
UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e
dell'Automazione

# VALUTAZIONE SISTEMATICA DI ACCESSIBILITÀ, INTEROPERABILITÀ E RIUTILIZZABILITÀ DI DATASET RELATIVI AL COVID-19

# A SYSTEMATIC EVALUATION OF ACCESSIBILITY, INTEROPERABILITY AND REUSABILITY OF COVID-19 DATASETS

**Relatore:**
Prof. Emanuele Storti

**Candidato:**
Davide Traini

**Correlatore:**
Prof. Alex Mircoli

---

**A.A. 2020-2021**

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

# Indice

# Indice Immagini

# Indice Codici

# Capitolo 1

# Introduzione

Il biennio 2020-2021 è stato sconvolto dalla pandemia dovuta alla diffusione a livello globale del virus SARS-CoV-2 (covid-19). Tale virus è stato inizialmente scoperto a Wuhan, in Cina, il 31 Dicembre 2019. Da quel momento scienziati da tutto il mondo hanno iniziato a svolgere attività di ricerca, dando vita ad una gran mole di dati.

Il progetto prende spunto da un'analisi pubblicata in data 2 Dicembre 2020 sul Journal "Oxford Academic" ed intitolata "How do we share data in COVID-19 research? A systematic review of COVID-19 datasets in PubMed Central Articles"[1]. Tale studio è stato svolto facendo riferimento agli articoli presenti nel motore di ricerca PubMed[2]. In particolare sono stati estratti circa dodicimila articoli di ambito biomedico, ottenendo in totale 128 dataset, i quali sono poi stati esaminati manualmente e classificati in base a 3 fattori: contenuto dei dataset, accessibilità e citazioni.
I ricercatori hanno perciò concluso che PubMed[2] è un importantissimo strumento di ricerca, ma i dataset forniti sono troppo eterogenei e non rispettano a pieno la filosofia FAIR.

Lo scopo di questo elaborato consiste nel riproporre un'analoga analisi, svolta però sulla piattaforma "Zenodo"[3] e con metodologie diverse. In particolare l'obiettivo è stato quello di eliminare quasi totalmente l'intervento umano durante la fase di classificazione, cercando inoltre di standardizzare il più possibile le procedure di analisi dei dati.
Perciò la ricerca è stata svolta in modo da ricalcare il più possibile gli standard e le classificazioni proposte nella pubblicazione di riferimento, in modo da confrontare nelle fasi finali i risultati ottenuti, così da verificare se i dati siano omogenei o meno e nel caso cercare di capire se ciò sia dovuto al differente mezzo dal quale sono stati estratti i dataset o dall'evoluzione scientifica ad un anno di distanza dalla pubblicazione dei due progetti.

Per poter valutare accessibilità e riutilizzabilità dei dati si è fatto riferimento al principio FAIR[4] (Findability, Accessibility, Interoperability, and Reuse), che fornisce linee guida per rendere i dati facilmente accessibili sia ad utenti umani sia alle macchine.

Tale filosofia è fortemente legata ai concetti di linked data e semantic web, secondo cui i dati devono essere pubblicati in formati strutturati, devono essere ricchi di metadati e devono essere contrassegnati con un URI (stringa di caratteri che identifica in maniera univoca ed universale una risorsa), in modo tale da creare una vera e propria rete di informazioni in cui tutti i nodi (cioè i dati) sono collegati ad altri nodi logicamente affini.

Prendendo come riferimento la documentazione fornita nel sito web GO FAIR[5] e tenendo conto delle linee guida presenti nella pubblicazione "The FAIR Guiding Principles for scientific data management and stewardship" [4], possiamo definire i punti cardine di tale principio:

- Findability: il dataset deve essere facile da trovare nel web e deve essere ricco di metadati. I dati devono essere identificati da id univoci (URI) e i metadati devono riferirsi in modo chiaro al corrispondente id, in modo da evitare ambiguità e rendere più facile il lavoro di lettura per le macchine.
- Accessibility: dati e metadati devono essere accessibili secondo un protocollo di comunicazione standardizzato, il quale può essere aperto e gratuito garantendo l'accesso ai dati a qualsiasi utente, oppure può richiedere autenticazione ed autorizzazione. Inoltre deve essere possibile accedere ai metadati anche se i dati non sono più disponibili.
- Interoperability: i dati devono essere integrati con altri dati e devono essere rappresentati con un formato aperto e non proprietario. Inoltre le singole strutture all'interno dei dataset devono essere collegati ad altri dati tramite l'utilizzo di URI in modo da creare la struttura tipica dei linked data.
- Reuse: i metadati devono essere caratterizzati da attributi che descrivano al meglio il contenuto dei dataset. Dati e metadati devono essere rilasciati con una licenza accessibile e che consenta il loro riutilizzo (ad esempio CC0, CC-BY, CC-BY-NC etc) e deve essere sempre possibile risalire alla provenienza dei dataset.

**Struttura dell'elaborato**

Nel capitolo 2 verranno presentati gli strumenti utilizzati: tipologia di database nel quale immagazzinare i dati, linguaggio di programmazione utilizzato e relative librerie.

Il capitolo 3 è dedicato alla presentazione del workflow del progetto e alla spiegazione dei codici per estrarre i dati da Zenodo e classificare le pubblicazioni. In particolare tale capitolo è diviso in più sezioni:

- Nella sezione 3.1 verrà svolta un'analisi preliminare dei dati: consultando la documentazione fornita, si analizzerà il modo in cui Zenodo rappresenta le pubblicazioni ed i rispettivi metadati; inoltre si discuterà l'API messa a disposizione da Zenodo per estrarre le informazioni da noi richieste.
- Nella sezione 3.2, verrà creato il database in cui saranno inseriti i dati utili per le analisi.
- Nella sezione 3.3 si approfondirà l'utilizzo dell'API messa a disposizone da Zenodo e si discuterà il codice relativo all'estrazione dei dati e la loro memorizzazione nel DB.
- La sezione 3.4 riguarderà strettamente la classificazione dei dati: i documenti verranno preprocessati e tramite l'utilizzo di un'API saranno divisi in base a degli ambiti da noi definiti.

Nel capitolo 4 verranno riportate tutte le analisi e le statistiche svolte, ponendo massima attenzione sui concetti di accessibilità, riutilizzabilità ed interoperabilità.

Il capitolo 5 sarà dedicato ad una discussione relativa ai risultati ottenuti nel capitolo precedente e saranno presentate delle proposte per espandere il progetto.

# Capitolo 2

# Strumenti utilizzati

I codici discussi di seguito sono tutti disponibili al seguente link GitHub: `https://github.com/dadezzzzz/Covid-19-Tirocinio`

Come già detto il sito di riferimento per il progetto è Zenodo: un archivio open access gestito da CERN per OpenAIRE (infrastruttura finanziata dall'UE per la promozione e la racolta di dati aperti).
Tale scelta è legata al fatto che durante il periodo della pandemia è stato uno dei repository più utilizzati per la pubblicazione di dataset ed articoli relativi al covid-19.
La popolarità di Zenodo è dovuta principalmente alle funzionalità che esso mette a disposizione dei ricercatori; infatti è possibile caricare dataset di grandi dimensioni (fino a 50GB) ai quali vengono automaticamente assegnati degli identificativi (DOI) nel caso in cui ne siano sprovvisti; è molto facile interfacciare Zenodo con GitHub e non è necessario avere alcun ORCID er poter caricare dataset, ciò rende la piattaforma molto flessibile.

Tutti gli script sono implementati in Python, poiché è un linguaggio che rende molto facile l'utilizzo delle librerie per il machine learning e per la gestione dei dati. In particolare è stato utilizzato Jupyter Notebook[6]: un'applicazione Web open source che permette di creare e condividere documenti testuali interattivi fornendo un'interfaccia grafica accattivante con la quale è più facile visualizzare e gestire i dati.

Per la memorizzazione dei dati si è optato per un database relazionale, in particolare è stato utilizzato il tool phpMyAdmin[1], il quale fornisce un'interfaccia web per la gestione di database MariaDB e MySQL.

[1] La documentazione è disponibile al link https://github.com/phpmyadmin/phpmyadmin

Inoltre sono stati utilizzati seguenti package e librerie Python:

- requests[7]: libreria che consente una facile implementazione di richieste HTTP.
- pymySQL: libreria che consente di interfacciarsi con DB SQL.
- json: libreria per trasformare dictionary in JSONObject e viceversa.
- re[8]: libreria per l'implementazione di regex (regular expression).
- BeautifulSoup[9]: libreria che trasforma il documento HTML in un albero di analisi e consente il parsing con poche istruzioni.
- nltk[10]: package che fornisce diverse librerie e modelli per l'analisi di testi.
- fasttext[11]: modulo Python che mette a disposizione funzioni per l'addestramento supervisionato e non supervisionato di modelli per l'elaborazione di testi.
- numpy[12]: libreria open source che consente la possibilità di svolgere calcoli in modo efficiente con vettori e matrici di grandi dimensioni.
- pandas[13]:libreria open source che consente una facile gestione dei dati grazie all'utilizzo di una struttura dati detta Dataframe.
- matplotlib[14]: libreria per la generazione di grafici.
- sklearn[15]: libreria per l'apprendimento automatico.

Oltre a ciò è stato scaricato il modello en_core_web_trf pre-addestrato e fornito da Spacy[16]. Tale modello verrà utilizzato nella sezione 3.4.2 poiché consente l'identificazione all'interno di un testo di entità facenti parte del seguente gruppo: CARDINAL, DATE, EVENT, FAC, GPE, LANGUAGE, LAW, LOC, MONEY,NORP, ORDINAL, ORG, PERCENT, PERSON,PRODUCT, QUANTITY, TIME.

L'uso specifico dei metodi e delle funzioni funzioni delle librerie sarà approfondito nelle sezioni in cui esse saranno utilizzate.

# Capitolo 3

# Metodologia

Questo capitolo è diviso in 4 sottosezioni:

- Nella sezione 3.1 sarà svolta un'analisi della documentazione fornita da Zenodo, in modo tale da capire come i dataset sono rappresentati all'interno della piattaforma ed il metodo più efficiente per estrarne dati e metadati.
- Alla luce delle conoscenze ottenute nella sezione precedente, nella sezione 3.2 verrà creato il DB, all'interno del quale saranno inseriti i dati estratti dal repository Zenodo.
- La sezione 3.3 sarà dedicata alla discussione dello script utilizzato per estrarre i dati ed inserirli nel DB precedentemente creato.
- Nella sezione 3.4 verranno utilizzati alcuni modelli e librerie per il machine learning in modo da classificare i dataset presenti nel DB in base all'ambito trattato.

## 3.1 Analisi della documentazione

Come già detto, l'obiettivo del progetto consiste nel verificare se i dataset relativi al Covid-19, presenti nella piattaforma Zenodo, rispettino o meno il principio FAIR[4]. A tale scopo devono essere analizzati i metadati delle pubblicazioni, ma prima di far ciò è necessario capire come Zenodo gestisca tali metadati e quale sia il metodo più efficente per estrarli.
Consultata la documentazione fornita[1], si è visto che esistono diversi formati di rappresentazione di dati e metadati: BibTeX, CSL, DataCite, Dublin Core, DCAT, JSON, JSON-LD, GeoJSON, MARCXML.
Inoltre nella documentazione è presente una sezione relativa ad un'API che consente di accedere a dati e metadati di un insieme di pubblicazioni senza dover applicare algoritmi di Web-Scraping. I risultati restituiti da tale API sono codificati sotto forma di JSON. Alla luce di ciò si è deciso di utilizzare l'API per velocizzare il processo di estrazione dei metadati, il suo utilizzo verrà approfondito nella sezione 3.3.1.

