UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Gestionale

INTEGRAZIONE DATI BIBLIOMETRICI

Integration of bibliometric data

Relatore:

Potena Domenico

Tesi di laurea di:

Petrelli Matteo

A.A. 2022/2023

Indice

1. Introduzione 4

2. Strumenti utilizzati 6

2.1 Python 6

2.2 Mysql 8

2.3 Scopus 8

3. Estrazione dati da scopus 9

3.1 Inserimento dati degli autori nel database 9

3.2 Inserimento dei dati dei paper nel database 14

4. Confronto Dati 20

4.1 Pubblicazioni con identificativo scopus non trovati nell'estrazione di dati 20

4.2 Controllo eventuali errori 26

5. Conclusioni e sviluppi futuri 29

Sitografia 32

Appendice 33

Codice 1: Inserimento dati autore iniziali 33

Codice 2: Completamento inserimento dati autori 33

Codice 3: Estrazioni dati paper da scopus 35

Codice 4: Estrazione paper non trovati da scopus 38

Codice 5: Confronto titoli 42

Codice 6: Controllo errori 44

Indice delle Tabelle

Tabella 3.1: Esempio tabella autori iniziale 10

Tabella 3.2: Struttura tabella ‘Autori’ 10

Tabella 3.3: Esempio record tabella ‘Autori’ 14

Tabella 3.4: Struttura tabella ‘paper’ 15

Tabella 3.5: Struttura tabella ‘Autor_ship’ 17

Tabella 3.6: Esempio record della tabella ‘paper’ 19

Tabella 3.7: Esempio record della tabella ‘Autor_ship’ 19

Tabella 4.1: Esempio di risultato della precedente query SQL 21

Tabella 4.2: Struttura della tabella ‘errore_base_eid’ 26

Tabella 4.3: Esempio record tabella ‘errori_base_eid’ 28

+

Capitolo Primo

INTRODUZIONE

Nell'era digitale, l'accesso a un'enorme quantità di informazioni bibliografiche è diventato essenziale per i ricercatori e gli studiosi. Scopus, uno dei più grandi database bibliografici multidisciplinari, offre una vasta gamma di pubblicazioni scientifiche provenienti da tutto il mondo. Tuttavia, nonostante l'ampiezza e la copertura di Scopus, l'integrità e l'accuratezza dei dati bibliografici presenti possono essere compromesse a causa di vari fattori, come errori di inserimento, duplicazioni e discrepanze tra diverse fonti di informazione.

Il confronto e l'integrazione dei dati bibliografici di Scopus rivestono un ruolo cruciale nel garantire la coerenza e l'affidabilità delle informazioni presenti nel database. Il confronto accurato dei dati provenienti da diverse fonti può aiutare a identificare e correggere duplicati, eliminare errori di formattazione e garantire la coerenza delle informazioni tra le pubblicazioni. Allo stesso tempo, l'integrazione dei dati bibliografici provenienti da Scopus con altre fonti di dati può arricchire il contenuto informativo, fornendo una visione più completa e approfondita delle pubblicazioni scientifiche.

In particolare, si intende spiegare le metodologie utilizzate per effettute un confronto tra i dati caricati su Scopus, i cui autori fanno parte del Dipartimento dell'Ingegneria dell'Informazione dell'Università Politecnica delle Marche, e i dati appartenenti al Database IRIS, in cui vi sono salvati i dati di ogni pubblicazione di tutti gli autori che sono associati all'Università in questione, con l'obiettivo di migliorare l'accuratezza e l'integrità delle informazioni bibliografiche disponibili. Saranno esaminate tecniche e metodologie avanzate per il confronto automatico dei dati, compresi algoritmi di confronto e misure di similarità specificamente progettati per dati bibliografici.

In conclusione, questa tesi si pone l'obiettivo di esplorare e affrontare le sfide legate al confronto e all'integrazione dei dati bibliografici di Scopus, con l'obiettivo di migliorare la coerenza, l'accuratezza e l'affidabilità delle informazioni presenti nel database fornendo strumenti e metodologie che semplifichino il processo di confronto e integrazione dei dati bibliografici. Attraverso l'uso di metodologie avanzate e l'applicazione di tecniche innovative, si spera di fornire un contributo significativo al campo della gestione delle informazioni scientifiche.

Capitolo Secondo

STRUMENTI UTILIZZATI

In questo capitolo si intendono descrivere gli strumenti utilizzati per la raccolta dei dati e per il seguente confronto dati tra il database Iris, in cui vi sono i dati riguardanti le pubblicazioni di interesse e i dati estratti da Scopus.

2.1 Python

I dati da Scopus riguardanti gli autori che appartengono al Dipartimento dell'Informazione delle Università Politecnica delle Marche sono stati estratti attraverso un codice Python.

Python è un linguaggio di programmazione ad alto livello pseudocompilato, il che significa che un interprete analizza direttamente il codice sorgente senza una fase di compilazione separata.

I principali punti di forza sono:

- Sintassi semplice e leggibile: Python ha una sintassi pulita e intuitiva, che lo rende facile da apprendere e leggere.
- Ampia libreria standard: Python dispone di una vasta libreria standard che offre moduli e funzioni pronte all'uso per una varietà di scopi. Questa libreria copre una vasta gamma di aree, come manipolazione

dei file, networking, elaborazione dei dati, sviluppo web, testing e molto altro ancora.

- Comunità attiva e supporto: Python ha una vasta e dinamica comunità di sviluppatori che contribuisce al suo sviluppo, fornisce supporto e condivide risorse.
- Versatilità: Python supporta diversi paradigmi di programmazione, come l'approccio object-oriented, l'imperativo e il funzionale.
- Portabilità: Python è un linguaggio portatile che può essere eseguito su diverse piattaforme, come Windows, macOS e Linux, senza la necessità di apportare modifiche significative al codice.
- Ampia adozione e popolarità: Python è diventato uno dei linguaggi di programmazione più popolari al mondo. È ampiamente utilizzato in vari settori, come lo sviluppo web, l'analisi dei dati, l'intelligenza artificiale, l'automazione dei processi e molto altro ancora.
- Facilità di integrazione: Python offre la possibilità di integrarsi facilmente con altri linguaggi di programmazione.
- Open-source e gratuito: Python è un linguaggio open-source, il che significa che è disponibile gratuitamente e il suo codice sorgente è accessibile a tutti.

2.2 MySQL

Per gestire i dati di interesse mi sono servito di un database MySQL, operando dall'applicazione PhpMyAdmin.

MySQL è un sistema di gestione di database relazionali (RDBMS) ampiamente utilizzato per memorizzare e organizzare grandi quantità di dati in modo strutturato. PhpMyAdmin, d'altra parte, è un'applicazione web che fornisce un'interfaccia utente intuitiva per gestire i database MySQL attraverso un browser web, che grazie alla sua interfaccia web user-friendly semplifica notevolmente le operazioni di gestione del database MySQL.

