex ideato e in seguito implementato in Java è il seguente:

- `10.[AlphNum] + /[AlphNum]+`

Il pattern riconoscerà in questo modo come Doi tutti gli identificatori che iniziano con la stringa "10." seguita da uno o più caratteri alfanumerici (`[AlphNum] +`), il carattere "/" ed infine altri caratteri alfanumerici (almeno uno). La possibilità di incorrere in identificatori che seguono lo stesso pattern ma non sono Doi è veramente molto bassa (non è stato trovato neanche un caso in un indice di circa 10.000 documenti).

6.3 Normalizzazione

In alcuni casi è stato trovato il Doi preceduto dall'indirizzo url (ad esempio `https://doi.org/10.1037/a0040251` o `http://dx.doi.org/10.766det/f554aa`). Di conseguenza è stata implementata una funzione per "pulire" il campo contenente il Doi ed estrarre il codice vero e proprio. Data la proprietà case insensitive dell'identificatore, ogni carattere alfabetico è stato portato in lettera minuscola, per evitare il problema (anche se raro) di escludere due pubblicazioni che differiscono tra loro per una stessa lettera maiuscola o minuscola (ad esempio `10.1037/5af` e `10.1037/5Af`) all'interno del campo Doi. Si è passati in seguito alla normalizzazione del titolo, fase essenziale per sfruttare al meglio le potenzialità della Fuzzy Query di Elastic. Queste operazioni evitano di scartare titoli potenzialmente molto simili ma che differiscono tra loro, ad esempio, per un segno di punteggiatura differente, un accento omesso, lo scambio di qualche lettera maiuscola con lettere minuscole o spazi superflui.

¹Un'espressione regolare, o regex, è una sequenza di simboli che identifica un insieme di stringhe

Non si è sovrascritto il campo *Title* presente in Elastic ma è stato aggiunto un nuovo campo, chiamato *normalized_title*, contenente solo il testo del titolo su cui sono state applicate tutte le operazioni di normalizzazione spiegate di seguito.

Come prima operazione sono stati rimossi tutti i segni di punteggiatura, è stato portato tutto il testo in lower case (tutto in minuscolo) e sono stati rimossi eventuali spazi bianchi all'inizio o alla fine del testo. Prendiamo come esempio le due stringhe seguenti, che rappresentano i titoli di due duplicati trovati all'interno di Elasticsearch.

- "V International UNIVEST Conference: The challenges of improving assessment."
- "V International Univest Conference: the challenges of improving assessment"

I due titoli differiscono per il diverso utilizzo di maiuscole e minuscole, e per un segno di punteggiatura finale. Con un threshold t_T adeguato, possono essere riconosciuti come stringhe uguali, ma ciò comporta comunque la possibilità di ignorare stringhe simili (di potenziali duplicati) che differiscono, ad esempio, per l'omissione di una parola, un errore di ortografia o, più probabile in questo caso, l'utilizzo del carattere "5" invece del numero romano "V" all'inizio della stringa.

Si potrebbe pensare di aumentare di conseguenza il valore di "minima somiglianza" tra i titoli t_T per coprire anche questi casi, ma si avrebbe in tal modo una maggiore probabilità di incorrere in falsi positivi (soprattutto per titoli abbastanza corti).

Di seguito vengono riportate le stesse stringhe dopo l'applicazione delle prime operazioni di normalizzazione.

- "v international univest conference the challenges of improving assessment"
- "v international univest conference the challenges of improving assessment"

In questo modo il confronto tra i titoli delle pubblicazioni è molto più immediato, e si riducono anche variazioni eccessive dell'iperparametro t_T (che potrebbe generare troppi falsi positivi).

Un'altra operazione di normalizzazione effettuata sul campo contenente il titolo è stata quella di passare dalla codifica Unicode ad ASCII. Infatti molte pubblicazioni sono scritte in una lingua contenente caratteri che non potevano essere rappresentati in ASCII (un esempio

sono gli ideogrammi cinesi). Per quest'ultima normalizzazione si è utilizzata una libreria Java (junidecode[7]) in grado di convertire, in maniera ragionevole, i caratteri di alfabeti stranieri, come l'arabo, il cinese o il greco, in caratteri dell'alfabeto latino (figura 6.2).

```
"original": "Життя особистості у просторі духовності"
"normalized": "zhittia osobistosti u prostori dukhovnosti"
```

Figura 6.1: Titolo in alfabeto cirillico dopo la normalizzazione

Di seguito viene riportata una tabella che riassume in maniera sintetica le normalizzazioni effettuate sui dati contenuti in Elastic.

Campo	Normalizzazioni
Titolo	rimozione della punteggiatura
	tutto in lower-case
	rimozione degli spazi all'inizio e alla fine della stringa
	conversione da Unicode ad ASCII
Url	riconoscimento ed estrazione del Doi
Identifiers	riconoscimento ed estrazione del Doi
	rimozione di identificatori non univoci
Autori	estrazione del numero di autori

Dopo l'estrazione e il riconoscimento del Doi, la rimozione degli identificatori non univoci (come l'ISSN) o identificatori troppo corti per poter essere considerati attendibili, e dopo tutte le operazioni di normalizzazione effettuate sulle stringhe contenenti i titoli dei record (oltre al conteggio del numero di autori già effettuato in precedenza), si è giunti all'interno del database di Elasticsearch al seguente mapping mostrato nella figura 6.3 (vengono riportati solo i dati utili all'euristica).