Osservando i campi standard presenti del JSON che descrive una singola pubblicazione, sono stati ricavati alcuni attributi fondamentali che saranno poi utili per le analisi successive:

- DOI (identificatore univoco della pubblicazione)
- title (titolo)
- notes (note aggiuntive)
- description (descrizione)
- method (metodo con cui sono stati raccolti i dati)
- license (licenza)
- access_right (possibili valori: Open, Restricted, Closed, Embargoed)
- access_right_category (possibili valori: Danger, Success, Warning)
- access_conditions (note relative all'accesso)
- created (data creazione della versione corrente del dataset)
- updated (data ultima modifica della versione corrente del dataset)
- volume (dimensione in byte del dataset)
- keywords (insieme di keyword della pubblicazione)
- ricercatori (insieme di ricercatori che hanno lavorato alla pubblicazione), caratterizzati da:

[1] Accessibile nel sito web di riferimento https://developers.Zenodo.org/

  - orcid (identificatore univoco del ricercatore, solo i ricercatori registrati ne hanno uno)
  - name (stringa costituita da nome e cognome del ricercatore)
  - affiliation (università o ente di appartenenza del ricercatore)

- file (insieme di file che costituiscono la pubblicazione), caratterizzati da:
  - link (link del file)
  - key_f (nome del file)
  - type (formato del file)
  - size (dimensione in byte del file)

Un esempio di JSON relativo ad una singola pubblicazione è il seguente:

Listing 3.1: JSON pubblicazione


```
{
  "files": [
    {
      "links": {
        "self": "https://Zenodo.org/api/files/e6bb8
           ae0-bdb4-4f61-8d0d-c4623e3cc90b/SC2
           _Index_Case.zip"
      },
      "checksum": "md5:564814f2817407978e34272a7d2be
         782",
      "bucket": "e6bb8ae0-bdb4-4f61-8d0d-c4623e3cc90
         b",
      "key": "SC2_Index_Case.zip",
      "type": "zip",
      "size": 2987618528
    }
  ],
  "owners": [
    90070
  ],
  "DOI": "10.5061/dryad.4f4qrfjbm",
  "stats": {
    "version_unique_downloads": 2.0,
    "unique_views": 3.0,
```

```
    "views": 3.0,
    "version_views": 3.0,
    "unique_downloads": 2.0,
    "version_unique_views": 3.0,
    "volume": 5975237056.0,
    "version_downloads": 2.0,
    "downloads": 2.0,
    "version_volume": 5975237056.0
  },
  "links": {
    "DOI": "https://DOI.org/10.5061/dryad.4f4qrfjbm"
      ,
    "latest_HTML": "https://Zenodo.org/record/459236
      3",
    "bucket": "https://Zenodo.org/api/files/e6bb8ae0
      -bdb4-4f61-8d0d-c4623e3cc90b",
    "badge": "https://Zenodo.org/badge/DOI/10.5061/
      dryad.4f4qrfjbm.svg",
    "HTML": "https://Zenodo.org/record/4592363",
    "latest": "https://Zenodo.org/api/records/459236
      3"
  },
  "created": "2021-03-09T19:01:11.242649+00:00",
  "updated": "2021-03-10T12:27:21.469547+00:00",
  "conceptrecid": "4592362",
  "revision": 2,
  "id": 4592363,
  "metadata": {
    "access_right_category": "success",
    "DOI": "10.5061/dryad.4f4qrfjbm",
    "description": "<p>Understanding when SARS-CoV-2
       emerged is critical to evaluating our
      current approach to monitoring novel zoonotic
       pathogens and understanding the failure of
      early containment and mitigation efforts for
      COVID-19. We employed a coalescent framework
      to combine retrospective molecular clock
      inference with forward epidemiological
      simulations to determine how long SARS-CoV-2
      could have circulated prior to the time of
      the most recent common ancestor. Our results
```

```
    define the period between mid-October and mid
    -November 2019 as the plausible interval when
     the first case of SARS-CoV-2 emerged in
    Hubei province.</p>",
  "license": {
    "id": "CC0-1.0"
  },
  "title": "Timing the SARS-CoV-2 index case in
    Hubei Province",
  "notes": "<p>BEAST data is available for use,
    along with the aggregated FAVITES results.
    The input files for the final results in the
    manuscript are available and they can be used
     with the Python script provided. Please see
    the README.txt file and manuscript for
    further details.\u00a0</p>",
  "relations": {
    "version": [
      {
        "count": 1,
        "index": 0,
        "parent": {
          "pid_type": "recid",
          "pid_value": "4592362"
        },
        "is_last": true,
        "last_child": {
          "pid_type": "recid",
          "pid_value": "4592363"
        }
      }
    ]
  },
  "access_right": "open",
  "communities": [
    {
      "id": "dryad"
    }
  ],
  "publication_date": "2021-03-09",
  "creators": [
```

```
      {
        "orcid": "0000-0003-0977-2886",
        "affiliation": "University of California,
           San Diego",
        "name": "Pekar, Jonathan"
      },
      {
        "affiliation": "University of California,
           San Diego",
        "name": "Wertheim, Joel"
      }
    ],
    "method": "<p>The dataset was collected by using
       BEAST and FAVITES to generate phylogenetic
      analyses and epidemic simulations,
      respectively. Both were manually processed to
       determine the tMRCA and stable coalescence,
      respectively. Refer to the manuscript for
      further details.\u00a0</p>",
    "resource_type": {
      "type": "dataset",
      "title": "Dataset"
    },
    "related_identifiers": [
      {
        "scheme": "DOI",
        "identifier": "10.1101/2020.11.20.392126",
        "relation": "cites"
      }
    ]
  }
}
```


Non tutti i campi del JSON sono obbligatori, alcuni possono anche non essere presenti, altri invece possono essere mutuamente esclusivi (se "access_right" risulta avere valore "open" allora non ci sarà alcun valore dell'attributo "access_conditions" poiché il file è aperto sotto qualsiasi condizione).

## 3.2 Creazione del DB

### 3.2.1 Schema E-R

Dopo aver analizzato la struttura del JSON nella sezione 3.1 a pagina 7, è stato creato lo schema E-R riportato in figura 3.1

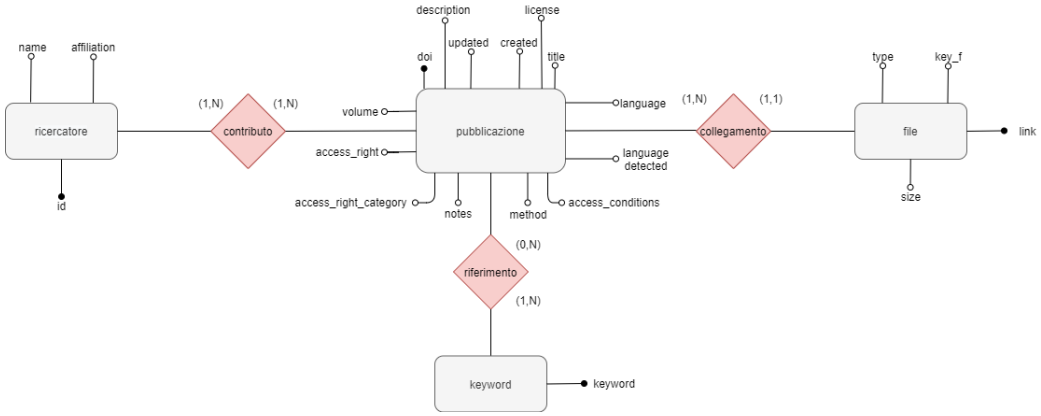


Figure 3.1: Schema ER

La cardinalità delle relationship deriva dalla struttura dei dati:

- Contributo:
  - Un RICERCATORE può contribuire ad 1 o più PUBBLICAZIONI
  - Ad una PUBBLICAZIONE può contribuire 1 RICERCATORE o più

- Riferimento:
  - Una KEYWORD può essere riferita ad 1 o più PUBBLICAZIONI
  - Ad una PUBBLICAZIONE possono essere riferite 0 o più KEYWORD

- Collegamento:
  - Un FILE può essere collegato ad 1 unica PUBBLICAZIONE
  - Ad una PUBBLICAZIONE possono essere collegati 1 o più FILE

### 3.2.2 Traduzione in modello relazionale

Tenendo conto della cardinalità delle relationship, le relazioni ottenute in seguito alla traduzione sono le seguenti:

- pubblicazione(DOI, title, language, notes, description, method, license, access_right, access_right_category, access_conditions, created, updated, volume)
- file(link, key_f, type, size, pubblicazione)
- keyword(keyword)
- riferimento(pubblicazione, keyword)
- ricercatore(id, name, affiliation)
- caricamento(pubblicazione, ricercatore)

Esistono poi dei vincoli di integrità referenziale:

- tra "pubblicazione" in FILE e la chiave di PUBBLICAZIONE
- tra "pubblicazione" in CARICAMENTO e la chiave di PUBBLICAZIONE
- tra "ricercatore" in CARICAMENTO e la chiave di RICERCATORE
- tra "pubblicazione" in RIFERIMENTO e la chiave di PUBBLICAZIONE
- tra "keyword" in RIFERIMENTO e la chiave di KEYWORD

A partire da queste relazioni, le tabelle sono state codificate come riportato nel repository GitHub nel file *"DB_create.SQL"*.
Ci sono però 2 attributi aggiuntivi nella tabella delle pubblicazioni:

- "language_detected": verrà utilizzato nella sezione 3.4.2 a pagina 21 quando per ogni documento verrà estratta la lingua
- "ambito_detected": verrà utilizzato nella sezione 3.4.3 a pagina 28 quando le pubblicazioni verranno classificate in base agli ambiti.

## 3.3 Estrazione dati

Una volta creato il DB in cui inserire i metadati e dopo aver analizzato la documentazione per valutare quali campi del JSON saranno utili per le analisi, è possibile approfondire l'utilizzo dell'API e discutere lo script che verrà utilizzato per chiamare l'API, estrarre i dati ed inserirli nel DB.

### 3.3.1 Descrizione API

L'API di Zenodo può essere utilizzata sia per depositare dati, che per estrarli, ciò può essere svolto direttamente all'interno di uno script Python utilizzando il package "requests"[7].
Nel caso in cui si vogliano estrarre dati, l'API consente la definizione di diversi parametri per specializzare la richiesta, per il nostro scopo sono stati utilizzati i seguenti:

- q: query di ricerca, utilizza la sintassi Elasticsearch[2].
- page: numero della pagina da restituire (default 1).
- size: numero di documenti per la pagina restituita (default 10).
- type: tipologia di pubblicazioni da restituire (di default le restituisce tutte, nel nostro caso saranno richiesti solamente dataset).
- access_token: codice necessario per poter utilizzare l'API, ottenuto previa registrazione nel sito di Zenodo.

L'API restituisce alcune statistiche sui documenti totali estratti considerando solo la query "q", più le informazioni (contenute nell'attributo hits) relative ai documenti che rispettano anche gli altri parametri inseriti nella richiesta. La struttura è la seguente:

- access_right: numero di documenti per ogni tipologia di accesso (open, closed etc).
- file_type: numero di documenti per ogni formato (pdf, jpg etc).
- keywords: numero di documenti per ogni keyword.
- type: numero di documenti per ogni categoria (dataset, image etc).

[2] Server di ricerca che consente di concatenare stringhe con operatori logici. La documentazione è disponibile al link https://www.elastic.co/

  - subtype: numero di documenti per ogni sotto-categoria (article, report etc).