2.3 Scopus

Scopus è la più grande banca dati di abstract e citazioni di letteratura scientifica sottoposta a revisione paritaria: riviste scientifiche, libri e atti di conferenze.

Offre una panoramica completa della produzione di ricerca mondiale nei campi della scienza, della tecnologia, della medicina, delle scienze sociali e delle arti e delle discipline umanistiche.

Scopus mette a disposizione strumenti intelligenti per tracciare, analizzare e visualizzare la ricerca.

Capitolo Terzo

ESTRAZIONE DATI DA SCOPUS

3.1 Inserimento dati degli autori nel Database

Per prima cosa bisogna creare una tabella 'Autori' contenente i dati identificativi Scopus degli autori del Dipartimento dell'Ingegneria dell'Informazione dell'Università Politecnica delle Marche.

L'estrazione dati da Scopus si intende effettuarla a partire dal codice identificativo dell'autore, l'AU-ID, unico per ogni autore e che dobbiamo recuperare dal sito https://api.elsevier.com/content/search/author.

Il professor Domenico Potena mi ha fornito una tabella con le informazioni riguardanti i professori del Dipartimento dell'Ingegneria dell'Informazione dell'Università Politecnica delle Marche, in cui i campi sono:

- Cognome
- Nome
- SSD: Settore Scientifico Disciplinare
- ORCID: codice univoco che identifica i ricercatori a livello internazionale

Tabella 3.1: Esempio tabella autori iniziale

Cognome	Nome	SSD	ORCID
Zingaretti	Primo	Settore ING-INF/05…	0000-0002-5709-2159

Creato il database DII_da_Scopus, si crea la tabella che ci permette di immagazzinare i dati di interesse attraverso delle query SQL, la struttura della tabella è la seguente:

Tabella 3.2: Struttura tabella 'Autori'

Nome	Tipo
Nome	varchar(20)
Cognome	varchar(20)
SSD	varchar(100)
ORCID	varchar(20)
AU_ID	varchar(20)
Nome_scopus	varchar(20)
Cognome_scopus	varchar(20)
id	int(11)

Quindi vado per inserire sia i dati forniti dal prof (Nome, Cognome, SSD, ORCID) che i dati, che si trovano su Scopus ('AU_ID', 'nome_Scopus', 'cognome_Scopus') ho sviluppato due codici Python differenti:

- Il codice 1 riportato nell'appendice per inserire i dati da un file di testo contenente i dati della tabella 1 nel database sfrutta le seguenti operazioni:

1. Importa il modulo ‘mysql.connector’, che fornisce le funzionalità per connettersi a un database MySQL da Python e crea una connessione al database MySQL, quindi attraverso un cursore riesce ad effettuare delle query nel database.
2. Apre un file di testo chiamato "ORCID.txt" in modalità di lettura in cui sono presenti i dati della Tabella 1.
3. Attraverso un ciclo for, per ogni riga divide il testo in parti separate in base al separatore di tabulazione “\t” e assegna le parti alle variabili ‘cognome’, ‘nome’, ‘settore’ e ‘orcid’.
4. Prepara una query SQL per l’inserimento dei valori nel database mediante una sintassi parametrizzata e poi esegue la query con i valori ottenuti dalla riga corrente.
5. Una volta terminato il ciclo for avviene la conferma della transazione con ‘db.commit()’ per rendere permanenti le modifiche nel database.

In sostanza, il codice legge le righe da un file di testo chiamato “ORCID.txt” e inserisce i valori estratti in una tabella chiamata “Autori” all’interno del database MySQL “DII_da_Scopus”.

- Il codice 2 riportato nell'appendice invece, inserisce i dati nel database dal sito https://api.elsevier.com/content/search/author attraverso i seguenti passi:
    1. Importa i moduli necessari: 'requests' per effettuare richieste HTTP, `json` per lavorare con dati JSON e 'mysql.connector' per connettersi al database MySQL.
    2. Crea una connessione al database MySQL e ottiene un oggetto cursore che consente di eseguire le query sul database.
    3. Esegue una query per selezionare tutti i valori della colonna "ORCID" dalla tabella "autori" che non siano nulli e non abbiano un valore nella colonna "AU_ID".
    4. Recupera i risultati della query utilizzando il metodo 'fetchall()' e li memorizza nella variabile 'orcid'.
    5. Itera su ogni valore di 'orcid' attraverso un ciclo for.
    6. Aspetta 1 secondo tra ogni iterazione utilizzando 'time.sleep(1)' per rispettare le limitazioni di accesso all'API Elsevier.
    7. Effettua una richiesta GET all'API Elsevier utilizzando l'endpoint "/content/search/author" e passando il parametro "query" che contiene l'ORCID del valore corrente.

8. Ottiene i dati JSON dalla risposta utilizzando il metodo ‘json()’ del oggetto della richiesta.
9. Controlla se la chiave “search-results” è presente nei dati. Se presente, itera su ogni elemento nella chiave “entry”.
10. Verifica se sono presenti le chiavi “dc:identifier” e “preferred-name” in ogni elemento. Se presenti, estrae l’ID (rimuovendo i primi dieci caratteri), il nome e il cognome.
11. Esegue una query SQL con i valori ottenuti per l’aggiornamento della tabella “Autori” con l’ID, il nome e il cognome ottenuti dalla risposta dell’API. La query aggiorna la riga corrispondente all’ORCID corrente.
12. Terminato il ciclo effettua la conferma della transazione con ‘db.commit()’ per rendere permanenti le modifiche nel database.

In sostanza, il codice esegue un’iterazione sui valori ORCID presenti nel database. Per ciascun valore ORCID, effettua una richiesta all’API Elsevier per cercare l’autore corrispondente. Se i dati dell’autore sono presenti nella risposta, estrae l’ID, il nome e il cognome e li aggiorna nella tabella “Autori” del database MySQL.

Implementati i due codici descritti abbiamo popolato la tabella Autori correttamente e possiamo passare al prossimo step.

Tabella 3.3: Esempio record tabella 'Autori'

Nome	Primo
Cognome	Zingaretti
SSD	Settore ING-INF/05 - Sistemi di Elaborazione delle Informazioni
ORCID	0000-0002-5709-2159
AU_ID	6701855717
Nome_scopus	Primo
Cognome_scopus	Zingaretti
id	5

3.2 Inserimento dei dati dei paper nel Database

A questo punto dobbiamo creare una tabella che sia in grado di contenere i dati di interesse dei paper in Scopus. La struttura della tabella è stata impostata in base ai file JSON che vengono estratti. Di seguito un esempio:


```
{                    …
                                    "prism:url":
"https://api.elsevier.com/content/abstract/scopus_id/85058976029",
                                    "dc:identifier":
"SCOPUS_ID:85058976029",
                                    "eid": "2-s2.0-85058976029",
                                    "dc:title": "Defining a data-driven
maintenance policy: an application to an oil refinery plant",
                                    "dc:creator": "Antomarioni S.",
                                    "prism:publicationName":
"International Journal of Quality and Reliability Management",
                                    "prism:issn": "0265671X",
                                    "prism:volume": "36",
                                    "prism:issueIdentifier": "1",
                                    "prism:pageRange": "77-97",
```

```
                                "prism:coverDate": "2019-02-21",
                                "prism:coverDisplayDate":   "21   Feb
2019",
                                "prism:doi": "10.1108/IJQRM-01-2018-
0012",
                                "citedby-count": "25",
                                "affiliation":[
                        …
                                ],
                                "prism:aggregationType": "Journal",
                                "subtype": "ar",
                                "subtypeDescription": "Article",
                                "source-id": "144740",
                                "openaccess": "0",
                                "openaccessFlag": false
                            }
```