title_triple	raw_data (raw_data.keyword)	year_of_publication (year_of_publication.keyword)	number_of_authors
[{"attributes":{"translated":"false","lang":"he","detected_lang":"..."},{"is_based_on_url":[],"identifier":["marc:nli:000040155"],"in_lang...		["2021","2021","2021"]	1
[{"attributes":{"translated":"false","lang":"he","detected_lang":"..."},{"is_based_on_url":[],"identifier":["marc:nli:000040156"],"in_lang...		["2021","2021","2021"]	1
[{"attributes":{"translated":"false","lang":"he","detected_lang":"..."},{"is_based_on_url":[],"identifier":["marc:nli:000040157"],"in_lang...		["2021","2021","2021"]	1
[{"attributes":{"translated":"false","lang":"he","detected_lang":"..."},{"is_based_on_url":[],"identifier":["marc:nli:000040158"],"in_lang...		["2021","2021","2021"]	1
[{"attributes":{"translated":"false","lang":"en","detected_lang":"..."},{"is_based_on_url":[],"identifier":["marc:nli:000040160"],"in_lang...		["2021","2021","2021"]	1
[{"attributes":{"translated":"false","lang":"en","detected_lang":"..."},{"is_based_on_url":[],"identifier":["marc:nli:000040161"],"in_lang...		["2021","2021","2021"]	1
[{"attributes":{"translated":"false","lang":"cy","detected_lang":"..."},{"is_based_on_url":[],"identifier":["marc:nli:000040163"],"in_lang...		["2021","2021","2021"]	0

Figura 6.2: Mapping dei dati prima dell'estrazione

year_of_publication (year_of_publication.keyword)	number_of_authors	doi (doi.keyword)	identifiers (identifiers.keyword)	normalized_title (normalized_title.keyword) ↓
["2019"]	3	10.17853/1994-5639-2019-3-106-124	[]	personallyoriented models of development of musically gifted chil
["2019"]	1	10.17853/1994-5639-2019-3-9-28	[]	personallyoriented approach in the system of education in the hum
["2014"]	3	10.22146/jpki.25231	[]	persepsi terhadap training from senior student dalam penugasan ke
["2015"]	5	10.14516/fde.2015.013.019.007	[]	perfiles profesionales de futuros maestros para el desarrollo sos
["2017"]	1	10.5281/zenodo.1038423	[]	perception of quality in sport qualitative and quantitative appre
["2017"]	2	10.24127/ajpm.v5i2.669	[]	peranan kemampuan verbal dan kemampuan numerik terhadap kemampua

Figura 6.3: Mapping dopo le operazioni di normalizzazione dei dati

Capitolo 7

Fase di implementazione

In questo capitolo verranno mostrate come sono state implementate, in Java, le operazioni di estrazione e normalizzazione dei dati contenuti in Elastic e come è stata implementata l'euristica per riconoscere i duplicati all'interno dello stesso database. Verrà inoltre mostrato come sono state affrontate alcune difficoltà incontrate nel corso della progettazione del codice¹.

7.1 RestHighLevelClient

Per potersi connettere ad Elasticsearch Java utilizza la RestHighLevelClient[9], tramite la quale è possibile inviare ogni tipo di richiesta e gestire le risposte ricevute.

```
final CredentialsProvider credentialsProvider = new BasicCredentialsProvider();
credentialsProvider.setCredentials(AuthScope.ANY,
    new UsernamePasswordCredentials( "?????", "?????" ));
RestClientBuilder builder =
    RestClient.builder(new HttpHost( hostname: "es.tirocini.netseven.it", port: 443, scheme: "https")
        .setHttpClientConfigCallback(httpClientBuilder ->
            httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider)));
RestHighLevelClient client = new RestHighLevelClient(builder);
```

Figura 7.1: connessione al database "tirocini"

Ogni API può essere chiamata in modo sincrono o asincrono. I metodi sincroni restituiscono un oggetto di risposta, mentre i metodi asincroni, richiedono un argomento di tipo

¹L'intero progetto può essere consultato sulla pagina Github: <https://github.com/LinoVelardita/Net7-Internship>

ActionListener che viene notificato (sul pool di thread gestito dal client di più basso livello) una volta ricevuta una risposta o un errore.

7.2 Estrazione e normalizzazione dei dati

Per la normalizzazione e l'estrazione dei dati dai campi di Elasticsearch è stata sviluppata una classe apposita (*NormalizationField*), che, dato in input il campo di Elasticsearch *raw_data* (figura 7.2), effettua il parsing dell'oggetto Json (tramite il metodo `parsing()` contenuto nella classe), estrae gli url, gli identificatori e il testo del titolo contenuti al suo interno ed effettua tutte le operazioni di normalizzazione e riconoscimento degli identificatori spiegati nel Capitolo 6.

```
"raw_data": "{\\"is_based_on_url\\":[],  
"identifier\\":[\\"0102-5473\\",\\"2175-795X\\",\\"10.5007\\\\"/2175-795X.2017v35n4p1296\\",  
"url\\":[],  
"title_triple\\":{\\"attributes\\":{\\"translated\\":\\"false\\",\\"lang\\":\\"\\",\\"detect  
"in_language\\":[\\"en\\",\\"es\\",\\"pt\\"],  
"keywords\\":{\\"attributes\\":{\\"lang\\":\\"en\\"},\\"text\\":\\"juventude\\"},\\"attribu  
"additional_type\\":{\\"text\\":\\"article\\"}},  
"abstract_triple\\":{\\"attributes\\":{\\"translated\\":\\"false\\",\\"lang\\":\\"\\",\\"det  
"main_entity_of_page\\":{\\"text\\":\\"https://\\\\"/doaj.org\\\\"/article\\\\"/be089504a3a84  
"temporal_coverage\\":[],  
"oaipmh_datestamp\\":[\\"2020-11-24T21:44:56Z\\"],  
"date_published\\":[\\"2017-12-01T00:00:00Z\\"],  
"mentions\\":[\\"Perspectiva, Vol 35, Iss 4, Pp 1296-1313 (2017)\\",  
"publisher\\":[\\"Universidade Federal de Santa Catarina\\"],\\"id\\":{\\"text\\":\\"oai:  
"authors\\":{\\"author\\":[\\"Luciana Pedrosa Marcassa\\",\\"Soraya Franzoni Conde\\"]}}
```