- hits: vettore contenente i singoli documenti estratti: ogni elemento di questo vettore è un oggetto JSON del tipo rappresentato nel listato 3.1

Di seguito è riportata la struttura del JSON in cui i campi sono però privi di qualsiasi valore:

Listing 3.2: API JSON response

```
{
  "aggregations": {
    "access_right": {
      "buckets": [
        {
          "doc_count": ,
          "key":
        }
      ]
    },
    "file_type": {
      "buckets": [
        {
          "doc_count": ,
          "key":
        }
      ]
    },
    "keywords": {
      "buckets": [
        {
          "doc_count": ,
          "key":
        }
      ]
    },
    "type": {
      "buckets": [
        {
          "doc_count": ,
```

```
            "key": ,
            "subtype": {
              "buckets": [
                {
                  "doc_count": ,
                  "key":
                }
              ]
            }
          }
        ]
      }
    },
    "hits": {
      "hits": [

      ]
    }
}
```

### 3.3.2 Codice Python estrazione

Lo script *"estrazione_dati_API.py"* presente nel repository GitHub, ha come obiettivo l'estrazione dei dati, con formato JSON, tramite l'utilizzo dell'API ed il loro inserimento nel DB.

Come già detto in precedenza, per estrarre i documenti è necessario specificare una query con sintassi Elasticsearch. Nel nostro caso si è deciso di utilizzare la query :

```
'q': '("covid") OR ("covid-19") OR ("sars-cov-2") OR
   ("sars cov 2")'
```

In questo modo vengono estratti i documenti in cui è citata almeno una delle locuzioni definite tra "...".
La ricerca dei documenti svolta da Zenodo può risultare però problematica, poiché se la parola "covid" compare anche solo una volta nella descrizione di una pubblicazione, essa viene comunque restituita nella ricerca: vengono perciò ottenuti dei documenti "spuri" (cioè documenti che non riguardano direttamente l'argomento Covid-19, ma nei quali esso è citato).
Per arginare questo problema successivamente sarà definito un modello creato tramite API, il quale ci consentirà di classificare i documenti attraverso più ambiti da noi scelti, perciò le pubblicazioni che non riguardano i suddetti ambiti verranno eliminate.

L'API consente la definizione di 2 elementi fondamentali: il numero della pagina da restituire e quanti documenti inserire nella suddetta pagina, perciò per richiedere tutti i documenti che rispettano la query "q" è necessario un ciclo while che iteri chiedendo pagine all'API finché essa non restituisca un messaggio di errore con codice 400 (documenti terminati). Ad esempio, supponiamo di aver deciso di voler 100 documenti per ogni pagina, se avessimo 800 documenti che rispettano la query "q" allora ciò vorrebbe dire che sarà necessario chiedere all'API la pagina 1, la 2, la 3 etc fino al termine dei documenti (pagina 9), cioè fino a che l'API non restituirà errore 400.
Inoltre si è scelto di utilizzare un numero ridotto di pagine contenenti un gran numero di documenti, poiché in questo modo si impiega un minor tempo, infatti se si utilizzano poche pagine, ma molto grandi, si riduce il numero di richieste all'API.

Alcuni campi testuali sono rappresentati come HTML, perciò è stato utilizzato il package BeautifulSoup[9], il quale trasforma il documento

HTML in un albero di analisi e consente il parsing con poche istruzioni.

Durante l'estrazione dei dati dal JSON vengono utilizzati dei "try...except" in modo che, se il campo non è presente nel JSON, viene estratta una stringa vuota. Inoltre nei campi di tipo stringa sono eliminati i carriage return (\n) e gli apici ("), per evitare problemi durante le elaborazioni svolte successivamente.

In alcune pubblicazioni le keyword sono rappresentate come elementi di un JSONArray, perciò è bastato iterare gli elementi ed inserirli nel DB; in altre le keyword sono state rappresentate all'interno della stessa stringa separate da "," o ";", perciò è stato necessario dividere la stringa in token ed aggiungere ogni elemento al DB.

Per identificare i ricercatori all'interno del DB, possono essere utilizzati gli ORCID, ma non tutti gli iscritti alla piattaforma Zenodo ne sono provvisti, perciò, qualora il ricercatore non ne abbia uno, si è deciso di utilizzare una funzione di hash che prende in ingresso le stringhe che rappresentano nome, cognome e affiliazione del ricercatore e restituisce un codice numerico univoco di 20 cifre che verrà poi utilizzato come chiave primaria al posto dell'ORCID.

## 3.4 Classificazione dei documenti

### 3.4.1 Obiettivi

L'obiettivo di questa sezione è quello di classificare i documenti in base all'ambito di appartenenza. Il processo è stato diviso in 2 fasi:

- preprocessing dei documenti ed estrazione delle keywords.
- utilizzo di un API per la classificazione automatica dei documenti.

Durante la prima fase sono state svolte due operazioni:

- estrazione della lingua (sottosezione 3.4.2): è necessario separare i documenti in funzione della lingua con cui sono stati scritti i relativi titoli e descrizioni, questo consente di dividere i dataset più facilmente leggibili, cioè quelli in inglese, da quelli in altre lingue. Inoltre tale divisione è necessaria poiché successivamente verrà utilizzato un lemmatizer, cioè un algoritmo addestrato per riconoscere la parola passata in input e se essa è presente nel suo dizionario ne restituisce la radice. Ad esempio, supponendo di avere un lemmatizer addestrato nel lemmizzare parole inglesi, nel caso in cui gli venisse passata la parola *played* o la parola *plays* esso restituirebbe il lemma *play*, ma non avrebbe alcun effetto se gli venisse passata la parola *giocare*.
  Per questo motivo per ogni pubblicazione registrata nel DB sono stati concatenati i valori presenti nel campo `title` e `description`; la stringa ottenuta è stata sottoposta ad un pre-processing durante il quale sono state eliminate parole non significative e simboli. Sono state tolte anche le Named Entities, estratte tramite un modello NER (Named Entity Recognition: modello tramite il quale vengono riconosciute nel testo parole facenti parte nel seguente insieme: CARDINAL, DATE, EVENT, FAC, GPE,LANGUAGE, LAW, LOC, MONEY,NORP, ORDINAL, ORG, PERCENT,PERSON,PRODUCT, QUANTITY, TIME). Successivamente si è utilizzato un language detector per estrarre la lingua del documento, il valore estratto è poi stato inserito nella colonna `language_detected` del DB.
- calcolo TF-IDF (sottosezione 3.4.2): dopo aver estratto la lingua di tutti i documenti si è visto che la maggior parte di essi ha `titolo` e `descrizione` in inglese, motivo per cui si è deciso di procedere con la classificazione dei documenti in lingua inglese escludendo gli altri poiché di numero esiguo.
  Per ogni pubblicazione con campo `language_detected = en`, il testo

ottenuto dalla concatenazione di `titolo` e `descrizione` è stato pre-processato eliminando parole contenenti caratteri non inglesi, simboli e Named Entities. La stringa risultante è stata ridotta ad una lista di parole, ad ognuna delle quali si è applicato un lemmatizer.
Dopo aver applicato tale algoritmo a tutti i documenti, ottenendo un dizionario con formato `{"DOI_documento":"lista di lemmi"}`, è stato calcolato il TF-IDF per ogni lemma. Il TF-IDF è un indicatore che rappresenta quanto un termine in un documento sia significativo rispetto all'intero corpus di documenti, tale indice verrà discusso in maniera accurata nella sezione 3.4.2.
Poiché il TF-IDF di una parola rappresenta quanto essa sia significativa per un dato documento, sono stati estratti per ogni documento i 20 termini con TF-IDF più alto e sono stati inseriti in un file CSV, verranno poi utilizzati come parole chiave durante la classificazione.

Durante la seconda fase, grazie all'API Dandelion[17], si è creato il modello con cui classificare i dataset; tramite il quale sono stai classificati i documenti sfruttando le keyword precedentemente memorizzate nel file CSV.

Di seguito è rappresentato il workflow che rappresenta i passi necessari per classificare un singolo documento:

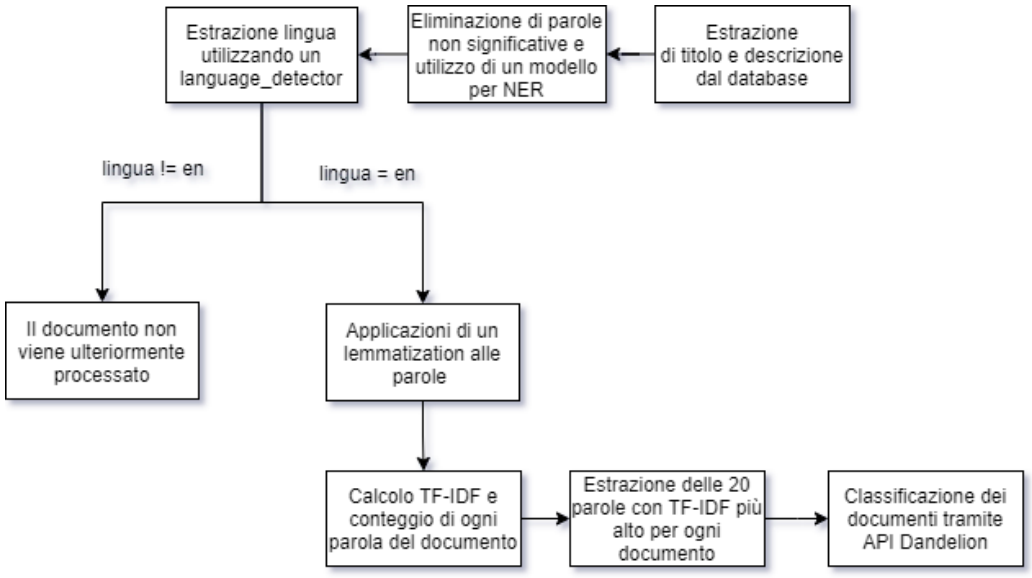


Figure 3.2: Workflow classificazione

### 3.4.2 Pre-processing ed estrazione parole chiave

#### Language Detection

Per ogni pubblicazione si è pensato di concatenare tra loro i campi `title`, `description`, `note` e `method`, ed estrarre le keywords dal testo così ottenuto. Procedendo in questo modo però si è visto che la precisione dell'API che verrà discussa nella sezione 3.4.3 a pagina 28 è molto minore rispetto alla precisione ottenuta concatenando tra loro solo titolo e descrizione.

Prima di classificare i documenti è stato applicato un language_detector, in modo da dividere i documenti in inglese (che saranno processati) da quelli in altre lingue. Per valutare quale package utilizzare si è fatto riferimento ad un benchmark intitolato "Benchmarking Language Detection for NLP"[18], nel quale si suggerisce di utilizzare fasttext[11] poiché consente di avere elevate precisioni e tempi di esecuzione bassi anche con testi molto estesi.
Prima di utilizzare fasttext, per ogni pubblicazione il testo è stato preparato tramite:

- regular expression per eliminare link, email, segni di punteggiatura, simboli, parole contenenti lettere ripetute più di 3 volte di seguito o parole contenenti più di 4 numeri uno di seguito all'altro. Ciò è stato svolto utilizzando il package re[8]. In questo modo sono state eliminate tutte quelle parole che l'algoritmo per il rilevamento della lingua sicuramente non riesce a riconoscere e che quindi peggiorerebbero i risultati.
- NER (Named Entity Recognitions) per l'estrazione di nomi di persone, organizzazioni, date etc.
  Per identificare le NER è stato utilizzato un modello pre-addestrato fornito da Spacy[16], il cui nome è en_core_web_trf (esistono molte varianti di tale modello, nel nostro caso è stato scelto il modello con precisione più alta nel rilevare le NER, a discapito dei tempi molto più lunghi per il processing). Esso consente di estrarre entità facenti parte del seguente insieme: CARDINAL, DATE, EVENT, FAC, GPE, LANGUAGE, LAW, LOC, MONEY, NORP, ORDINAL, ORG, PERCENT, WORK_OF_ART, PERSON, PRODUCT, QUANTITY, TIME. Nel nostro caso abbiamo eliminato dal testo tutte le entità rilevate, escludendo WORK_OF_ART, PRODUCT, LAW e FAC, poiché si è visto che la loro eliminazione peggiorava la qualità del language_detector, ciò probabilmente è dovuto alla scarsa precisione

con cui il modello rileva tali Named Entities[3].