Quindi il professor Potena mi ha fornito i campi significativi con cui creare la tabella paper, che quindi avrà la seguente struttura:

Tabella 3.4: Struttura tabella 'paper'

Nome	**Tipo**	**Descrizione**
EID	varchar(25)	identificatore univoco assegnato a ciascun documento all'interno della piattaforma Scopus.
Url	varchar(255)	l'indirizzo web che punta alla pagina del paper su Scopus.
Title	varchar(255)	titolo del paper.
Pubblication_Name	varchar(255)	nome della pubblicazione o della rivista scientifica in cui il paper è stato pubblicato.
Issn	varchar(20)	codice univoco che identifica in modo univoco una pubblicazione

		periodica (rivista) a livello internazionale.
Isbn	varchar(30)	codice univoco che identifica in modo univoco un libro o un volume.
volume	varchar(255)	numero del volume in cui è stato pubblicato il paper all'interno della rivista scientifica.
PageRange	varchar(20)	L'intervallo di pagine in cui si trova il contenuto del paper all'interno della pubblicazione.
Cover_date	varchar(20)	La data di copertina o la data di pubblicazione del paper.
doi	varchar(40)	identificatore permanente univoco assegnato a un documento digitale.
citedby_count	int(11)	numero di volte in cui il paper è stato citato in altri documenti scientifici.
AggregationType	varchar(20)	tipo di aggregazione del paper all'interno della piattaforma Scopus.
subbytype	varchar(5)	sottotipo o la sottocategoria a cui appartiene il paper all'interno della sua categoria principale.
subtypeDescription	varchar(200)	descrizione del sottotipo o della sottocategoria a cui appartiene il paper.
Openaccessflag	tinyint(1)	Un indicatore che specifica se il paper è di accesso aperto.

Per collegare gli autori ai rispettivi paper si ha la necessità di creare una ulteriore tabella che contenga l'EID del paper e l'AU-ID dell'autore tramite due vincoli di integralità referenziale.

Tabella 3.5: Struttura tabella 'Autor_ship'

Nome	**Tipo**
AU_ID	varchar(20)
EID	varchar(25)

Una volta create le tabelle possiamo quindi andare a popolarle attraverso il seguente algoritmo che fa riferimento al codice 3 in appendice:

1. Importa i moduli necessari: 'requests', 'json' e 'mysql.connector'.
2. Si connette al database MySQL specificato e prepara un cursore per eseguire query sul database.
3. Esegue una query per selezionare tutti gli 'AU_ID' dalla tabella 'Autori' che non sono nulli e salva i risultati vengono salvati nella variabile 'autori'.
4. Definisce una funzione chiamata 'numero_chiamate' che prende un argomento ('arg') e restituisce il numero di chiamate necessarie per ottenere tutti i risultati dalla API di Scopus utilizzando l'AU-ID specificato poiché si possono ottenere massimo 25 pagine per chiamata.
5. Per ogni 'AU_ID' nella lista 'autori', viene eseguito il ciclo che segue:
   - Ottiene il numero di chiamate necessarie utilizzando la funzione 'numero_chiamate' e lo salva nella variabile 'x'.

- Esegue un altro ciclo per il numero di chiamate calcolato (‘x’).
- Per ogni iterazione, calcola il valore di ‘start’ in base all’indice dell’iterazione e al numero di risultati per pagina.
- Esegue una richiesta HTTP GET all’API di Scopus per ottenere i risultati corrispondenti all’AU-ID specificato e all’offset (‘start’) corrente.
- Analizza la risposta JSON e, se presenti risultati validi, estrae le informazioni riguardanti gli attributi della tabella ‘paper’.
- Esegue una query di inserimento nella tabella ‘Paper’ per memorizzare i dati estratti. Se una riga con lo stesso EID (identificativo) esiste già, viene eseguita un’operazione di aggiornamento (ON DUPLICATE KEY UPDATE) per aggiornare i valori corrispondenti.
- Esegue una query di inserimento nella tabella ‘Author_ship’ per memorizzare l’associazione tra l’EID e l’AU_ID.

7. Esegue il commit delle modifiche apportate al database.

In sintesi, il codice estrae i dati di ricerca da Scopus utilizzando gli AU-ID presenti nel database MySQL specificato e li memorizza nelle tabelle `Paper` e ‘Author_ship’, aggiornando le righe esistenti in caso di duplicati.

Tabella 3.6: Esempio record della tabella 'paper'

EID	2-s2.0-0000351904
Url	https://api.elsevier.com/content/abstract/scopus_i...
Title	Numerical aspects in time domain modelling of arbi...
Pubblication_Name	International Journal of Numerical Modelling: Elec...
Issn	08943370
Isbn	NULL
volume	12
PageRange	275-294
Cover_date	1999-01-01
doi	10.1002/(SICI)1099-1204(199907/08)12:4<2
citedby_count	11
AggregationType	Journal
subbytype	AR
subtypeDescription	Article
Openaccessflag	0

Tabella 3.7: Esempio record della tabella 'Autor_ship'

AU-ID	36055785900
EID	2-s2.0-0000351904

Capitolo Quarto

CONFRONTO DATI

In questo capitolo andremo a fare un confronto fra i dati estratti da Scopus e i dati del 'dbPubblicazioni' di IRIS. Per fare questo confronto costruiamo delle query che in modo da confrontare i paper in base al loro codice identificativo (EID).

4.1 Pubblicazioni con identificativo Scopus non trovati nell'estrazione di dati

Facciamo un controllo che le pubblicazioni con l'identificativo Scopus siano state effettivamente state estratte nella tabella paper attraverso la query:

```
'SELECT * FROM dbPubblicazioni.Pubblicazioni pu LEFT JOIN DII_da_Scopus.paper p ON p.EID = pu.dc_identifier_scopus WHERE p.EID IS NULL AND pu.dc_identifier IS NOT NULL'.
```

In teoria non dovrebbe dare alcun risultato in quanto non è possibile che esistano pubblicazioni con identificativo Scopus ma che non si trovano dentro Scopus, quindi ipotizziamo tre tipi di errore:

- L'identificativo è del paper inserito in Pubblicazioni è sbagliato.
- Il paper non è stato estratto da Scopus.

- L’autore del paper non fa parte del Dipartimento dell’Ingegneria dell’Informazione.