Figura 7.2: campo raw_data

Una volta estratto il titolo, quest'ultimo viene passato al metodo *GetNormalizedTitle*, dove vengono applicate in ordine tutte le operazioni di normalizzazione: conversione da Unicode ad ASCII tramite la funzione *unidecode* della libreria *junidecode*[7], rimozione di ogni segno di punteggiatura, conversione di tutti i caratteri alfabetici in minuscolo e rimozione degli spazi superflui all'inizio e alla fine della stringa (tramite il metodo *trim()*[10]).

```
String result = unidecode(title);
result = result.replaceAll("\\p{Punct}", "");
result = result.toLowerCase();
result = result.trim();
return result;
```

In seguito vengono estratti eventuali codici Doi dall'Url e dagli identificatori e vengono riconosciuti, per poi essere scartati dal campo *Identifier*, gli ISSN. Per fare ciò sono stati ideati e testati due regex per il riconoscimento degli identificatori.

```
//regex per il riconoscimento del Doi
Pattern pattern = Pattern.compile(" ^10\\.\\p{Alnum}+/" ,
                                   Pattern.CASE_INSENSITIVE );
Matcher matcher = pattern.matcher(id);
boolean matchFound = matcher.find();
return matchFound;
```

Fortunatamente, riconoscere un ISSN è facile in quanto segue una struttura semplice da identificare; è infatti costituito da quattro cifre numeriche seguite dal carattere "-" e da altre quattro cifre numeriche (l'ultima di queste può eventualmente essere una lettera).

```
//Regex per il riconoscimento dell'ISSN
Pattern pattern = Pattern.compile("[\\d]{4}\\-[\\d]{3}[\\w]" ,
                                   Pattern.CASE_INSENSITIVE );
Matcher matcher = pattern.matcher(id);
boolean matchFound = matcher.find();
return matchFound;
```

7.3 Aggiornamento

Dopo aver implementato e testato una classe Java in grado di estrarre e normalizzare i dati di un singolo record, si è passati all'aggiornamento di un indice Elastic contenente diversi metadati relativi a pubblicazioni.

A questo scopo è stata implementata la classe *UpdateIndex*; si collega con l'API RestHighLevelClient al database "tirocini", e usando la *SearchRequest* viene recuperato ed aggiornato ogni singolo record.

```
SearchRequest request = new SearchRequest("index_name");
SearchSourceBuilder builder = new SearchSourceBuilder();
```



```
builder.query(QueryBuilders.matchAllQuery());  
request.source(builder);  
SearchResponse response = client.search(request, DEFAULT);
```

Per ognuno di questi viene istanziato un oggetto *NormalizationField* a cui viene passato il campo *raw_data* di quel record.

```
raw_data = (String) jsonObject1.get("raw_data");  
NormalizationField nf = new NormalizationField(raw_data);
```

Vengono poi create delle Map in cui vengono salvati i nuovi dati aggiornati e caricati nei rispettivi record.

```
// Doi  
Map<String, Object> updating =  
    singletonMap("doi", nf.getDoi());  
// Titolo normalizzato  
Map<String, Object> updating2 =  
    singletonMap("title", nf.getTitle());  
// updating  
request.doc(updating);  
client.update(request, RequestOptions.DEFAULT);  
request.doc(updating2);  
client.update(request, RequestOptions.DEFAULT);
```

7.4 Query Elasticsearch

Per interfacciarsi con Elasticsearch, si hanno a disposizione diversi tipi di Query. Come è stato accennato precedentemente nel paragrafo 3.1.2, Elastic utilizza un linguaggio basato su Json per implementare la struttura di una ricerca. Ciò garantisce una maggiore libertà di ricerca sacrificando però la semplicità con cui si interroga il database.

7.4.1 MatchQuery

Restituisce tutti i documenti che contengono il testo, il valore numerico o booleano o la data che è stata passata come parametro. Ad esempio, se desideriamo tutti i documenti che contengono nel campo Doi il valore *10.18172/con.630*:

```
QueryBuilder q = MatchQueryBuilder( "doi", "10.18172/con.630" );
```

7.4.2 BoolQuery

Query più complessa rispetto alla classica MatchQuery[3]. Restituisce tutti i documenti che soddisfano le combinazioni booleane dichiarate la suo interno. Viene costruita utilizzando una o più clausole booleane, ciascuna con un'occorrenza tipizzata. I tipi di occorrenza sono:

- **must**. La clausola (query) che deve essere soddisfatta dai documenti restituiti e contribuirà allo score. (Corrisponde all'operatore logico AND)
- **filter**. La clausola (query) che deve essere soddisfatta dai documenti restituiti. Tuttavia, a differenza di must, il punteggio della query verrà ignorato. Utilizzata in casi molto più specifici, non verrà utilizzata per l'implementazione dell'euristica.
- **should**. Almeno una di queste clausole deve comparire nei documenti. (Corrisponde all'operatore logico OR). Si utilizza il parametro *minimum_should_match* per specificare il numero di clausole "should" che devono essere soddisfatte.
- **must_not** La clausola (query) NON deve essere soddisfatta.

Ai fini dell'euristica, avranno maggiore rilevanza le clausole *must* e *should*.