Fatto ciò è stato applicato il metodo predict("string") di FastText, il quale restituisce due valori: *language* (lingua estratta) e *confidence* (un valore che rappresenta quanto sia attendibile il risultato ottenuto). Nel nostro caso si è stabilita una soglia minima di confidence per considerare corretta la lingua estratta, tale valore corrisponde a 0.85. Nel caso in cui venga restituita una lingua con confidence superiore a tale soglia allora la lingua estratta viene inserita nel DB in corrispondenza della colonna `language_detected`; se la confidence è inferiore alla soglia minima allora nella colonna `language_detected` viene inserita la stringa *"low confidence"*.

### Calcolo TF-IDF

L'obiettivo di questa sezione è quello di estrarre le parole chiave, cioè le parole che meglio rappresentano il contenuto del documento. Tra i possibili approcci, si è deciso di utilizzare il TF-IDF: una funzione molto utilizzata in Information Retrieval per misurare quanto un termine risulta importante rispetto ad un documento o ad una collezione di documenti. Tale indice è rappresentato come il prodotto tra TF (Term Frequency: direttamente proporzionale al numero di occorrenze della parola nel documento) ed IDF (Inverse Document Frequency: inversamente proporzionale al numero di documenti del corpus che contengono la parola).

Formalmente, un documento $G$ è definito come un insieme di termini, cioè $G = \{g_1, \ldots, g_n\}$. Dato un corpus di documenti $\mathcal{G}$, un documento $G_i \in G$, un insieme di termini di ricerca $T$, un termine $t_i \in T$ da confrontare, la funzione TF-IDF tra il termine $t_i$ e il documento $G_j$ è definita come TF-IDF$(t_i, G_j, \mathcal{G})$=$TF(t_i, G_j)$*$IDF(t_i, \mathcal{G})$, dove TF e IDF sono definite come di seguito:

- $TF(t_i, G_j)$=$\frac{numOcc(t_i, G_j)}{|G_j|}$
- $IDF(t_i, \mathcal{G})$=$log_{10}(\frac{|\mathcal{G}|}{numDoc(t_i, \mathcal{G})})$

dove $numOcc(t_i, G_j)$ è il numero di occorrenze di $t_i$ in $G_j$, cioè $numOcc(t_i, G_j) = |\{g_k \in G_j : g_k = t_i\}|$. La funzione $numDoc(t_i, \mathcal{G})$ rappresenta invece il numero di documenti di $\mathcal{G}$ che contengono $t_i$, cioè $numDoc(t_i, \mathcal{G})$=$|\{G_k \in \mathcal{G} : t_i \in G_k\}|$.

[3] Come è possibile osservare dai dati relativi alla precisione presenti nel repository GitHub accessibile al link https://github.com/explosion/spacy-models/blob/en_core_web_trf-3.1.0/meta/en_core_web_trf-3.0.0a1.json

Prima di estrarre le keyword è stato necessario creare un nuovo DB che chiameremo keywDB, nel quale esse saranno inserite. Il codice per la creazione è disponibile nel repository GitHub sotto il nome di *"DB_key_create.mysql"*.

Dal DB sono stati estratti tutti i documenti in lingua inglese (cioè in cui il campo `language_detected` ha valore `en`), e per ognuno di essi sono stati concatenati titolo e descrizione. Successivamente il testo è stato processato eliminando simboli, link, email, parole contenenti caratteri non inglesi (poiché i documenti sono tutti in inglese perciò se sono presenti parole con caratteri non inglesi potrebbero essere parole che rappresentano nomi stranieri, università etc cioè elementi che non sono significativi) e Named Entities. Successivamente la stringa ottenuta è stata trasformata in una lista di parole.
Il lemmatizer che verrà utilizzato è il *WordnetLemmatizer()*. Wordnet[19] è un database lessicale in lingua inglese che cerca di stabilire relazioni semantiche tra parole e fornisce supporto per il processo di lemmizzazione. Grazie ad NLTK è possibile utilizzare un'interfaccia per collegarsi ad esso, ma prima è necessario scaricarla nel proprio ambiente python.
Analizzando i benchmark[20] relativi ai vari lemmatizer è possibile osservare come il WordnetLemmatizer() ottenga delle prestazioni migliori nel caso in cui riceva in input oltre alla parola da lemmizzare anche il relativo POS_tag. Il POS_tagging (Part of Speech) è un processo che consente la marchiatura delle parole in base al loro contesto all'interno di una frase, tale processo può essere manuale (infattibile per testi lunghi) o automatico; nel nostro caso verrà utilizzato il metodo pos_tag() di NLTK[10] per assegnare ad ogni parola del testo il corrispondente tag di contesto; poiché tale modello è stato addestrato con un corpus Treebank, utilizza i suoi stessi tag, i quali sono disponibili al link https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html, i principali sono rappresentati di seguito:

- ADJ: aggettivo
- ADP: preposizione
- ADV: avverbio
- CONJ: congiunzione
- DET: articolo determinativo
- NOUN: nome
- PRON: pronome

- VERB: verbo

Applicando il metodo pos_tag() alla lista di parole precedentemente creata, si ottiene una lista di tuple con formato `(parola, pos_tag_Treebank)`.
Il WordnetLemmatizer() però utilizza un insieme di POS_tag[4] diverso, in particolare riconosce solo 4 tag specifici: j,r,n,v. Si è quindi utilizzato un dictionary che mappa i tag NLTK nei corripondenti tag Wordnet richiesti dal lemmatizer:

- ADJ –> 'j'
- ADV –> 'r'
- NOUN –> 'n'
- VERB –> 'v'

Facendo ciò si passa da pos_tag_nltk a pos_tag_wordnet, il quale può assumere 5 valori: "j","r","n","v","None", quest'ultimo valore è assunto nel caso in cui il tag NLTK non abbia alcun corrispettivo in Wordnet.
A questo punto, per ogni parola della lista di tuple, se il tag mappato è diverso da "None", applichiamo il lemmatizer e, se la parola lemmizzata ha più di un carattere (altrimenti non è significativa), essa viene inserita in una lista, che, dopo aver analizzato tutte le parole, conterrà tutti i lemmi del documento analizzato.
Come già detto, utilizzando i POS_tag sono migliorate le prestazioni del lemmatizer in fatto di precisione Inoltre sono state lemmizzate solo le parole che corrispondono a: verbi, nomi, aggettivi ed avverbi, evitando di lemmizzare pronomi o connettivi, i quali non sono significativi.

Tale processo è stato ripetuto per ogni documento ottenendo un dictionary con formato `{"DOI_documento":[lista_lemmi_documento]}`. Il dizionario è poi stato trasformato in un Dataframe sfruttando la libreria pandas[13] e successivamente per ogni parola è stato calcolato il TF-IDF ed il numero di occorrenze per ogni documento; in entrambi i casi sono stati utilizzati 2 metodi del package sklearn[15]:

- Per il conteggio si è utilizzato il metodo CountVectorizer di sklearn.feature_extraction.text.
- Per il calcolo del TF-IDF si è utilizzato il metodo TfidfVectorizer di sklearn.feature_extraction.text.

[4] Come è osservabile all'interno della documentazione disponibile al link http://www.nltk.org/_modules/nltk/corpus/reader/wordnet.html

In entrambi i casi sono state eliminate le parole che compaiono in un singolo documento e le parole che compaiono in più del 75%
Successivamente per ogni pubblicazione sono state estratte le parole con TF-IDF $>$ 0.1 e sono state inserite con il corrispondente conteggio nel keywDB precedentemente creato.La soglia minima è stata posta a 0.1 poiché in questo modo possono essere eliminate tutte le parole che non sono per nulla significative, come ad esempio i verbi modali o parole molto comuni.
Un esempio del risultato ottenuto dopo questa operazione è rappresentato nella figura 3.3.

Per estrarre le migliori 20 parole chiave per ogni documento è stata utilizzata una query SQL e i risultati sono stati inseriti in un file CSV (il codice è disponibile nel file *"estrazione_best_20_keywords"* nella repository GitHub definita nella sezione 2 a pagina 4). Un esempio del risultato ottenuto dopo quest'operazione può essere osservato nelle figure 3.4, 3.5 e 3.6; nella prima figura sono rappresentati i DOI dei documenti con le prime 6 parole chiave, poi nelle altre figure sono rappresentate le restati parole chiave.
Si è deciso di prendere in considerazione le prime 20 parole in quanto utilizzando l'API definita nella sezione 3.4.3 a pagina 28 la precisione delle classificazioni risulta essere massima.

| | | | |
|---|---|---|---|
| 10.1093/bioinformatics/btab160 | grant | 3 | 0.306881 |
| 10.1093/bioinformatics/btab160 | image | 3 | 0.252343 |
| 10.1093/bioinformatics/btab160 | institute | 1 | 0.131429 |
| 10.1093/bioinformatics/btab160 | lab | 1 | 0.114386 |
| 10.1093/bioinformatics/btab160 | metric | 2 | 0.220337 |
| 10.1093/bioinformatics/btab160 | microscope | 1 | 0.137475 |
| 10.1093/bioinformatics/btab160 | microscopy | 2 | 0.204588 |
| 10.1093/bioinformatics/btab160 | no | 1 | 0.1051 |
| 10.1093/bioinformatics/btab160 | num | 1 | 0.137475 |
| 10.1093/bioinformatics/btab160 | phd | 1 | 0.137475 |
| 10.1093/bioinformatics/btab160 | procedure | 1 | 0.112172 |
| 10.1093/bioinformatics/btab160 | quality | 2 | 0.174159 |
| 10.1093/bioinformatics/btab160 | robust | 2 | 0.213315 |
| 10.1093/bioinformatics/btab160 | support | 3 | 0.212931 |
| 10.1093/bioinformatics/btab160 | visit | 1 | 0.102294 |
| 10.1093/bioinformatics/btab160 | zip | 1 | 0.106658 |
| 10.1093/cvr/cvaa193 | chest | 1 | 0.223633 |
| 10.1093/cvr/cvaa193 | clinical | 1 | 0.139927 |
| 10.1093/cvr/cvaa193 | cohort | 1 | 0.185289 |
| 10.1093/cvr/cvaa193 | compute | 1 | 0.201433 |
| 10.1093/cvr/cvaa193 | death | 2 | 0.329906 |

Figure 3.3: Rappresentazione delle keyword nel keywDB

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 10.1093/bioinformatics/btab160 | fund 4 | grant 3 | comparable 2 | image 3 | metric 2 | robust 2 |
| 10.1093/cvr/cvaa193 | myocardial 2 | injury 2 | death 2 | risk 2 | tomography 1 | chest 1 |
| 10.1101/2020.04.22.20075143 | blood 8 | rrt 4 | machine 5 | routine 4 | feasibility 3 | learn 4 |
| 10.1101/2020.11.09.20228858 | systematic 2 | meta 2 | review 2 | indicator 1 | analysis 2 | course 1 |
| 10.1136/ heartjnl-2020-317322 | cardiac 6 | biomarkers 5 | hs 4 | elevated 4 | elevate 4 | bnp 3 |
| 10.1515/cclm-2020-1294 | external 6 | validation 6 | value 7 | stand 4 | feature 5 | internal 3 |
| 10.18632/aging.202307 | mortality 9 | test 6 | vitamin 3 | age 5 | gender 3 | nursing 2 |
| 10.25338/B82G9B | glycosylated 4 | fc 3 | ace2 3 | variant 3 | configuration 2 | rbd 2 |
| 10.25338/B8C91C | acid 4 | meta 3 | coffee 2 | sensory 2 | chemical 2 | composition 2 |
| 10.25338/B8M92X | mediator 3 | permeability 2 | acid 3 | regulatory 2 | patient 4 | plasma 2 |