Per verificare quest’ultimo errore ho lavorato ad una query che, come risultato per ogni paper conti, il numero di autori facenti parte del Dipartimento:

```
SELECT pu.handle, pu.dc_identifier_scopus, SUM(IF(DII_da_Scopus.autori
.Cognome IS NULL, 0, 1)) FROM ((dbPubblicazioni.Pubblicazioni pu LEFT J
OIN DII_da_Scopus.paper p ON pu.dc_identifier_scopus = p.EID) JOIN dbP
ubblicazioni.Autori a ON a.id_pubblicazione = pu.handle) LEFT JOIN DII
_da_Scopus.autori ON (CONCAT(LOWER(DII_da_Scopus.autori.Nome), ’’, LOW
ER(DII_da_Scopus.autori.Cognome)) = a.nome OR CONCAT(LOWER(DII_da_Sco
pus.autori.Cognome), ’’, LOWER(DII_da_Scopus.autori.Nome)) = a.nome) W
HERE p.EID IS NULL AND pu.dc_identifier_scopus IS NOT NULL GROUP BY pu
.handle, pu.dc_identifier_scopus
```

Tabella 4.1: Esempio di risultato della precedente query SQL

handle	**dc_identifier_scopus**	**SUM(IF(DII_da_Scopus.autori.Cognome IS NULL, 0, 1))**
11566/106663	2-s2.0-84878142175	0
11566/124467	s2.0-84896344147	1
11566/236228	2-s2.0-85015351808	2

Quindi, i paper con somma di autori del Dipartimento uguali a zero non ci interessano, in quanto non appartengono a nessun autore del Dipartimento. Sfruttando la query SQL esposta sopra tramite un codice Python possiamo identificare i paper che hanno identificativo Scopus autori del Dipartimento, ma non si trovano sulla tabella ‘paper’ (contiene l’esportazione dei paper

degli autori del Dipartimento) e andare a richiederli a Scopus e, se il codice identificativo si riferisce a un paper veramente esistente, inserirli in una tabella chiamata ‘paper_non_trovati’ che ha la stessa struttura della tabella ‘paper’.

Il codice 4 riportato in appendice svolge le successive operazioni:

1. Vengono importate le librerie necessarie per il codice, inclusi mysql.connector, requests e json, che forniscono funzionalità per la connessione al database MySQL e le richieste HTTP.
2. Connessione ai database: Vengono stabiliti due collegamenti a database MySQL, ‘dbPubblicazioni’ e ‘DII_da_Scopus’ e vengono creati due oggetti cursor per eseguire le query sui rispettivi database.
3. Viene eseguita la query SQL sopra riportata e vengono riportati i dati in una variabile.
4. Iterazione sui risultati e recupero dei dati da Scopus: Vengono iterati i risultati ottenuti dalla query precedente. Se il paper appartiene a un autore del Dipartimento (cioè, se la somma degli autori è maggiore di 1), viene effettuata una richiesta HTTP a un’API di Scopus per recuperare ulteriori informazioni su un paper utilizzando l’ID Scopus (EID).

5. Estrazione dei dati dalla risposta JSON di Scopus: I dati richiesti vengono estratti dalla risposta JSON restituita dall'API di Scopus e assegnati a variabili corrispondenti.
6. Preparazione ed esecuzione di un'altra query SQL: Vengono preparati i valori ottenuti dal punto precedente e viene eseguita una query SQL utilizzando il cursore cursor2. La query inserisce i dati nel database 'DII_da_Scopus' nella tabella 'paper_non_trovati', aggiornando eventualmente i valori esistenti in caso di duplicati.
7. Commit dei cambiamenti nel database: Vengono confermati i cambiamenti apportati ai database tramite le operazioni di commit.

Dunque, prima di inserire i dati dei paper nella nuova tabella 'paper_non_trovati' è opportuno fare un confronto tra il titolo di quest'ultimi con i paper della tabella 'Pubblicazioni', dove il codice identificativo Scopus è uguale, in modo da verificare se il dato salvato in 'Pubblicazioni' era effettivamente corretto e qualora lo fosse inserire il paper nella tabella 'paper'.

Per effettuare queste operazioni ho implementato un codice Python (codice 5 nell'appendice) che esegue le successive operazioni:

1. Importazione delle librerie: Vengono importate le librerie necessarie per il codice, inclusa ‘mysql.connector’ per la connessione al database MySQL e ‘numpy’ per la manipolazione di array multidimensionali.
2. Definizione della funzione ‘get_cosine_similarity’: Questa funzione calcola la similarità del coseno tra due frasi. Prende in input due frasi (‘sentence1’ e ‘sentence2’) e una lista di stopwords (parole da ignorare). La funzione converte le frasi in minuscolo, crea un elenco di tutte le parole presenti in entrambe le frasi e quindi calcola i vettori di conteggio delle parole per entrambe le frasi. Infine, restituisce la similarità del coseno tra i due vettori.
3. Definizione della funzione ‘cos_sim_matrix’: Questa funzione calcola una matrice di similarità del coseno per un elenco di frasi. Prende in input una lista di frasi (‘sentences’) e una lista di stopwords. La funzione crea una matrice vuota delle dimensioni corrispondenti al numero di frasi. Successivamente, calcola la similarità del coseno tra tutte le coppie di frasi utilizzando la funzione ‘get_cosine_similarity’ e riempie la matrice di similarità di conseguenza. Infine, restituisce la matrice di similarità.

4. Connessione ai database: Vengono stabiliti due collegamenti a database MySQL, uno con il nome ‘dbPubblicazioni’ e l’altro con il nome ‘DII_da_Scopus’.
5. Creazione dei cursori: Vengono creati due oggetti ‘cursor’ per eseguire le query sui rispettivi database.
6. Esecuzione della query SQL: Viene eseguita una query SQL utilizzando il cursore ‘cursor1’ per selezionare i dati dalla tabella ‘paper_non_trovati’ nel database ‘DII_da_Scopus’ che abbiano un titolo. I risultati vengono salvati nella variabile ‘response_1’.
7. Iterazione sui risultati della query: Viene iterato su ogni riga dei risultati ottenuti dalla query precedente.
8. Calcolo della matrice di similarità del coseno: Viene utilizzata la funzione ‘cos_sim_matrix’ per calcolare una matrice di similarità del coseno tra la seconda e la ventunesima colonna della riga corrente dei risultati. Non vengono utilizzate stopwords nel calcolo.
9. Condizioni e inserimento nel database: Viene effettuato un controllo sulla ventunesima colonna della riga corrente. Se il valore non è nullo e la similarità del coseno tra le due frasi è superiore a 0.98, viene eseguita una query di inserimento nel database ‘DII_da_Scopus’ nella tabella ‘Paper’ utilizzando i valori della riga corrente.

10. Commit delle modifiche nel database: Viene confermata l'esecuzione delle operazioni di inserimento nel database tramite l'operazione di commit.

4.2 Controllo eventuali errori

Dopo aver completato il processo di recupero dei paper pubblicati dai professori del Dipartimento, possiamo procedere a fare un confronto tra gli attribuiti compatibili delle tabelle 'paper' nel Database 'DII_da_Scopus' e 'Pubblicazioni', in modo da segnalare eventuali errori.