Supponiamo di volere tutti e soli quei documenti che sono stati pubblicati nell'anno 2021 oppure hanno un numero di autori pari a 5 (in formule logiche *year=2021 OR authors=5*). La BoolQuery implementata in Java avrà la forma seguente:

```
BoolQueryBuilder bool_query = QueryBuilders.boolQuery()  
    .should(new MatchQueryBuilder( "Year", 2021))  
    .should(new MatchQueryBuilder( "Authors", 5));
```

Supponiamo adesso di volere tutti e soli quei documenti che hanno come titolo "Deduplicazione" e in aggiunta, come richiesto prima, l'anno di pubblicazione 2021 o esattamente 5 autori (in formule logiche $Title="Deduplicazione" \text{ AND } (Year=2021 \text{ OR } Authors=5)$):

```
BoolQueryBuilder bool_query = QueryBuilders.boolQuery()  
    .must(new MatchQueryBuilder("Title", "Deduplicazione"))  
    .should(new MatchQueryBuilder("Year", 2021))  
    .should(new MatchQueryBuilder("Authors", 5));
```

Se, invece, vorremmo che almeno due delle clausole sopra citate siano soddisfatte, bisognerebbe usare il parametro *minimum_should_match* (di default uguale ad 1):

```
BoolQueryBuilder bool_query = QueryBuilders.boolQuery()  
    .should(new MatchQueryBuilder("Title", "Deduplicazione"))  
    .should(new MatchQueryBuilder("Year", 2021))  
    .should(new MatchQueryBuilder("Authors", 5))  
    .minimumShouldMatch(2);
```

7.4.3 FuzzySearch

La Fuzzy Query di Elasticsearch è un potente strumento per una moltitudine di situazioni, specialmente per problemi legati al linguaggio naturale. Data una stringa di partenza, la fuzzy trova stringhe simili che necessitano al massimo di un certo numero di modifiche ai caratteri per soddisfare la query. Ad esempio una ricerca per "duplicazione", potrebbe restituire, se presente in qualche record e con i giusti parametri, la stringa "deduplicazione", poichè necessita di sole due modifiche per renderla uguale alla stringa di partenza.

Damerau-Levenshtein La metrica utilizzata dalla Fuzzy Query per determinare una corrispondenza tra due stringhe è la distanza di Damerau-Levenshtein; questa distanza corrisponde al numero di inserimenti, eliminazioni, sostituzioni e trasposizioni necessarie per far corrispondere una stringa all'altra.

La formula della distanza di Damerau-Levenshtein è una modifica della classica formula della distanza di Levenshtein, con l'aggiunta dell'operazione di trasposizione, ovvero lo

scambio di due caratteri vicini (la differenza tra "ab" e "ba" è pari a uno). In Elastic è comunque possibile specificare quale algoritmo di matching utilizzare modificando il parametro "transposition" all'interno della FuzzyQuery.

Parametri La Fuzzy Query utilizza al suo interno diversi parametri:

- **value.** Termine da ricercare.
- **fuzziness.** Numero di modifiche ammesse.
- **max_expansions.** Massimo numero di variazioni create. Il valore di default è 50.
- **prefix_length.** Numero di caratteri iniziali che devono rimanere invariati durante la creazione delle espansioni.
- **transposition.** Indica se tra le modifiche ammesse è inclusa la trasposizione ($ab \rightarrow ba$). Di default è true.
- **rewrite.** Metodo utilizzato per riscrivere la query.

I parametri **fuzziness** e **max_expansions** sono i più cruciali per la definizione dell'euristica. La **fuzziness** è il numero di modifiche di un carattere che devono essere apportate a una stringa per renderla uguale a un'altra. Se viene utilizzato "AUTO" all'interno del parametro, la Fuzzy genera una distanza di modifica basata sulla lunghezza del termine da ricercare. Il parametro **max_expansions** definisce il numero massimo di termini che la query controllerà prima di interrompere la ricerca. Un valore troppo alto può avere effetti drammatici sulle prestazioni della fuzzy. La riduzione dei termini della query ha tuttavia un effetto negativo, in quanto alcuni risultati validi potrebbero non essere trovati.

Fondamentalmente, quando si utilizza questo parametro all'interno della fuzzy si sta limitando il numero di termini considerati nella ricerca. Da ciò si evince l'importanza di questo parametro nell'implementazione dell'euristica per il riconoscimento di duplicati. Saranno invece ignorati a questo scopo i parametri **prefix_length** e **rewrite**.

7.5 Implementazione dell'euristica

È stata implementata la classe *Heuristic* che, passati come parametri il Doi, il titolo, l'anno di pubblicazione, il numero di autori e i valori di threshold (t_T , t_{length} , t_A e t_Y) costruisce una query in base ai valori ricevuti, per simulare l'algoritmo di riconoscimento dei duplicati, riportato di seguito:

Euristica ($a, b, t_Y, t_A, t_T, t_{length}$)

if ($Doi(A) == Doi(B)$) **then return** 1

$b_e \leftarrow Check(A, t_{length})$

if ($b_e == true$) **then**

if ($Fuzzy(Title(a), Title(b), t_T) == 1$) **then**

if ($\|Year(a)-Year(b)\| \leq t_Y \vee \|Authors(a)-Authors(b)\| \leq t_A$) **then**

return 1

Per il confronto del Doi viene implementata una MatchQuery sul campo ElasticSerch *doi.keyword*.

```
if ( doi != null ) {
    MatchQueryBuilder doi_query =
        new MatchQueryBuilder ( "doi.keyword" , doi );
}
```

Per verificare l'attendibilità degli altri parametri viene effettuato un controllo sulla lunghezza del titolo, sull'anno di pubblicazione e sul numero di autori.

```
boolean exists_data = false;
if ( title.length() > title_size_threshold )
    if ( ! year_of_publication.isEmpty() || n_authors != 0 )
        exists_data = true;
```

La variabile *exists_data* assumerà valore true solo nel caso in cui la lunghezza del titolo supera la soglia *title_size_threshold* (t_{length}) e se è presente almeno un dato tra anno di pubblicazione e numero di autori. Se la variabile booleana assume valore false, non viene implementata nessun'altra query e l'algoritmo termina.