Figure 3.4: CSV con keywords_1

| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|
| support 3 | microscopy 2 | quality 2 | benchmarking 1 | costa 1 | funded 1 | germany 1 |
| compute 1 | study 2 | insight 1 | cohort 1 | factor 1 | clinical 1 | patient 1 |
| pcr 4 | test 5 | positive 4 | develop 4 | alt 2 | hematochemical 2 | patient 5 |
| full 1 | predict 1 | severe 1 | dataset 1 | | | |
| optimise 3 | stratification 3 | patient 6 | covid 8 | tni 2 | mortality 3 | hospital 3 |
| blood 4 | dataset 7 | hematochemical 2 | auc 2 | expensive 2 | rrt 2 | complete 3 |
| home 3 | resident 2 | due 3 | diagnostic 2 | swab 2 | previous 2 | male 2 |
| fully 2 | domain 2 | spike 2 | protein 2 | glycans 1 | promise 1 | fusion 1 |
| measurement 2 | different 3 | concentration 2 | review 2 | analysis 3 | organic 1 | distinct 1 |
| burn 1 | metabolism 1 | omics 1 | toxicity 1 | covid 4 | chemistry 1 | degradation 1 |

Figure 3.5: CSV con keywords_2

| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|
| microscope 1 | num 1 | phd 1 | institute 1 | clear 1 | acquire 1 | lab 1 |
| | | | | | | |
| covid 8 | crp 2 | expensive 2 | reference 3 | platelet 2 | negative 3 | gold 2 |
| | | | | | | |
| admission 2 | detection 2 | | | | | |
| use 6 | positive 3 | specific 3 | alternative 2 | routine 2 | sex 2 | train 2 |
| old 2 | associate 3 | high 3 | negative 2 | scale 2 | score 2 | low 2 |
| ns 1 | binding 1 | pdb 1 | therapeutic 1 | | | |
| least 1 | extract 1 | point 1 | | | | |
| illuminate 1 | later 1 | ameliorate 1 | biomarker 1 | vascular 1 | disease 2 | associate 2 |

Figure 3.6: CSV con keywords_3

### 3.4.3 Classificazione

La sezione corrente ha come obiettivo la classificazione dei documenti presenti nel DB.
Per farlo si è deciso di utilizzare Dandelion[17], un sevizio Web che consente l'analisi di dati testuali tramite chiamate ad un'API. Registrandosi al sito di Dandelion è possibile riscattare un codice al quale sono associati 1000 token giornalieri. I token possono essere sfruttati per diverse tipologie di operazioni, come: estrazione di entità, sentiment analysis, estrazione di keyword etc.
Nel nostro caso l'operazione richiesta è la classificazione di documenti in funzione di alcuni argomenti da noi definiti; ciò ha un costo di 2 token per ogni documento elaborato.
Il vantaggio di quest'API sta nel fatto che è possibile creare un modello utilizzando pagine Wikipedia per il training, risparmiando perciò molto tempo, nonostante si abbia il vincolo di poter processare al massimo 500 documenti al giorno (a fronte dei circa 700 documenti in lingua inglese nel DB).

#### Creazione modello

Dopo aver analizzato in maniera generale i possibili ambiti con cui classificare le pubblicazioni e per poter confrontare i risultati ottenuti con quelli dell'analisi "How do we share data in COVID-19 research? A systematic review of COVID-19 datasets in PubMed Central Articles"[1] presentata in precedenza, si è deciso di utilizzare i seguenti ambiti per la classificazione dei documenti:

- Ambito Clinico
- Ambito Genomico
- Ambito Psicologico
- Ambito Epidemiologico
- Ambito Informatico
- Social Network

Per creare il modello è necessario creare un oggetto JSON contenente alcuni attributi:

- description: nome del modello;
- lang: lingua con cui sono scritte le pagine Wikipedia utilizzate nel modello;

- categories: vettore contenente un oggetto JSON in cui sono rappresentati gli ambiti con i rispettivi topic:
  - name: nome dell'ambito
  - topics: oggetto JSON contenente i titoli delle pagine Wikipedia riferite all'ambito corrente, con il corrispondente peso, il quale sta ad indicare quanto ogni topic sia rilevante rispetto a tutti gli altri.

Se ad esempio si volesse creare un modello che classifichi i documenti come appartenenti all'ambito dell'*Intelligenza Artificiale* o all'ambito dei *Social Network* si potrebbe strutturare un JSON con questa forma:

Listing 3.3: API Dandelion: model

```
modello = {
  "description": "classificatore Web",
  "lang": "en",
  "categories": [
    {
      "name": "intelligenza artificiale",
      "topics": {
        "Artificial_intelligence": 1.0,
        "Neural_network": 1.0,
        "Deep_learning": 1.0,
        "Supervised_learning":1.0,
        "Reinforcement_learning":1.0,
        "Natural_language_processing":1.0
      }
    },
    {
      "name": "social network",
      "topics": {
        "Social_media":1.0,
        "Reddit": 1.0,
        "Twitter": 1.0,
        "Facebook": 1.0,
        "Social_network": 1.0
      }
    }
  ]
}
```

Il modello finale utilizzato per questo progetto è disponibile nella repository GitHub sotto il nome di *"modello_API_Dandelion.json"*.

Dopo aver strutturato il JSON, è possibile creare il modello tramite una chiamata con verbo `POST` come quella riportata di seguito:

```
requests.post("https://api.dandelion.eu/datatxt/cl/
   models/v1", params = {"data": "modello", "token": "
   xxx"})
```

Nel caso in cui la struttura del JSON sia corretta, viene restituito il messaggio `<Response [200]>`, invece in caso di errore viene restituito `<Response [400]>`.

### Utilizzo modello

Per poter utilizzare il modello per catalogare una stringa di caratteri, è necessario svolgere una richiesta di tipo `GET` all'API con il seguente formato:

```
testo_catalogato = requests.get('https://api.
   dandelion.eu/datatxt/cl/v1',
params = {'token': xxx, 'text':yyy,
'model':id modello})
```

Il primo attributo rappresenta il link relativo all'API.
Il *token* è un codice fornito da Dandelion subito dopo la registrazione.
Il *text* rappresenta il testo da catalogare.
Il *model* è l'identificativo del modello, il quale può essere ottenuto tramite la richiesta:

```
requests.get('https://api.dandelion.eu/datatxt/cl/
   models/v1', params = {'token': xxx})
```

la quale restituisce la lista dei modelli con i corrispondenti parametri, compreso l'identificatore.

Come da documentazione, l'API è performante solamente per stringhe brevi, perciò per classificare i documenti si è deciso di utilizzarla passando in ingresso le keyword anziché i testi completi.
Inoltre si è visto che l'API tiene anche conto delle occorrenze delle parole, motivo per cui è stato necessario, nelle sezioni precedenti, considerare anche il conteggio, oltre al TF-IDF, all'interno dei singoli documenti, in modo da dare più importanza alle parole ripetute più volte.
Come visto nella sezione 3.4.2 a pagina 22, si è deciso di considerare come keyword di un documento le 20 parole con TF-IDF più alto, le quali sono state inserite in un file CSV in cui la prima colonna corrisponde al DOI del documento e le successive corrispondono alle parole chiave.
Il file *"classificazione_keyword.py"* nella repository GitHub ha come obiettivo l'estrazione delle parole chiave dal suddetto CSV e la classificazione dei documenti. In particolare, il CSV è stato estratto e convertito in dataframe; successivamente per ogni riga del dataframe sono state concatenate le keyword tenendo conto del corrispondente conteggio. Ad esempio supponendo di avere la seguente riga del CSV:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 10.1101/2020.11.09.20228858 | systematic 2 | meta 2 | review 2 | indicator 1 | analysis 2 | course 1 | full 1 | predict 1 | severe 1 | dataset 1 |

La stringa ottenuta in seguito alla concatenazione è la seguente:

```
"systematic systematic meta meta review review
   indicator analysis analysis course full predict
   severe dataset"
```

A questo punto è stato creato un dictionary in cui inserire per ogni documento la corrispondente lista di parole concatenate, come riportato di seguito:

```
{"DOI_della_pubblicazione": "keywords_concatenate"}
```

Successivamente per ogni elemento del dictionary (cioè per ogni documento) è stata svolta una chiamata all'API passando come parametro *text* la stringa costituita dalle parole concatenate. Un esempio di risposta ad una singola chiamata all'API è la seguente:

```
{
  '10.1093/cvr/cvaa193': {
    'time': 6,
    'categories': [
      {
        'name': 'ambito epidemiologico',
        'score': 0.6635944
      },
      {
        'name': 'ambito clinico',
        'score': 0.4559046
      },
      {
        'name': 'ambito psicologico',
        'score': 0.4339194
      },
      {
        'name': 'ambito genomico',
        'score': 0.4124909
      },
      {
        'name': 'ambito informatico',
        'score': 0.2632625
      },
      {
        'name': 'social network',
        'score': 0.07400844
      }
    ],
```

```
    'lang': 'en',
    'timestamp': '2021-10-15T12:26:54.730'
  }
}
```

Nel caso in cui nel JSON esista almeno un ambito con score $> 0.4$ il documento viene inserito in un file CSV, nel quale troviamo nella prima colonna il DOI del documento e nelle colonne successive tutti gli ambiti che hanno superato la soglia di 0.4 ordinati in ordine decrescente di score.
Si è deciso di non inserire solamente l'ambito con valore più alto poiché in alcuni casi ci sono argomenti parecchio legati tra loro, come ad esempio Ambito Psicologico e Social Network; in questo modo è quindi possibile rappresentare delle classificazioni multiple. Un esempio di come è strutturato tale CSV è rappresentato nell'immagine 3.7.

In seguito alcuni dei documenti sono stati esaminati da un operatore umano e sono stati classificati manualmente in modo tale da confrontare i risultati con quelli forniti dall'API.
La percentuale di successo ottenuta però si è rivelata essere mediocre, ciò è dovuto principalmente al fatto che per definire il modello è necessario avere una conoscenza approfondita degli argomenti trattati, in modo da scegliere le pagine Wikipedia più significative.
Uno sviluppo successivo del progetto potrebbe mirare all'ottimizzazione del processo di classificazione, in modo da ottenere precisione sempre più elevata utilizzando tecniche alternative di machine learning.
I risultati della classificazione ottenuta in seguito all'utilizzo dell'API sono osservabili nel grafico 4.2 nella sezione 4.2.

| | | | | |
|---|---|---|---|---|
| 10.1093/cvr/cvaa193 | ambito epidemiologico (0.6635944) | ambito clinico (0.4559046) | ambito psicologico (0.4339194) | ambito genomico (0.4124909) |
| 10.1101/2020.04.22.20075143 | ambito clinico (0.49572983) | ambito genomico (0.45973438) | | |
| 10.1101/2020.11.09.20228858 | ambito epidemiologico (0.47471595) | | | |
| 10.1136/ heartjnl-2020-317322 | ambito epidemiologico (0.5800308) | | | |
| 10.1515/cclm-2020-1294 | ambito epidemiologico (0.42643213) | | | |
| 10.18632/aging.202307 | ambito epidemiologico (0.7165884) | ambito clinico (0.45550007) | ambito psicologico (0.43395284) | ambito genomico (0.401147) |
| 10.25338/B82G9B | | | | |
| 10.25338/B8C91C | ambito genomico (0.46915594) | ambito clinico (0.43576512) | | |
| 10.25338/B8M92X | ambito genomico (0.5959719) | ambito clinico (0.4061688) | | |
| 10.25338/B8ZS7R | | | | |
| 10.25583/1760388 | ambito genomico (0.44581023) | | | |
| 10.3390/ijerph17249533 | ambito clinico (0.41894162) | | | |
| 10.5061/dryad.02v6wwpzr | ambito psicologico (0.41457367) | | | |
| 10.5061/dryad.08kprr52t | | | | |
| 10.5061/dryad.0cfxpnw32 | ambito psicologico (0.6494706) | ambito clinico (0.5036363) | ambito epidemiologico (0.407499) | |
| 10.5061/dryad.0k6djh9z4 | | | | |
| 10.5061/dryad.0k6djhb1d | ambito clinico (0.5184724) | ambito psicologico (0.508823) | ambito epidemiologico (0.4390016) | |
| 10.5061/dryad.0vt4b8gzq | ambito genomico (0.5107015) | ambito epidemiologico (0.45322958) | ambito clinico (0.42230245) | |

Figure 3.7: CSV pubblicazioni classificate

# Capitolo 4

# Risultati

Dopo aver esaminato e classificato i documenti, è stato possibile analizzare dati e metadati associati ad essi. In particolare l'attenzione è stata rivolta verso l'accessibilità.