Per salvare gli errori ho creato una nuova tabella nel Database 'DII_da_Scopus' con il nome di 'errore_base_eid'.

Tabella 4.2: Struttura della tabella 'errore_base_eid'

Nome	**Descrizione**
handle	È un identificatore persistente che può essere utilizzato per accedere in modo univoco a un paper all'interno del sistema Scopus.
eid	È un codice alfanumerico che identifica in modo univoco il documento all'interno del sistema Scopus.
tipo_errore	È il campo in cui è stato riscontrato l'errore
dato_paper	È il dato della tabella 'paper' del Database DII_da_Scopus in cui è stato riscontrato l'errore
dato_pubblicazioni	È il dato della tabella 'Pubblicazioni' del Database dbPubblicazioni presente in Iris in cui è stato riscontrato l'errore

Il controllo viene effettuato tramite un codice Python (codice 6 nell'appendice) che esegue che esegue le seguenti operazioni:

1. Import delle librerie: Vengono importate le librerie 'mysql.connector', 'cosine_distance' da 'nltk.cluster.util' e 'numpy' come 'np'.
2. Definizione delle funzioni: Sono definite due funzioni. get_cosine_similarity calcola la similarità del coseno tra due frasi, mentre cos_sim_matrix genera una matrice di similarità per un elenco di frasi.
3. Connessione ai database MySQL: Il codice stabilisce due connessioni separate a due database MySQL utilizzando 'mysql.connector'. Vengono creati oggetti 'cursor' per eseguire query sui database.
4. Esecuzione di una query sul database: Viene eseguita una query SQL che recupera i dati confrontabili dalle tabelle 'paper' e 'Pubblicazioni' database 'DII_da_Scopus' e 'dbPubblicazioni' (titolo, eid, doi…).

   I risultati della query vengono memorizzati nella variabile 'response_5'.
5. Iterazione sui risultati della query: Viene iterato su ogni riga di 'response_5' per analizzare i dati dei paper.
6. Calcolo della similarità coseno e inserimento dei dati nel database: Viene calcolata la similarità coseno tra le colonne "Title" e "dc_title"

di ciascuna riga utilizzando la funzione ‘cos_sim_matrix’. Se la similarità coseno è inferiore a 0.98, vengono inseriti i dati relativi all’errore nella tabella “errori_base_eid” del database. Vengono eseguite operazioni simili per altri campi come “doi”, “citedby_count”, “issn”, “isbn”, “subbytype”, “volume” e “PageRange”. In caso di rilevamento di un errore si esegue una query di inserimento nella tabella ‘errori_base_eid’, in cui verranno inseriti l’handle del paper dalla tabella ‘Pubblicazioni’, l’EID dalla tabella ‘paper’, la tipologia dell’errore e i due dati che hanno avuto riscontro negativo dal confronto effettuato.

7. Infine, le modifiche apportate al database vengono confermate utilizzando il metodo ‘commit()’ sulla connessione al database ‘db2’.

In questo modo abbiamo popolato la tabella ‘errori_base_eid’, in cui ora vi è la possibilità di controllare dato un EID se il paper presenta errori.

Tabella 4.3: Esempio record tabella ‘errori_base_eid’

handle	11566/100062
eid	2-s2.0-84878736343
tipo_errore	ISBN
dato_paper	true;9781467318112
dato_pubblicazioni	9781467318129

Capitolo Quinto

CONCLUSIONI E SVILUPPI FUTURI

Il presente lavoro di tesi ha affrontato il confronto e l'integrazione dei dati bibliografici provenienti da due database differenti, Scopus e un database interno denominato "dbPubblicazioni". L'obiettivo principale era identificare gli errori presenti nei dati e sviluppare soluzioni automatizzate utilizzando codici Python e query SQL.

Durante lo svolgimento della ricerca, sono state affrontate diverse sfide e complessità legate alla natura eterogenea dei dati e alle differenze strutturali tra i due database. Tuttavia, attraverso un approccio sistematico e l'implementazione di algoritmi e procedure specifiche, sono stati ottenuti risultati significativi.

Innanzitutto, mediante l'utilizzo di codici Python, sono stati sviluppati algoritmi per il calcolo della similarità coseno tra coppie di frasi e la creazione di una matrice di similarità. Questo ha permesso di confrontare i titoli dei paper presenti nei due database e individuare possibili discrepanze. La similarità coseno è stata un indicatore efficace per evidenziare le

differenze tra i titoli dei paper, fornendo un'indicazione sulla presenza di errori o discrepanze tra le fonti.

Inoltre, le query SQL sono state utilizzate per interrogare i due database e confrontare i valori dei campi chiave come DOI, ISSN, ISBN e altri identificatori univoci. Attraverso l'esecuzione di queste query, è stato possibile rilevare discrepanze ed errori, ad esempio nel caso in cui il DOI fosse diverso tra i due database o nel caso di ISSN mancanti o non corrispondenti.

Durante l'analisi dei dati, è emerso che la qualità e l'accuratezza delle informazioni presenti nei due database possono variare. Ciò può essere attribuito a diversi fattori, come errori umani nella digitazione dei dati, processi di acquisizione delle informazioni differenti o semplici errori di estrazione dei dati.

Complessivamente, l'implementazione di codici Python e query SQL ha consentito di individuare una serie di errori e discrepanze nei dati bibliografici tra Scopus e il database interno. Questi risultati sono stati di fondamentale importanza per evidenziare le criticità e le aree di miglioramento nei processi di integrazione e gestione dei dati bibliografici.

Dunque, la presente ricerca ha dimostrato l'importanza e la necessità di un confronto accurato e di una corretta integrazione dei dati bibliografici provenienti da diverse fonti. I codici Python e le query SQL sviluppati hanno fornito strumenti efficaci per identificare e correggere gli errori presenti nei dati, migliorando la qualità complessiva delle informazioni.

Tuttavia, è importante sottolineare che l'identificazione degli errori è solo il primo passo. Sulla base dei risultati ottenuti, si consiglia di adottare strategie e procedure specifiche per la correzione e l'integrazione dei dati, al fine di garantire l'affidabilità e l'accuratezza delle informazioni bibliografiche.

In prospettiva futura, si suggerisce di approfondire ulteriormente lo sviluppo di algoritmi di apprendimento automatico e di intelligenza artificiale per migliorare l'efficienza e l'automazione del processo di confronto e integrazione dei dati bibliografici.

In conclusione, il confronto e l'integrazione dei dati bibliografici rappresentano un ambito di ricerca in continua evoluzione, che richiede competenze tecniche avanzate e una profonda comprensione delle dinamiche dei dati. La presente tesi ha gettato le basi per ulteriori sviluppi e indagini, fornendo un quadro solido per il miglioramento dei processi di gestione dei dati bibliografici e l'ottimizzazione dell'informazione nel contesto accademico e scientifico.