Il metodo *BuildTitleQuery* della classe *Heuristic* costruisce la *FuzzyQuery* a partire dal titolo e dall'iperparametro t_T .

```
FuzzyQueryBuilder title_query =
    new FuzzyQueryBuilder("normalized_title.keyword", title)
        .fuzziness(Fuzziness.AUTO)
        // soglia per la somiglianza del titolo
        .maxExpansions(title_threshold)
        .transpositions(true)
        .prefixLength(0)
        .rewrite("constant_score"); // valore di default
```

Il valore di soglia t_T è utilizzato all'interno del parametro della *FuzzyQuery* *max_expansions* per definire il grado di somiglianza del titolo, e viene ammessa la trasposizione come operazione di modifica per misurare la distanza tra due stringhe.

Il metodo *BuildAuthorsQuery* costruisce, dato il numero di autori e l'iperparametro t_A , una *BoolQuery* per il confronto del numero di autori.

```
BoolQueryBuilder authors_query = QueryBuilders.boolQuery()
    .should(new MatchQueryBuilder("number_of_authors", authors));

if(authors_threshold == 1){ //threshold per il numero di autori
    authors_query.should(
        new MatchQueryBuilder("number_of_authors", authors + 1));
    authors_query.should(
        new MatchQueryBuilder("number_of_authors", authors - 1));
}
authors_query.minimumShouldMatch(1);
```

Ho assunto che il valore di threshold t_A possa assumere come valori 0 o 1. Se uguale ad 1 vengono considerati anche i record con un numero di autori la cui differenza è al più 1.

Infine, il metodo *BuildYearQuery* costruisce, dati l'anno di pubblicazione e l'iperparametro t_Y , una *BoolQuery* (molto simile a quella ideata per il numero di autori).

```
BoolQueryBuilder year_query = QueryBuilders.boolQuery()
    .should(new MatchQueryBuilder("year_of_publication", year));

if(years_threshold == 1){ //threshold per l'anno di pubblicazione
    year_query.should(
        new MatchQueryBuilder("year_of_publication", year + 1));
    year_query.should(
        new MatchQueryBuilder("year_of_publication", year - 1));
}
year_query.minimumShouldMatch(1);
```

Analogamente per quanto fatto per il numero di autori, anche qui il valore di threshold t_y può assumere come valore 0 o 1, ammettendo così al più un anno di differenza.

title_query, authors_query e year_query vengono poi messe insieme all'interno di una query booleana.

```
BoolQueryBuilder sliceQuery = QueryBuilders.boolQuery()
    .must(title_query)
    .should(year_query)
    .should(authors_query)
    .minimum_should_match(1);
```

In questo modo verranno restituiti tutti e soli i documenti che soddisfanno la title_query e almeno una tra le clausole definite dalla year_query e la authors_query.

7.6 Scroll e Paginazione

Un problema importante cui si è dovuto far fronte è stato quello della presenza di indici contenenti un grande numero di record. In particolare si sono riscontrati problemi con un indice contenente oltre 150.000 record. Il classico metodo di estrazione e aggiornamento dei dati, utilizzato in precedenza per indici di Elasticsearch più piccoli, non dava il responso che ci si aspettava. Di conseguenza è stata necessaria una modifica al codice, in particolare al modo con

cui venivano estratti i record dall'indice. Precedentemente, si utilizzava un semplice costrutto *for* che, avendo prima impostato i parametri degli oggetti `builder.from()` e `builder.size()`, scorreva tutte le *hit* (ovvero i documenti che soddisfavano la clausola) contenute nell'oggetto `SearchHits` restituito dal client.

```
builder.from(0);
builder.size(10.000);
for(SearchHit hit : response.getHits()) {
    //parsing every hit
    // ...
}
```

Nell'esempio del codice riportato sopra, venivano controllati esattamente 10.000 documenti (valore definito da `builder.from(0)` e `builder.size(10.000)`). Ma, come si legge dalla documentazione di Elastic[5], per indici contenenti un maggior numero di documenti è necessario applicare metodi di paginazione dei risultati.

ElasticSearch mette a disposizione, a tale scopo, la *Scroll API*, una libreria che aiuta a gestire indici particolarmente voluminosi. Di seguito viene riportato un esempio dell'utilizzo della scroll.

```
SearchRequest request = new SearchRequest(index_name);
SearchSourceBuilder builder = new SearchSourceBuilder();
builder.query(query);
builder.from(0);
builder.size(1000);
request.source(builder);
request.scroll(TimeValue.timeValueMinutes(15));
SearchResponse response = client.search(request, DEFAULT);
SearchHits hits = response.getHits();
boolean hasNext = false;
//primo ciclo della scroll
for (SearchHit hit : response.getHits()) {
```



```
//do something
hasNext = true;
}

//scroll Id per gestire pi richieste
String scrollId = response.getScrollId();

while (hasNext) {
    hasNext = false;
    SearchScrollRequest scroll_request = new SearchScrollRequest(scrollId
    scroll_request.scroll(TimeValue.timeValueMinutes(15));
    SearchResponse response = client.scroll(scroll_request, DEFAULT);

    for (SearchHit hit : searchScrollResponse.getHits()) {
        //do something
        hasNext = true;
    }
    scrollId = searchScrollResponse.getScrollId();
}
```

In questo caso, vengono controllati 1000 documenti per ogni ciclo (*builder.size(1000)*), con un limite di tempo per ogni iterazione di 15 minuti (*TimeValue.timeValueMinutes(15)*). Dopo ogni iterazione si effettua il controllo sulla variabile booleana *hasNext* per verificare se nell'indice ci sono ulteriori record, e viene salvato lo *scrollId* (*scrollId = searchScrollResponse.getScrollId()*) per continuare dall'ultimo record restituito.