Le statistiche proposte sono tutte riferite ai documenti estratti ed elaborati in data 28/09/2021.

Come già detto in precedenza, queste statistiche cercano di ricalcare il più possibile quelle trovate nell'articolo "How do we share data in COVID-19 research? A systematic review of COVID-19 datasets in PubMed Central Articles"[1] al quale si sta facendo riferimento. In alcuni casi però, non è stato possibile seguire la stessa strada, poiché alcune informazioni non sono fornite da Zenodo. Ad esempio non è disponibile alcun dato relativo alla posizione geografica dalla quale sono stati caricati i dataset, inoltre il sistema per gestire le citazioni è ancora in beta ed è poco utilizzato, perciò non è possibile tracciare un grafo che rappresenti i collegamenti tra le varie pubblicazioni (in realtà alcuni dataset sono provvisti di un "identificatore relazionale" definito nel JSON con il quale viene rappresentato il DOI del dataset citato, ma è un sistema poco utilizzato, infatti solamente 24 dataset su 1265 presentano questo attributo, perciò in questa analisi è stato trascurato).

## 4.1 Analisi estrazione lingua

Il numero totale di documenti all'interno del DB è pari a 1265. Utilizzando le tecniche proposte nella sezione 3.4.2 a pagina 21, tali documenti sono stati divisi in base alla lingua estratta, il grafico 4.1 riporta i risultati ottenuti:

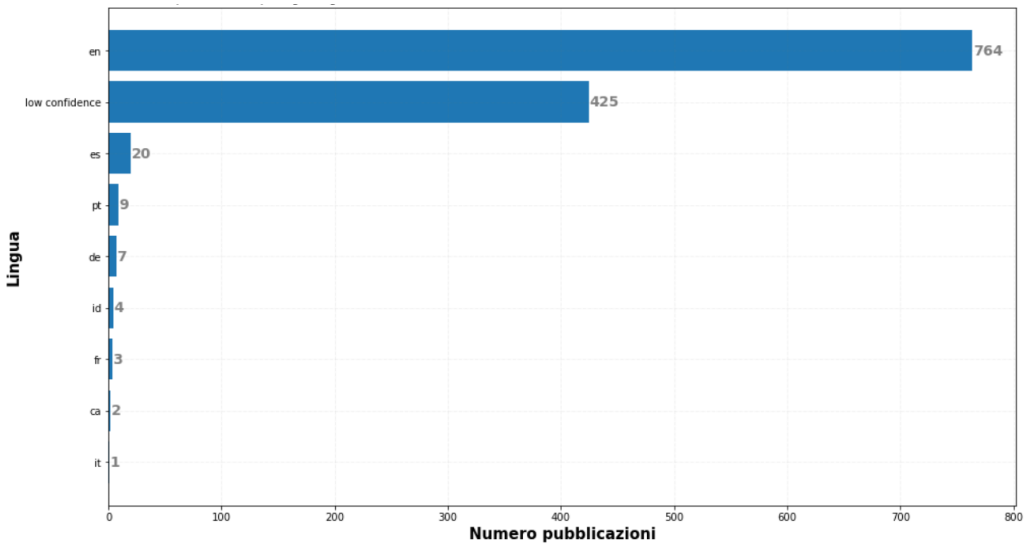


Figure 4.1: Divisione in base alla lingua

Già analizzando il JSON che descrive le singole pubblicazioni, si può vedere come effettivamente esista un campo "language", nel quale però è definita la lingua dei dataset, mentre nel nostro caso abbiamo estratto la lingua del titolo e della descrizione. Confrontando i due valori, si può facilmente osservare che essi non corrispondono mai, se non in rarissimi casi (bisogna anche tener conto che in molte casistiche il campo "language" nel JSON non è neppur inserito, poiché non significativo, ad esempio nel caso in cui il dataset è un'immagine o un file testuale con la rappresentazione di una molecola).

In ogni caso, il fatto che la maggior parte dei dataset abbia descrizione e titolo in Inglese li rende sicuramente più facilmente interpretabili da operatori umani.

## 4.2 Analisi risultati classificazione

Dei 764 documenti in lingua inglese solamente 349 sono stati classificati con una confidence superiore allo 0.4. La percentuale di documenti classificati per ogni ambito è rappresentata nel diagramma 4.2.

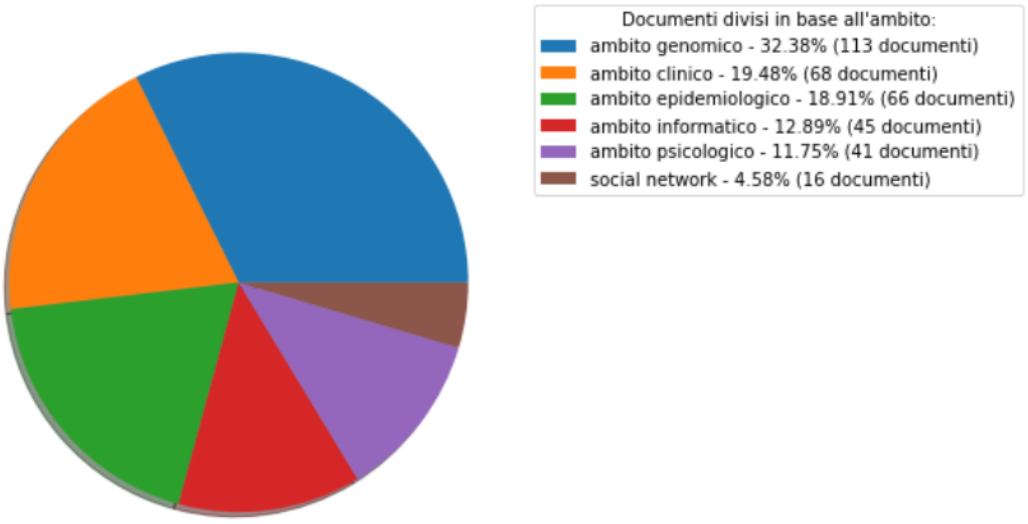


Figure 4.2: Classificazione

Come già detto nella sezione 3.4 i risultati forniti sono soggetti ad errore. Ciò è dovuto a diversi fattori: l'API sfrutta pagine Wikipedia per definire il modello, perciò è necessario selezionare in maniera accurata le pagine che meglio rappresentano i vari ambiti, operazione che deve essere svolta necessariamente da un esperto di dominio. Inoltre 4 degli ambiti definiti riguardano il macro-ambito "medico", perciò le pagine Wikipedia che descrivono questi argomenti utilizzano un lessico simile, motivo per cui il modello fa molta fatica a distinguere tra un ambito e l'altro.

In Figura 4.3 è riportata la matrice di confusione utilizzata per valutare la bontà del modello. Le righe rappresentano gli ambiti reali, le colonne gli ambiti ottenuti dopo la classificazione. Nella riga *"altro"* sono rappresentati i documenti ai quali il modello ha attribuito uno dei 6 ambiti definiti, ma che in realtà trattano argomenti che non riguardano nessuno tra questi ambiti.

Per ogni ambito sono stati ricavati dalla matrice 4 valori fondamentali:

- TP (true positive): numero di documenti correttamente classificati con l'ambito corrente;

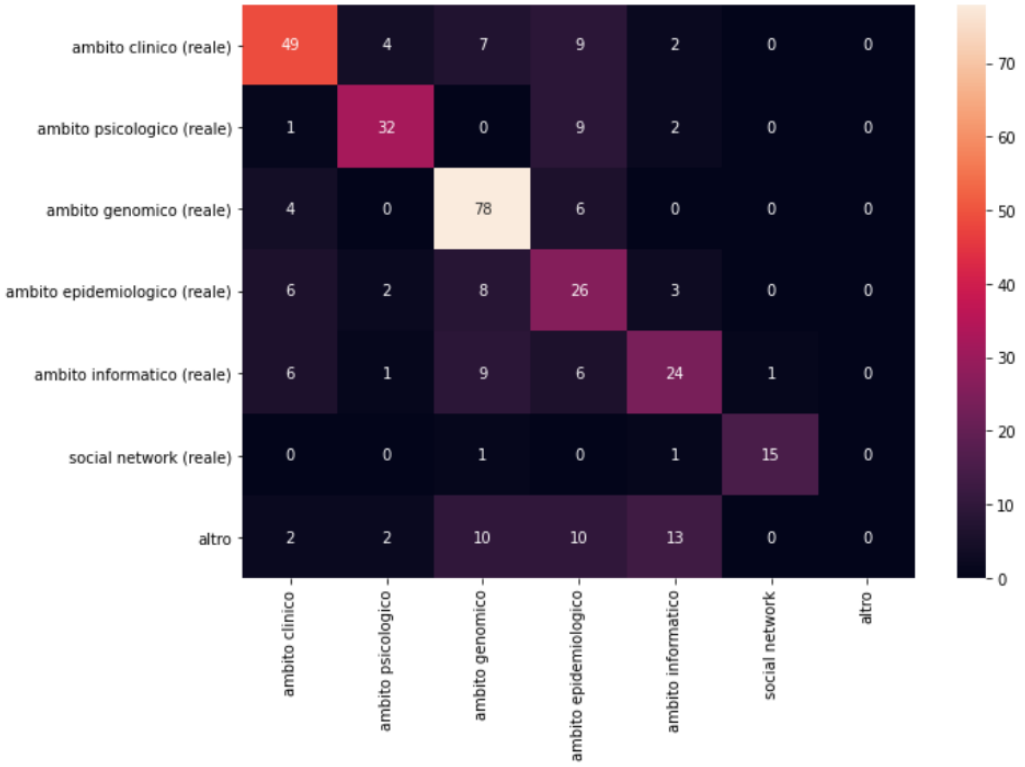


Figure 4.3: Matrice di confusione

- TN (true negative): numero di documenti che non sono stati classificati con l'ambito corrente e non trattano tale ambito;
- FP (false positive): numero di documenti che sono stati erroneamente classificati con l'ambito corrente;
- FN (false negative): numero di documenti che non sono stati classificati con l'ambito corrente, ma in realtà trattano tale ambito.

A partire da questi 4 valori sono stati calcolati alcuni indici che rappresentano la qualità del modello relativamente ad uno specifico ambito:

- Accuracy: accuratezza del modello, ovvero la frazione di campioni totali che sono stati correttamente classificati: $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision: frazione di documenti classificati positivamente sul numero totale di documenti classificati con l'ambito attuale: $\frac{TP}{TP+FP}$
- Recall: frazione di documenti classificati positivamente sul numero totale di documenti che realmente trattano l'ambito attuale: $\frac{TP}{TP+FN}$

- F1-score: combina precisione e recall: $\frac{2TP}{2TP+FP+FN}$

| Ambito | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Ambito Clinico | 88.25% | 0.72 | 0.69 | 0.71 |
| Ambito Psicologico | 93.98% | 0.78 | 0.73 | 0.75 |
| Ambito Genomico | 87.11% | 0.69 | 0.89 | 0.78 |
| Ambito Epidemiologico | 83.09% | 0.39 | 0.58 | 0.47 |
| Ambito Informatico | 87.39% | 0.53 | 0.51 | 0.52 |
| Social Network | 99.14% | 0.94 | 0.88 | 0.91 |

La precisione generale del modello è stata valutata a partire da 2 indici:

- Accuracy media: media aritmetica delle *"Accuracy"* di tutti gli ambiti di classificazione; ha valore pari a 89.82%;
- Overall Accuracy: percentuale di documenti classificati correttamente sul numero totale di documenti: 64.18%.