Sitografia

- Wikimedia Foundation. (2023, May 10). MySQL. Wikipedia. https://it.wikipedia.org/wiki/MySQL
- Cos’è python: Linguaggio di Programmazione Python. Cos’è Python | Linguaggio di programmazione Python. (n.d.). https://www.python.it/about/
- Wikimedia Foundation. (n.d.). PhpMyAdmin. Wikipedia. https://it.wikipedia.org/wiki/PhpMyAdmin.
- Sensini, S. (2022, May 26). Misurare La Distanza tra due testi con python. avatar. https://www.theredcode.it/intelligenza-artificiale/misurare-la-distanza-tra-due-testi-con-python/
- Elsevier. (n.d.). Scopus. Elsevier. https://www.elsevier.com/solutions/scopus

Appendice

Codice 1: Inserimento dati Autore iniziali

```
import mysql.connector

db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="DII_da_Scopus"
)
cursor = db.cursor()
with open("ORCID.txt", "r") as f:
    for line in f:
        parts = line.strip().split("\t")
        cognome = parts[0]
        nome = parts[1]
        settore = parts[2]
        orcid = parts[3]
        sql = "INSERT INTO Autori(Nome, Cognome, SSD, Orcid) VALUES (%s,
%s, %s, %s)"
        value = (nome, cognome, settore, orcid)
        cursor.execute(sql, value)

db.commit()
```

Codice 2: Completamento inserimento dati Autori

```
import requests
import json
import mysql.connector
import time
```

```
db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="DII_da_Scopus"
)
cursor = db.cursor()
sql_1 = "SELECT ORCID FROM autori WHERE orcid IS NOT NULL AND AU_ID IS
NULL"
cursor.execute(sql_1)
orcid = cursor.fetchall()

for line in orcid:
    time.sleep(1)
    url                                                                =
requests.get("https://api.elsevier.com/content/search/author",
params={'query':
"ORCID({})".format(line[0]),'apiKey':"923da013762e48706b584503f0dc3b0a
"})
    data = url.json()
    if "search-results" in data:
        for item in data["search-results"]["entry"]:
            if "dc:identifier" in item and "preferred-name" in item:
                id = item["dc:identifier"][10:]
                nome = item["preferred-name"]["given-name"]
                cognome = item["preferred-name"]["surname"]
                sql_2 = "UPDATE Autori SET AU_ID = %s, nome_scopus =
%s, cognome_scopus = %s WHERE Orcid = %s"
                value = (id, nome, cognome, line[0])
                cursor.execute(sql_2, value)

db.commit()
```

Codice 3: Estrazioni dati paper da Scopus

```
import requests
import json
import mysql.connector
db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="DII_da_Scopus"
)
cursor = db.cursor()
sql_1 = "SELECT AU_ID FROM Autori WHERE AU_ID IS NOT NULL"
cursor.execute(sql_1)
autori = cursor.fetchall()

def numero_chiamate(arg):
    arg = "AU-ID({})".format(arg)
       url   =   requests.get("https://api.elsevier.com/content/search/
scopus",                                                params={'query':
arg,'apiKey':"923da013762e48706b584503f0dc3b0a","count":1})
    data = url.json()
    pagine = int(data["search-results"]["opensearch:totalResults"])
    chiamate = pagine // 25 + 1
    return chiamate

for line in autori:
    id = line[0]
    x = numero_chiamate(id)
    for i in range(x):
        start = i * 25
        url = requests.get("https://api.elsevier.com/content/search/
scopus",  params={'query': "AU-ID({})".format(id),'apiKey':
"923da013762e48706b584503f0dc3b0a","start":start})
```

```
        data = url.json()
          if "search-results" in data and "entry" in data["search-
results"]:
            for item in data["search-results"]["entry"]:
                if "prism:url" in item:
                    url = item["prism:url"]
                else: url = None
                if "dc:title" in item:
                    title = item["dc:title"]
                else: title = None
                if "prism:publicationName" in item:
                    publicationName = item["prism:publicationName"]
                else: publicationName = None
                if "prism:issn" in item:
                    issn = item["prism:issn"]
                else: issn = None
                if "prism:isbn" in item:
                    isbn = item["prism:isbn"]
                    if isinstance(isbn, list):
                      isbn_string = ";".join(["{};{}".format(i["@_fa"],
i["$"]) for i in isbn])
                      else: isbn_string = "{};{}".format(isbn["@_fa"],
isbn["$"])
                else: isbn_string = None
                if "prism:volume" in item:
                    volume = item["prism:volume"]
                else: volume = None
                if "prism:pageRange" in item:
                    pageRange = item["prism:pageRange"]
                else: pageRange = None
                if "prism:coverDate" in item:
                    coverDate = item["prism:coverDate"]
                else: coverDate = None
                if "prism:doi" in item:
```

```
                    doi = item["prism:doi"]
                else: doi = None
                if "citedby-count" in item:
                    citedby_count = item["citedby-count"]
                else: citedby_count = None
                if "prism:aggregationType" in item:
                    aggregationType = item["prism:aggregationType"]
                else: aggregationType = None
                if "subtype" in item:
                    subtype = item["subtype"]
                else: subtype = None
                if "subtypeDescription" in item:
                    subtypeDescription = item["subtypeDescription"]
                else: subtypeDescription = None
                if "openaccessFlag" in item:
                    openaccessFlag = item["openaccessFlag"]
                else: openaccessFlag = None
                      sql_2 = "INSERT INTO Paper (EID, Url, Title,
Pubblication_Name, Issn, Isbn, volume, PageRange, Cover_date, doi,
citedby_count, AggregationType, subbytype, subtypeDescription,
Openaccessflag) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s) ON DUPLICATE KEY UPDATE Url = %s, Title = %s,
Pubblication_Name = %s, Issn = %s, Isbn = %s, volume = %s, PageRange =
%s, Cover_date = %s, doi = %s, citedby_count = %s, AggregationType =
%s, subbytype = %s, subtypeDescription = %s, Openaccessflag = %s"
                 values_2 = (item["eid"], url, title, publicationName,
issn, isbn_string, volume, pageRange, coverDate, doi, citedby_count,
aggregationType, subtype, subtypeDescription, openaccessFlag, url,
title, publicationName, issn, isbn_string, volume, pageRange,
coverDate, doi, citedby_count, aggregationType, subtype,
subtypeDescription, openaccessFlag)
                cursor.execute(sql_2, values_2)
                 sql_3 = "INSERT IGNORE INTO Author_ship (EID, AU_ID)
VALUES (%s, %s)"
```

```
                values_3 = (item["eid"], id)
                cursor.execute(sql_3, values_3)

db.commit()
```