L'implementazione della Scroll come metodo di paginazione dei risultati è stato molto cruciale per i test sulla robustezza del codice, soprattutto se si pensa che, in futuro, un indice di Elasticsearch potrà contenere tra i 15 e i 20 milioni di record di metadati relativi a pubblicazioni scientifiche, e saper gestire correttamente una tale mole di dati è fondamentale per il buon funzionamento dell'euristica.

Capitolo 8

Testing e validazione dei risultati

Dopo aver sviluppato e implementato un metodo in grado di aggiornare e estrarre i dati necessari all'euristica, e aver implementato quest'ultima, si è passati ad una fase di test e validazione dei risultati ottenuti. Sono stati misurati gli effetti dell'euristica al variare dei valori di threshold adottando un adeguata metrica di misurazione. Lo scopo principale delle misurazioni effettuate è quello di riuscire a migliorare l'approccio scelto per affrontare il problema della deduplicazione, osservando come si comporta l'algoritmo al variare degli iperparametri su un dataset ben definito.

8.1 Dataset

Per verificare i risultati ottenuti con l'euristica è stato creato un dataset contenente 240 record studiato ad hoc, di cui si conoscevano con esattezza tutti i duplicati. È stato quindi creato un foglio di calcolo contenente gli id di tutti i documenti, il numero di duplicati che ogni documento aveva all'interno del dataset e il suo numero di cluster, per trovare più facilmente i duplicati del documento originale.

Come mostrato nella figura 8.1, per ogni record si conosce il suo id, il numero di duplicati e l'id del cluster contenente quegli stessi duplicati. La creazione dell'insieme di record è nata da alcune osservazioni effettuate su un indice molto più grande (contenente più di 150.000 record). Di seguito vengono riportate alcune considerazioni:

ID	CLUSTER ID	#DUPLICATES
oai:doaj.org_article:33090d581aa44faab7f0360667e7daba	1	1
oai:doaj.org_article:0fa28032072a455990f1f6040821e206	1	1
oai:doaj.org_article:c1a97a75ed5040678de061e098d057f7	2	1
oai:doaj.org_article:9cf23c7c62f34cefbfc0a9f04ee87a56	2	1
oai:doaj.org_article:2d9f5462f21a488dbfe8c82071c12a25	0	0
oai:doaj.org_article:35a226cab615441796e16aa1f1afb7ac	3	2
oai:doaj.org_article:d0fa3023d7ea4f7db051a72f1cdc85a2	3	2
oai:doaj.org_article:c818ebf37d684d9b9ee0b1ec1e370d6f	3	2

Figura 8.1: primi record del dataset

- È molto raro riuscire ad individuare duplicati di un documento solamente grazie al Doi. Lo stesso codice univoco, infatti, difficilmente è presente in più di un documento.
- Possono essere presenti dei record che rappresentano delle riviste e che hanno la maggior parte delle volte, titoli molto corti. In aggiunta, può capitare che dei metadati molto simili (stesso titolo e anno di pubblicazione, ma mancante del numero di autori) si riferiscano ad articoli diversi della stessa rivista. Questo genera spesso numerosi falsi positivi e falsi negativi e sono i casi più difficili da trattare (l'unico modo sarebbe rendere ancora più restrigente l'euristica; ma se da un lato ci si concentra su questo particolare problema, dall'altro si ha un aumento considerevole di falsi negativi).
- Molto raramente i documenti presentano degli identificatori (oltre al Doi) abbastanza lunghi per poter essere considerati univoci.
- È molto raro trovare duplicati della stessa pubblicazione che differiscono per il numero di autori. Quest'ultimo sembra essere il dato più attendibile e meno soggetto a variazioni. Per questo motivo si è preferito concentrarsi più sugli altri iperparametri e lasciare pressoché invariato (per gli esperimenti sul dataset da 240 record) il threshold t_A .

8.2 Metriche adottate

Dopo aver creato il dataset su cui condurre i test era necessario scegliere delle metriche di misurazione adeguate. Era importante tenere conto anche della "qualità" dei documenti restituiti, ovvero tenere conto dei falsi positivi (quei record riconosciuti erroneamente come duplicati

dall'euristica) e falsi negativi (quei record duplicati, ma che non vengono riconosciuti come tali).

La scelta è ricaduta sull'utilizzo della precisione, della recall e dell'accuratezza (detta anche F_1 -misura), in quanto riescono meglio a descrivere il comportamento dell'euristica basandosi sul numero di documenti riconosciuti come duplicati e tenendo conto dei falsi positivi e falsi negativi. Tutte e tre le metriche possono assumere valori compresi tra 0 e 1.

8.2.1 Precision

La precisione è definita come il numero di documenti attinenti recuperati da una ricerca (i veri positivi) diviso il numero totale di documenti recuperati dalla stessa ricerca (i veri positivi e i falsi positivi)[11]. In formule:

$$precision = \frac{true\ positive}{true\ positive + false\ positive}$$

Un valore di precisione di 1.0 significa che ogni risultato recuperato da una ricerca è attinente, ovvero, nel nostro caso, tutti i documenti recuperati e quindi riconosciuti dall'euristica come duplicati sono effettivamente duplicati. Ma la precisione non tiene conto dei possibili falsi negativi. Se, ad esempio, su 10 documenti recuperati, tutti e 10 sono effettivamente duplicati, ma nel dataset sono presenti 40 documenti duplicati, la precisione avrà comunque valore massimo.