## 4.3 Analisi di Accessibilità

L'accessibilità è stata definita in base a 3 fattori:

- diritti di accesso
- tipologia di licenza
- tipologia di codifica dei file allegati ai dataset

### 4.3.1 Diritti di accesso

Analizzando la documentazione di Zenodo sono state ricavate alcune informazioni riguardanti i diritti di accesso, essi sono divisi in 4 gruppi:

- open: dataset aperto e disponibile (i limiti sull'utilizzabilità dipendono dalla licenza);
- restricted: dataset chiuso, ma accessibile sotto alcune condizioni (richiesta tramite email, iscrizione sito web etc);
- closed: dataset non accessibile sotto nessuna condizione;
- embargoed: dataset temporaneamente inaccessibile.

Zenodo utilizza anche altri 3 termini per classificare i dataset:

- success: corrisponde ai dataset open;
- danger: corrisponde ai dataset restricted e closed;
- warning: corrisponde ai dataset embargoed.

In questa analisi ci riferiremo alla prima tipologia di classificazione, poiché più specifica.

Nel grafico 4.4 sono definite le percentuali relative ai vari diritti di accesso. Si può notare come solamente 2 documenti siano sotto stato di embargoed e saranno rilasciati rispettivamente il 30/12/2021 ed il 30/06/2023.

Tutti i documenti con diritto di accesso restricted sono caratterizzati da un campo JSON nel quale sono definite le condizioni di accesso. All'interno di esse in 47 documenti è presente almeno un'email alla quale rivolgersi per definire le proprie intenzioni riguardo l'utilizzo dei dataset.

Perciò si può affermare che la gran parte dei dataset sono accessibili (open) ed una parte di quelli non accessibili (restricted) può essere comunque reperita contattando il responsabile della pubblicazione.

Inoltre alcune delle pubblicazioni forniscono delle informazioni in più, ad esempio sono presenti il campo note (249 documenti) o il campo method (107 documenti). Entrambi consentono una maggior leggibilità.

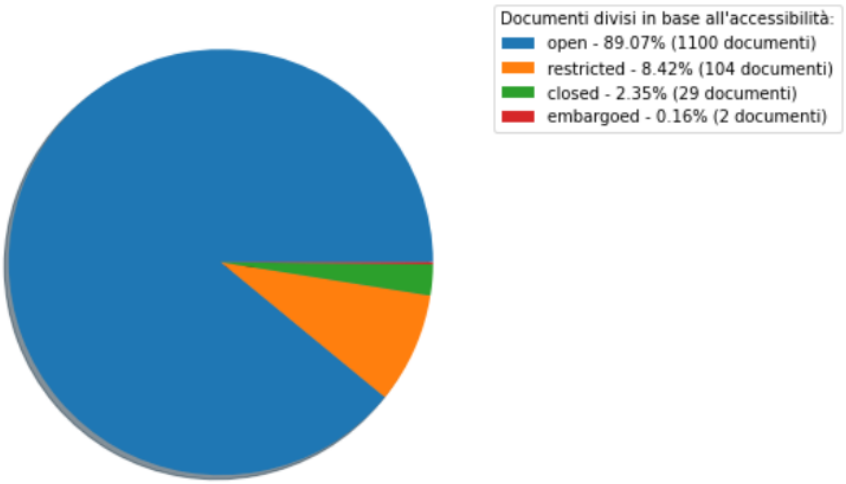


Figure 4.4: Condizioni di accesso

### 4.3.2 Tipologia di licenza

Nel grafico 4.5 è definito il numero di pubblicazioni per ogni tipologia di licenza. In particolare si può notare come le 4 licenze più utilizzate siano tutte aperte:

- CC-BY-4.0/CC-BY-1.0: totale possibilità di utilizzare il dataset, anche per fini commerciali. Necessario citare l'autore ed indicare se sono state effettuate delle modifiche.
- CC0-1.0: nessun diritto d'autore, totale accesso e riutilizzabilità.
- CC-BY-NC-4.0: totale possibilità di utilizzare il dataset, tranne che per fini commerciali. Necessario citare l'autore ed indicare se sono state effettuate delle modifiche.

L'unica licenza che non consente riutilizzabilità dei dati è la CC BY-NC-ND 4.0 (necessario citare l'autore, possibilità di condivisione del dataset, ma non di modifica), ma essa è utilizzata da solamente 3 dataset. La riga con nome "no-license" è relativa ai documenti con diritti di accesso restricted o closed.

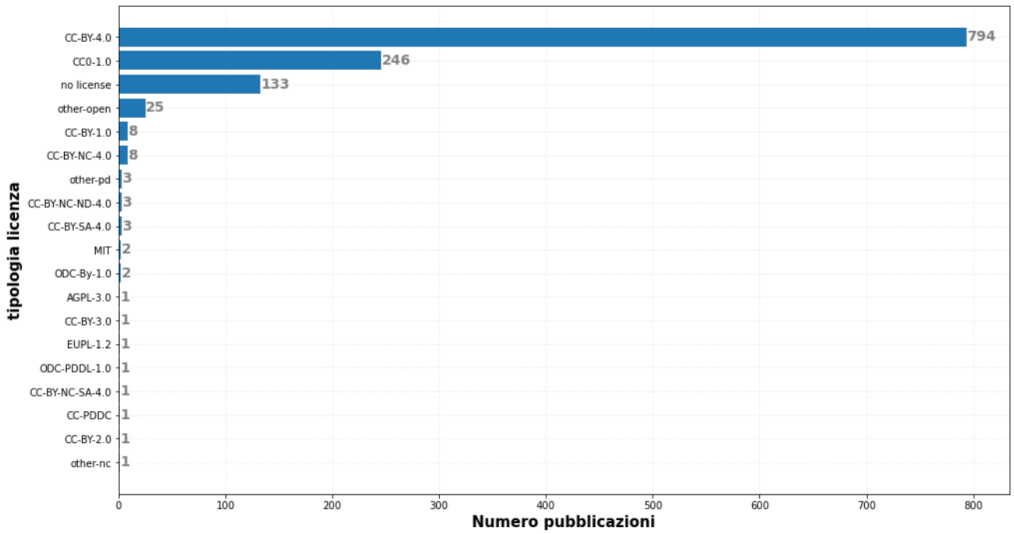


Figure 4.5: Licenze

### 4.3.3 Codifica file

Come già visto, ogni pubblicazione può avere uno o più file allegati. Per definire il grado di utilizzabilità di tali file si sono analizzati i formati più utilizzati, rappresentati nel diagramma 4.6 (sono stati tralasciati i formati utilizzati in 20 o meno file allegati, a fronte dei 9697 file totali, ciò riduce il numero di documenti da analizzare da 1235 a 1013). Tali formati sono stati classificati secondo la gerarchia proposta da Tim Berners-Lee[21]:

- 1 stella: formato con licenza aperta
- 2 stelle: formato con dati strutturati
- 3 stelle: formato non proprietario ed aperto
- 4 stelle: formato con elementi contrassegnati da URI
- 5 stelle: formato con elementi collegati ad altri dataset

I file con "access_right: open" presenti nel DB sono disponibili e regolamentati da licenze aperte, perciò sono classificabili tutti con almeno una stella. Di seguito è riportata una breve descrizione per ogni formato del diagramma e la relativa classificazione.
I formati non classificabili sono formati per immagini o compressi, per i quali non è possibile dire se esistono metadati o dati strutturati all'interno.

| classificazione | formato | descrizione |
|---|---|---|
| 1 stella | txt | formato testuale, può essere strutturato o meno |
| | pdf | formato non strutturato |
| | dat | formato che può contenere qualsiasi tipologia di documento |
| 2 stelle | cbf | formato proprietario per codificare calendari |
| | fcs | formato standard proprietario supportato da una grande quantità di software e hardware per citometria a flusso |
| | xlsx | fogli calcolo Excel |

| | | |
|---|---|---|
| | xtc | formato compresso per descrivere traiettorie |
| | docx | formato proprietario Microsoft |
| | xls | formato binario di Excel |
| | mcd | file binario salvato in formato CAD proprietario |
| | pzfx | formato leggibile solamente tramite l'applicazione Prism, presenta dati strutturati sotto forma di grafici o tabelle |
| 3 stelle | ods | formato aperto formattato usato lo standard OASIS OpenDocument basato su XML |
| | csv | formato testuale in cui i dati sono separati da virgole |
| | nc | formato binario aperto e indipendente dalla piattaforma utilizzato dagli strumenti NetCDF gestiti da Unidata. Il formato NC, tramite API e librerie, è supportato da molti linguaggi di programmazione popolari e facilita la creazione e la condivisione di dati scientifici. Il codice sorgente è scritto in C, mentre librerie e interfacce aggiuntive sono disponibili per Java, C ++ e Fortran |
| | tar | file binario costituito da blocchi, ad inizio file sono presenti metadati che ne descrivono le caratteristiche |
| | pdb | file testuale in cui sono descritte le strutture tridimensionali delle molecole proteiche |
| | mrc | formato standard nella microscopia crioelettronica |
| | pkl | file generato dal modulo "pickle" di Python, consente la serializzazione degli oggetti su disco |
| | fasta | formato testuale per rappresentare sequenze di nucleotidi o sequenze di amminoacidi |
| | tsv | formato testuale per rappresentare valori tabulari |
| | r | formato associato al linguaggio R |
| | mzml | formato aperto basato su XML per file relativi a dati di "spettrometri di massa" |
| | py | codice Python |

| | | |
|---|---|---|
| | json | formato strutturato ed aperto |
| | fa | file in formato FASTA |
| | tabular | file strutturato |
| non classificabili | tif | formato per le immagini |
| | zip | formato compresso |
| | 7z | formato compresso |
| | gz | formato compresso |
| | raw | formato per le immagini, non viene applicata alcuna compressione |
| | mp4 | formato per la rappresentazione di video |
| | fig | immagine generata tramite il linguaggio Matlab |
| | rar | formato compresso |
| | png | formato per le immagini |

Si può facilmente notare come nessuno dei formati presenti possa essere classificato con 4 o 5 stelle.

A partire dai formati ottenuti sono state classificate le pubblicazioni nel DB,

- grado 1 se almeno uno dei file allegati rientra nel gruppo "3 stelle"
- grado 2 se almeno uno dei file allegati rientra nel gruppo "2 stelle"
- grado 3 se tutti i file appartengono al gruppo "1 stella"

Il grafico 4.7 rappresenta esattamente questa divisione: si può osservare come gran parte dei 1013 dataset siano di qualità 3 o 2, ciò indica che la metà delle pubblicazioni ha allegato almeno un file con formato che consente la definizione di metadati ed è di tipo strutturato.