Codice 4: Estrazione paper non trovati da Scopus

```
import mysql.connector
import requests
import json

db1 = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="dbPubblicazioni"
)
cursor1 = db1.cursor()
db2 = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="DII_da_Scopus"
)
cursor2 = db2.cursor()
sql_1      =      ("SELECT      dbPubblicazioni.Pubblicazioni.handle,
dbPubblicazioni.Pubblicazioni.dc_identifier_scopus,
SUM(IF(DII_da_Scopus.autori.Cognome    IS    NULL,    0,    1))    FROM
((dbPubblicazioni.Pubblicazioni   LEFT   JOIN   DII_da_Scopus.paper   ON
dbPubblicazioni.Pubblicazioni.dc_identifier_scopus                    =
DII_da_Scopus.paper.EID)      JOIN      dbPubblicazioni.Autori      ON
dbPubblicazioni.Autori.id_pubblicazione                               =
dbPubblicazioni.Pubblicazioni.handle) LEFT JOIN DII_da_Scopus.autori ON
(CONCAT(LOWER(DII_da_Scopus.autori.Nome),             '             ',
```

```
LOWER(DII_da_Scopus.autori.Cognome)) = dbPubblicazioni.Autori.nome OR
CONCAT(LOWER(DII_da_Scopus.autori.Cognome), ' ', 
LOWER(DII_da_Scopus.autori.Nome)) = dbPubblicazioni.Autori.nome) WHERE
DII_da_Scopus.paper.EID IS NULL AND
dbPubblicazioni.Pubblicazioni.dc_identifier_scopus IS NOT NULL GROUP BY
dbPubblicazioni.Pubblicazioni.handle,
dbPubblicazioni.Pubblicazioni.dc_identifier_scopus")
cursor1.execute(sql_1)
response_1 = cursor1.fetchall()
i = 0
j = 0
count_1 = cursor1.rowcount
print("Numero di risultati:", count_1)
print("Questi sono gli eid corrispondenti ad autori non presenti in
DII_da_Scopus.autori")
for row in response_1:
    if row[2] == 0:
        i += 1

print("Numero di risultati:", i)
print("Questi sono gli EID corrispondenti a degli autori inseriti in
DII_da_Scopus.autori")
for row in response_1:
    if row[2] != 0:
          url = requests.get("https://api.elsevier.com/content/search/
scopus", params={'query': 'EID({})'.format(row[1]), 'apiKey':
'923da013762e48706b584503f0dc3b0a'})
        data = url.json()
        j += 1
        for item in data["search-results"]["entry"]:
            if "prism:url" in item:
                url = item["prism:url"]
            else:
                url = None
```

```
            if "dc:title" in item:
                title = item["dc:title"]
            else:
                title = None
            if "prism:publicationName" in item:
                publicationName = item["prism:publicationName"]
            else:
                publicationName = None
            if "prism:issn" in item:
                issn = item["prism:issn"]
            else:
                issn = None
            if "prism:isbn" in item:
                isbn = item["prism:isbn"]
                if isinstance(isbn, list):
                    isbn_string = ";".join(
                           ["{};{}".format(i["@_fa"], i["$"]) for i in
isbn])
                else:
                  isbn_string = "{};{}".format(isbn["@_fa"], isbn["$"])
            else:
                isbn_string = None
            if "prism:volume" in item:
                volume = item["prism:volume"]
            else:
                volume = None
            if "prism:pageRange" in item:
                pageRange = item["prism:pageRange"]
            else:
                pageRange = None
            if "prism:coverDate" in item:
                coverDate = item["prism:coverDate"]
            else:
                coverDate = None
```

```
            if "prism:doi" in item:
                doi = item["prism:doi"]
            else:
                doi = None
            if "citedby-count" in item:
                citedby_count = item["citedby-count"]
            else:
                citedby_count = None
            if "prism:aggregationType" in item:
                aggregationType = item["prism:aggregationType"]
            else:
                aggregationType = None
            if "subtype" in item:
                subtype = item["subtype"]
            else:
                subtype = None
            if "subtypeDescription" in item:
                subtypeDescription = item["subtypeDescription"]
            else:
                subtypeDescription = None
            if "openaccessFlag" in item:
                openaccessFlag = item["openaccessFlag"]
            else:
                openaccessFlag = None
             sql_2 = "INSERT INTO paper_non_trovati (EID, Url, Title,
Pubblication_Name, Issn, Isbn, volume, PageRange, Cover_date, doi,
citedby_count, AggregationType, subbytype, subtypeDescription,
Openaccessflag) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s) ON DUPLICATE KEY UPDATE Url = %s, Title = %s,
Pubblication_Name = %s, Issn = %s, Isbn = %s, volume = %s, PageRange =
%s, Cover_date = %s, doi = %s, citedby_count = %s, AggregationType =
%s, subbytype = %s, subtypeDescription = %s, Openaccessflag = %s"
```

```
            values_2 = (row[1], url, title, publicationName, issn,
isbn_string, volume, pageRange, coverDate, doi, citedby_count,
aggregationType, subtype, subtypeDescription,
                        openaccessFlag, url, title, publicationName,
issn, isbn_string, volume, pageRange, coverDate, doi, citedby_count,
aggregationType, subtype, subtypeDescription, openaccessFlag)
            cursor2.execute(sql_2, values_2)
print("Numero di risultati:", j)

db1.commit()
db2.commit()
```

Codice 5: Confronto titoli

```
import mysql.connector
from nltk.cluster.util import cosine_distance
import numpy as np

def get_cosine_similarity(sentence1, sentence2, stopwords=None):
    if stopwords is None:
        stopwords = []
    sentence1 = [w.lower() for w in sentence1]
    sentence2 = [w.lower() for w in sentence2]
    words_all = list(set(sentence1 + sentence2))
    array1 = [0] * len(words_all)
    array2 = [0] * len(words_all)
    for w in sentence1:
        if w in stopwords:
            continue
        array1[words_all.index(w)] += 1
    for w in sentence2:
        if w in stopwords:
            continue
        array2[words_all.index(w)] += 1
```

```
    return 1 - cosine_distance(array1, array2)

def cos_sim_matrix(sentences, stop_words):
    matrix = np.zeros((len(sentences), len(sentences)))
    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):
            if idx1 == idx2:  # ignore if both are same sentences
                continue
            matrix[idx1][idx2] = get_cosine_similarity(
                sentences[idx1], sentences[idx2], stop_words)
    return matrix

db1 = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="dbPubblicazioni"
)
cursor1 = db1.cursor()
db2 = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="DII_da_Scopus"
)
cursor2 = db2.cursor()

sql_1 = "SELECT * FROM DII_da_Scopus.paper_non_trovati p INNER JOIN
dbPubblicazioni.Pubblicazioni pu on p.EID=pu.dc_identifier_scopus WHERE
p.Title IS NOT NULL"
cursor1.execute(sql_1)
response_1 = cursor1.fetchall()

for row in response_1:
```

```
    matrix = cos_sim_matrix([row[2], row[20]], [])
    if row[20] != None:
        if matrix[0][1] > 0.98:
           sql_2="INSERT INTO Paper (EID, Url, Title, Pubblication_Name,
Issn, Isbn, volume, PageRange, Cover_date, doi, citedby_count,
AggregationType, subbytype, subtypeDescription, Openaccessflag) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
            values = (row[0], row[1], row[2], row[3], row[4], row[5],
row[6], row[7], row[8], row[9], row[10], row[11], row[12], row[13],
row[14])
           cursor2.execute(sql_2, values)

db2.commit()
```