8.2.2 Recall

La recall è definita come il numero di documenti attinenti recuperati da una ricerca diviso il numero totale di documenti attinenti esistenti (ovvero tutti i record duplicati che dovrebbero essere stati recuperati). In formule:

$$recall = \frac{true\ positive}{true\ positive + false\ negative}$$

Un valore di recall pari a 1.0 significa che sono stati recuperati tutti i record duplicati presenti nel dataset, ma senza tener conto dei possibili falsi positivi.

8.2.3 F_1

Rappresenta il grado di accuratezza del test. Tiene in considerazione sia la precisione che il recupero (o recall), e viene calcolato tramite la media armonica di precisione e recupero:

$$F_1 = 2 \times \frac{\text{precision} + \text{recall}}{\text{precision} \times \text{recall}}$$

Assumerà valore massimo solo nel caso in cui precisione e recupero assumono anch'essi valore massimo, ovvero se vengono recuperati tutti e soli i documenti duplicati (nessun falso positivo o falso negativo).

8.3 Osservazioni sperimentali

Nella fase di test è stata posta maggiore enfasi al threshold t_{length} , il quale specifica la lunghezza minima che deve avere il titolo per poter essere considerato un dato attendibile, e al threshold t_T , che specifica il grado di somiglianza minimo che deve avere una stringa per poter essere considerata uguale alla stringa di partenza. A causa del numero limitato di record presenti nel dataset, la variazione dei valori degli altri iperparametri non permette di fare delle osservazioni precise sulla variazione delle metriche di misurazioni scelte. In figura 8.2 viene riportata una tabella riassuntiva del test condotto sul dataset.

	A	B	C	D	E	F	G	H	I	J	K
1	SizeTitle t.	Similar title t.	Authors t.	Year t.	Precision	Recall	F1	true +	false +	false -	TOT MATCH
2	18	200	1	1	0.88	0.88	0.88	37	5	5	42
3	18	1000	0	1	0.88	0.88	0.88	37	5	5	42
4	28	500	0	1	0.92	0.83	0.87	35	3	7	38
5	24	1000	0	1	0.87	0.81	0.84	34	5	8	39
6	22	1000	0	1	0.85	0.83	0.84	35	6	7	41
7	8	500	0	1	0.82	0.95	0.88	40	9	2	49

Figura 8.2: test

Vengono prima riportati i valori degli iperparametri, seguiti dai valori delle metriche calcolati e dal numero di veri positivi, falsi positivi, falsi negativi e il numero totale di documenti restituiti.

Come si evince dai risultati, un valore di t_{length} troppo alto aumenta la precisione del test ma diminuisce molto il recupero (aumentano cioè i falsi negativi). Viceversa, un valore t_{length} troppo basso causa un aumento considerevole di falsi positivi (una precisione più bassa), ma allo stesso tempo genera un aumento del recupero.

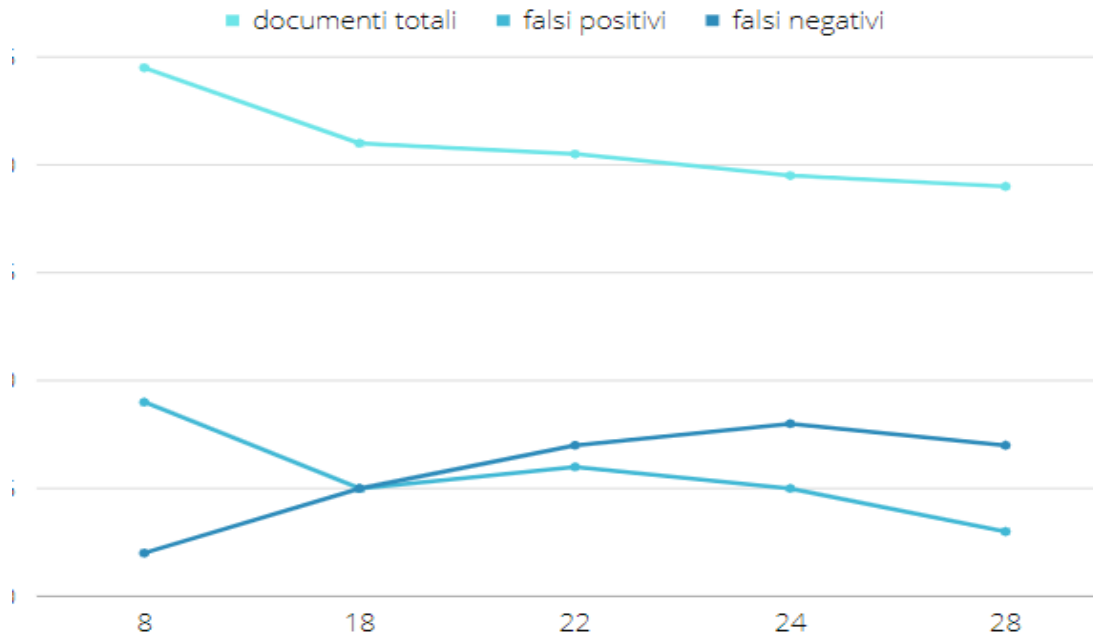


Figura 8.3: falsi positivi e falsi negativi al variare di t_{length}

La figura 8.3 mostra più visivamente come variano i falsi positivi e i falsi negativi all'aumentare di t_{length} . Si noti come un aumento di questo iperparametro comporta, da un lato, un conseguente aumento di falsi negativi, mentre dall'altro un decremento di falsi positivi.