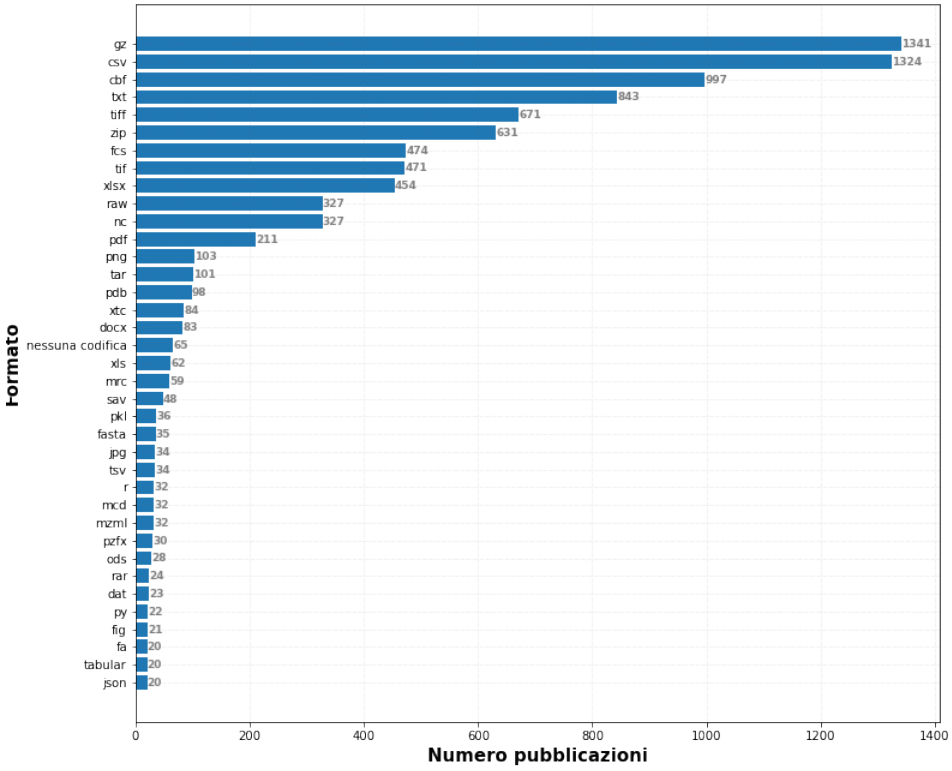


Figure 4.6: Formati più utilizzati

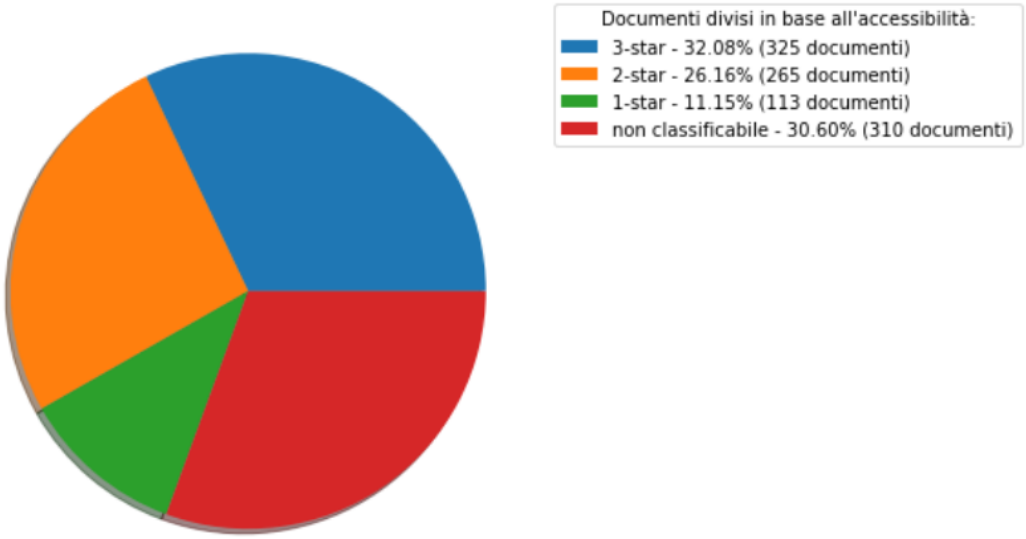


Figure 4.7: Accessibilità

# Capitolo 5

# Conclusioni

In questo progetto di tesi è stata svolta un'analisi riguardo accessibilità, utilizzabilità e interoperabilità dei dataset relativi al Covid-19 presenti nel repository Zenodo.

Per estrarre i metadati riguardanti le pubblicazioni di interesse è stata utilizzata un'API messa a disposizione da Zenodo. Grazie ad essa per ogni dataset sono stati estratti tutti i metadati rilevanti e sono stati inseriti in un database.

Successivamente utilizzando delle librerie per il machine learning è stata estratta la lingua relativa alla descrizione e al titolo di ogni documento; quelli in lingua inglese sono stati perciò pre-processati, ne sono state estratte le migliori 20 parole chiave ed esse sono state utilizzate per classificarli in base a degli ambiti definiti precedentemente.

A partire dai dati presenti nel DB e da quelli ricavati durante la classificazione ed il processing, sono state prodotte alcune statistiche rigurdanti accessibilità, utilizzabilità e interoperabilità dei dataset. Per quest'ultima analisi sono state prese come riferimento le linee guida fornite da Tim Berners-Lee e la classificazione 5-star[21].

Zenodo fornisce un'API molto utile sia per estrarre dati che per depositarli, ciò è un elemento che favorisce l'accessibilità poiché rende molto facile il processo di lettura dei dati da parte di macchine.
D'altra parte non tutti i ricercatori che pubblicano dati su tale piattaforma rispettano le linee guida per garantire accessibilità al dataset, infatti l'11.93% dei dataset non sono immediadatamente accessibili, in particolare il 2.35% sono inaccessibili, l'8.42% accessibili con condizione e lo 0.16% sono sotto embargo. Il restante 89.03% dei documenti è totalmente aperto e disponibile

per il download, inoltre gran parte di essi è anche provvista di una licenza che consente il riutilizzo e la modifica dei dati: è perciò rispettato il principio di Reuse.

Tutti i dataset con condizioni di accesso "open" sono aperti, disponibili per il download e consentono il riutilizzo dei dati, ma solo una parte di essi ha allegati formati che consentono la definizione di metadati (2-star: 35.31%), e solo una parte sono formati non proprietari (3-star: 36.31%), inoltre è presente una percentuale di documenti che contiene formati non classificabili (formati compressi o per immagini: 4.80%). Perciò le pubblicazioni sono state catalogate tutte con meno di 4 stelle, poiché i formati utilizzati non consentono la possibilità di collegare tramite link strutture dati all'interno del documento con strutture dati esterne (come accade invece per RDF: formati standard W3C), per questo i dataset non rispettano a pieno il principio di Interoperability. L'unico strumento che consentirebbe interoperabilità è il sistema di "citing", che però è ancora in beta ed è perciò poco utilizzato.

Come già discusso, Zenodo sfrutta la sintassi ElasticSearch per la ricerca di pubblicazioni nel suo database. Tale ricerca restituisce tutti i documenti in cui è citata almeno una volta la parola cercata, in questo modo vengono restituiti anche documenti che non trattano strettamente l'argomento cercato (ad esempio la pubblicazione "Acids in Coffee - A Review of Sensory Measurements and Meta-Analysis of Chemical Composition"[22] cita la parola Covid-19 solamente nella frase "Access was limited to online versions of publications due to COVID-19 restrictions during the time of the database search", infatti il dataset tratta argomenti completamente slegati rispetto alla pandemia covid). Per evitare la presenza di pubblicazioni non inerenti all'ambito cercato e per rendere più semplice l'analisi e la leggibilità dei dataset sarebbe più adatto classificarli in automatico o obbligare chi carica il dataset ad associare il documento ad uno specifico ambito.
Nel nostro caso avremmo potuto sfruttare le communities, ma non tutti i documenti che trattano il tema "covid-19" si trovano inseriti nella community relativa: alcuni fanno parte di community esterne o ne sono totalmente privi.
Alla luce di ciò, l'implementazione di funzionalità che consentano una ricerca più precisa migliorerebbero sicuramente l'accessibilità dei dataset.

Di seguito sono riportate alcune considerazioni che potranno essere utilizzate per un successivo sviluppo del progetto:

- Potrebbe essere utile sviluppare un miglior modello per la

classificazione, in modo da poter produrre parallelamente statistiche sui documenti totali e statistiche sui documenti dei singoli ambiti di classificazione, per vedere se esistono diversi pattern nei diversi ambiti. Una soluzione alternativa a quella proposta in questo progetto consiste nello sfruttare i word embedding (ad esempio word2vec fornito da Gensim) per trasformare le parole in vettori di numeri reali, in modo da poter rappresentare i vari documenti in funzione di tali vettori ed applicare un algoritmo per il raggruppamento (ad esempio il k-means) ottenendo in output un certo numero di raggruppamenti che rappresenteranno i diversi ambiti con cui classificare i documenti.

- Potrebbe risultare utile utilizzare metadati che nel nostro caso sono stati tralasciati, come ad esempio i ricercatori, le parole chiave o le dimensioni dei dataset, in modo da produrre statistiche più ricche e che non riguardino solamente il principio FAIR.
- Il progetto ed i codici messi a disposizione potrebbero essere utilizzati per proporre un'analoga analisi su piattaforme diverse

# Capitolo 6

# Documenti aggiuntivi

All'interno del file *"Documento_riassuntivo.csv"* presente nel repository GitHub sono riportate le caratteristiche più importanti dei documenti esaminati.
In particolare per ogni documento sono rappresentati:

- DOI: identificativo del dataset (direttamente estratto dal JSON restituito da Zenodo);
- title: titolo del dataset (direttamente estratto dal JSON restituito da Zenodo);
- language: lingua dei dati presenti nei file che compongono i dataset (direttamente estratta dal JSON restituito da Zenodo);
- language_detected: linguaggio con cui sono scritti titolo e descrizione della pubblicazione (tale valore è stato individuato nella sezione 3.4.2);
- ambito_reale: ambito effettivo del dataset (ricavato a partire da un'analisi manuale di titolo e descrizione dei dataset);
- ambito_classificazione: ambito ricavato in seguito all'applicazione del modello per la classificazione (discusso nella sezione 3.4);
- access_right: diritti di accesso al documento (direttamente estratto dal JSON restituito da Zenodo; possibili valori: open, restricted, embargoed, closed; analizzato nella sezione 4.3.1);
- license: licenza a cui è sottoposto il dataset (direttamente estratta dal JSON restituito da Zenodo; discussa nella sezione 4.3.2);
- formati: lista dei formati dei file allegati alle pubblicazioni (direttamente estratta dal JSON restituito da Zenodo);
- star: valore utilizzato per analizzare l'accessibilità dei dataset (tale valore è stato discusso nella sezione 4.3.3);

- updated: data di ultima modifica del dataset (direttamente estratta dal JSON restituito da Zenodo);
- volume: dimensioni in byte del dataset completo (direttamente estratto dal JSON restituito da Zenodo).

# Bibliografia


[1] X. Zuo, Y. Chen, L. Ohno-Machado, and H. Xu, "How do we share data in covid-19 research? a systematic review of covid-19 datasets in pubmed central articles," *Briefings in Bioinformatics*, vol. 22, no. 2, pp. 800–811, 2021.

[2] N. L. of Medicine, "Pubmed® comprises more than 33 million citations for biomedical literature from medline, life science journals, and online books."

[3] P. by CERN Data Centre & Invenio.

[4] M. Wilkinson, M. Dumontier, and I. e. a. Aalbersberg, "The fair guiding principles for scientific data management and stewardship [published correction appears in sci data. 2019 mar 19;6(1):6]," *Sci Data*, Mar 2016.

[5] G. FAIR, "Fair principles," May 2021.

[6] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F. Loizides and B. Schmidt, eds.), pp. 87 – 90, IOS Press, 2016.

[7] R. V. Chandra and B. S. Varanasi, *Python requests essentials*. Packt Publishing Ltd, 2015.

[8] G. Van Rossum, *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.

[9] L. Richardson, "Beautiful soup documentation," *April*, 2007.

[10] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[11] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 427–431, Association for Computational Linguistics, April 2017.

[12] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020.

[13] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, Austin, TX, 2010.

[14] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[16] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "Spacy: Industrial-strength natural language processing in python," 2020.

[17] SpazioDati, "Semantic text analytics as a service."

[18] J. Lee, "Benchmarking language detection for nlp.," *towardsdatascience*, Dec 2020.

[19] C. Fellbaum, *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[20] P. Selva, "Lemmatization approaches with examples in python," *Machine learning +*, October 2018.

[21] H. Michael, "5stardata," january 2012.

[22] S. Yeager, "Acids in coffee - a review of sensory measurements and meta-analysis of chemical composition," 2021.