Codice 6: Controllo errori

```
import mysql.connector
from nltk.cluster.util import cosine_distance
import numpy as np

def get_cosine_similarity(sentence1, sentence2, stopwords=None):
    if stopwords is None:
        stopwords = []

    sentence1 = [w.lower() for w in sentence1]
    sentence2 = [w.lower() for w in sentence2]

    words_all = list(set(sentence1 + sentence2))

    array1 = [0] * len(words_all)
    array2 = [0] * len(words_all)

    for w in sentence1:
        if w in stopwords:
```

```
            continue
        array1[words_all.index(w)] += 1

    for w in sentence2:
        if w in stopwords:
            continue
        array2[words_all.index(w)] += 1

    return 1 - cosine_distance(array1, array2)

def cos_sim_matrix(sentences, stop_words):
    matrix = np.zeros((len(sentences), len(sentences)))

    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):
            if idx1 == idx2:  # ignore if both are same sentences
                continue
            matrix[idx1][idx2] = get_cosine_similarity(
                sentences[idx1], sentences[idx2], stop_words)

    return matrix

db1 = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="dbPubblicazioni"
)

cursor1 = db1.cursor()

db2 = mysql.connector.connect(
    host="localhost",
    user="root",
```

```
    password="",
    database="DII_da_Scopus"
)
cursor2 = db2.cursor()

scopus_type = ["ar","ed","le","re","er","dp","no"]
scopus_aggregation = ["Journal","Trade Journal","Book","Book Series"]

sql = ("SELECT p.EID, pu.dc_identifier_scopus, p.Title, pu.dc_title,
p.doi, pu.dc_identifier_doi, p.citedby_count, pu.citation_scopus_total,
p.issn, pu.jissn, p.isbn, pu.isbn, p.subbytype, pu.scopus_type,
p.volume, pu.volume, p.PageRange, pu.dc_relation_firstpage,
pu.dc_relation_lastpage, pu.handle, p.AggregationType FROM
DII_da_Scopus.paper p INNER JOIN dbPubblicazioni.pubblicazioni pu ON
p.EID = pu.dc_identifier_scopus")
cursor2.execute(sql)
response_5 = cursor2.fetchall()
for row in response_5:

    matrix = cos_sim_matrix([row[2], row[3]], [])
    cosine = matrix[0][1]
    if matrix[0][1] < 0.98:
        sql_1 = "INSERT INTO errori_base_eid (handle, eid, tipo_errore,
dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s, %s)"
        errore_1 = "Title"+" "+str(cosine)
        value_1 = (row[19], row[0], errore_1, row[2], row[3])
        cursor2.execute(sql_1, value_1)
    if ((row[4] != None) and (row[5] != None)):
        if row[4].lower() != row[5].lower():
                sql_2 = "INSERT INTO errori_base_eid (handle, eid,
tipo_errore, dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s,
%s)"
            errore_2 = "DOI"
            value_2 = (row[19], row[0], errore_2, row[4], row[5])
```

```
            cursor2.execute(sql_2, value_2)
    elif (row[4] != None):
        sql_2 = "INSERT INTO errori_base_eid (handle, eid, tipo_errore,
dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s, %s)"
        errore_2 = "DOI (Mancante su IRIS)"
        value_2 = (row[19], row[0], errore_2, row[4], row[5])
        cursor2.execute(sql_2, value_2)
    if row[6] != row[7]:
        sql_3 = "INSERT INTO errori_base_eid (handle, eid, tipo_errore,
dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s, %s)"
        errore_3 = "citedby_count"
        value_3 = (row[19], row[0], errore_3, row[6], row[7])
        cursor2.execute(sql_3, value_3)
    if ((row[13] in scopus_type) and (row[20] in scopus_aggregation)):
        if (row[8] != None):
            if "{}-{}".format(row[8][0:4], row[8][4:8]) != row[9]:
                  sql_4 = "INSERT INTO errori_base_eid (handle, eid,
tipo_errore, dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s,
%s)"
                errore_4 = "ISSN (Scopus: "+row[20]+")"
                issn_1 = "{}-{}".format(row[8][0:4], row[8][4:8])
                value_4 = (row[19], row[0], errore_4, issn_1, row[9])
                cursor2.execute(sql_4, value_4)
        elif row[9] != "None":
                sql_5 = "INSERT INTO errori_base_eid (handle, eid,
tipo_errore, dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s,
%s)"
            errore_5 = "ISSN non presente su Scopus"
            value_5 = (row[19], row[0], errore_5, row[8], row[9])
            cursor2.execute(sql_5, value_5)
    if row[10] != None:
        if row[11] == None:
            res11 = ""
        else:
```

```
            res11 = row[11].replace('-','').replace(' ','').lower()
        res10 = row[10].replace('-','').replace(' ','').lower()
        res10arr = res10.split(";")
        if res11 not in res10arr:
                sql_6 = "INSERT INTO errori_base_eid (handle, eid,
tipo_errore, dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s,
%s)"
            errore_6 = "ISBN"
            isbn_1 = row[10]
            value_6 = (row[19], row[0], errore_6, isbn_1, row[11])
            cursor2.execute(sql_6, value_6)
    if row[12] != row[13]:
        sql_8 = "INSERT INTO errori_base_eid (handle, eid, tipo_errore,
dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s, %s)"
        errore_8 = "scopus_type"
        value_8 = (row[19], row[0], errore_8, row[12], row[13])
        cursor2.execute(sql_8, value_8)
    if row[15] != "None":
        if row[14] != row[15]:
                sql_9 = "INSERT INTO errori_base_eid (handle, eid,
tipo_errore, dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s,
%s)"
            errore_9 = "volume1"
            value_9 = (row[19], row[0], errore_9, row[14], row[15])
            cursor2.execute(sql_9, value_9)
    elif row[14] != None:
        sql_9 = "INSERT INTO errori_base_eid (handle, eid, tipo_errore,
dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s, %s)"
        errore_9 = "volume2"
        value_9 = (row[19], row[0], errore_9, row[14], None)
        cursor2.execute(sql_9, value_9)
    if row[17] is None or row[18] is None:
        if row[16] != row[17]:
```

```
                sql_10 = "INSERT INTO errori_base_eid (handle, eid,
tipo_errore, dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s,
%s)"
            errore_10 = "pageRange1"
            value_10 = (row[19], row[0], errore_10, row[16], None)
            cursor2.execute(sql_10, value_10)
    elif row[16] != "{}-{}".format(row[17], row[18]):
        date = "{}-{}".format(row[17], row[18])
       sql_11 = "INSERT INTO errori_base_eid (handle, eid, tipo_errore,
dato_paper, dato_pubblicazioni) values (%s, %s, %s, %s, %s)"
        errore_11 = "pageRange2"
        value_11 = (row[19], row[0], errore_11, row[16], date)
        cursor2.execute(sql_11, value_11)

db2.commit()
```