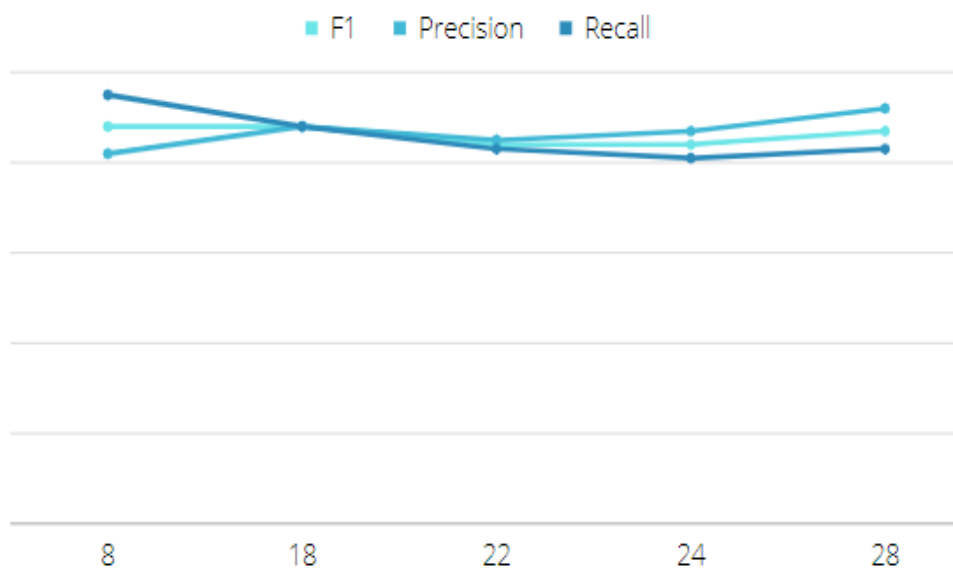


Figura 8.4: Precision, recall e F_1 al variare di t_{length}

La figura 8.4 mostra invece come variano precisione, recupero e accuratezza al variare di t_{length} . Si noti come un aumento dell'iperparametro comporta, in questo caso, un aumento della precisione con un conseguente decremento del valore di recall.

Capitolo 9

Conclusioni

Il lavoro svolto durante il tirocinio non è stato solo puramente pratico. Si è trattato di un'attività di analisi, approfondimento e studio dello stato dell'arte. Poiché il tirocinio si è svolto nell'ambito di un progetto europeo, durante la fase di sviluppo dell'euristica ho avuto modo di avere un confronto di idee e opinioni non solo con il team della Net7 ma anche con un team di sviluppo francese coinvolto nel progetto. È stata un'importante possibilità di apprendimento di nuove conoscenze e tecniche in ambito informatico e di avvicinamento ad un contesto lavorativo moderno e stimolante, dove niente è lasciato al caso ed ogni piccolo dettaglio è studiato e inserito perfettamente all'interno di un contesto molto più ampio.

Posso affermare di non aver solo "imparato" nuovi strumenti e tecniche di sviluppo e progettazione, ma ho avuto modo di maturare un'esperienza professionale non di poco conto per il breve tempo che ho passato sotto l'attenta e costante supervisione del mio tutore aziendale e dei miei colleghi. Inoltre, l'occasione di poter partecipare ad un progetto di ricerca internazionale è stato indubbiamente un ulteriore valore aggiunto del tirocinio.

Tutto il lavoro svolto fino ad ora ha gettato delle basi solide per poter utilizzare l'euristica implementata all'interno del progetto TRIPLE. Saranno tuttavia necessari ulteriori test per migliorare la qualità del servizio di duplicazione e un'analisi più approfondita dei casi di fallimento per garantire l'affidabilità e la robustezza del codice implementato.

Bibliografia

- [1] Galante Becker Heuser. “An Automatic Approach for Duplicate Bibliographic Metadata Identification Using Classification”. In: *Computational Science Center Federal University of Rio Grande Rio Grande* 3.24 (2011). DOI: 10.1109/SCCC.2011.8. URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6363382>.
- [2] Carvalho Borges. “An unsupervised heuristic-based approach for bibliographic metadata deduplication”. In: *Institute of informatics, Federal University of Rio Grande do Sul* 3.24 (feb. 2011). DOI: 10.1016/j.ipm.2011.01.009. URL: <https://www.sciencedirect.com/science/article/pii/S0306457311000100>.
- [3] Elasticsearch Documentation. *BoolQuery*. URL: <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html>.
- [4] Elasticsearch Documentation. *Java ES API*. URL: <https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/current/index.html>.
- [5] Elasticsearch Documentation. *Scroll API*. URL: <https://www.elastic.co/guide/en/elasticsearch/reference/current/scroll-api.html>.
- [6] Java Documentation. *Json.simple*. URL: <https://javadoc.io/doc/org.json/json/latest/index.html>.
- [7] gcardone. *junidecode library*. URL: <https://github.com/gcardone/junidecode>.

- [8] GoTriple.eu. *GoTriple*. URL: <https://www.gotriple.eu/>.
- [9] javadoc. *RestHighLevelClient*. URL: <https://javadoc.io/static/org.elasticsearch.client/elasticsearch-rest-high-level-client/7.6.0/org/elasticsearch/client/RestHighLevelClient.html>.
- [10] javadoc. *Trim()*. URL: [https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#trim\(\)](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#trim()).
- [11] Wikipedia. *Precision and Recall*. URL: <https://en.wikipedia.org/wiki/Precisionandrecall>.