UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

**Dipartimento di ingegneria Enzo Ferrari**

**Corso di laurea Ingegneria Informatica**

# Documentazione e aggiunte alla libreria per l'entity resolution SparkER

**Relatori:**
Prof.ssa Sonia Bergamaschi
PhD Luca Gagliardelli

**Candidato:**
Arturo Bianchi
Matricola N° 132615

ANNO ACCADEMICO 2020/2021

# Indice

**1 Introduzione** **3**

**2 SparkER** **4**

2.1 Cos'è SparkER . . . . . 4

2.2 Apache Spark e i big data . . . . . 4

2.3 Entity resolution . . . . . 5

2.3.1 Data Deduplication . . . . . 5

2.3.2 Data Linkage . . . . . 6

2.3.3 Blocking . . . . . 7

2.3.4 Token Blocking . . . . . 8

2.3.5 Block Cleaning . . . . . 8

2.3.6 Meta-Blocking . . . . . 9

2.3.7 Entity matching e data fusion . . . . . 9

2.3.8 BLAST . . . . . 10

2.3.9 Esempio real-life . . . . . 11

2.4 Workflow di SparkER . . . . . 12

2.5 Mancanze della libreria . . . . . 12

**3 Aggiunta di funzionalità alla libreria** **13**

3.1 Aggiunta di nuovi formati . . . . . 13

3.1.1 Creare dataframe da formato excel/xlsx . . . . . 13

3.1.2 Creare dataframe da query sql . . . . . 14

3.1.3 Vantaggi dell'usare pandas . . . . . 15

3.2 Metodo facciata . . . . . 16

**4 Documentazione della libreria** **18**

4.1 Strumenti utilizzati . . . . . 18

4.1.1 ReadTheDocs . . . . . 18

4.1.2 Sphinx e reStructuredText . . . . . 18

4.1.3 Sintassi e costrutti utilizzati . . . . . 19

4.2 Struttura della documentazione . . . . . 20

**5 Aggiunta della libreria su PyPI** **22**

5.1 Perchè usare PyPI . . . . . 22

5.2 Creazione del package . . . . . 22

5.3 Upload su PyPI . . . . . 24

**6 Conclusioni** **25**

6.1 Cosa è stato realizzato . . . . . 25

6.2 Difficoltà incontrate e conoscenze acquisite . . . . . 25

6.3 Ringraziamenti . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

**7 Bibliografia** **26**

**8 Sitografia** **27**

# 1 Introduzione

L'oggetto di questa tesi è la descrizione delle procedure e degli strumenti utilizzati per migliorare SparkER, una libreria software scritta in Python.
Il compito che mi è stato assegnato era quello di capire innanzitutto i problemi che tale libreria si pone di risolvere e dunque a grandi linee la teoria che sta alla base di SparkER; in seguito il lavoro è stato capire, a livello di codice, come funzionasse la libreria ed espandere le funzionalità già presenti.
Infine ci si è focalizzati sulla parte più descrittiva del lavoro, la documentazione della libreria su readthedocs e l'aggiunta di SparkER sul repository di PyPi, per permetterne l'installazione attraverso il comando pip install dell'interprete python.
Il seguente elaborato è organizzato come segue:

- una breve spiegazione di SparkER, la teoria su cui si basa, le funzionalità e alcune delle iniziali mancanze;
- come è stata affrontata la programmazione delle funzionalità aggiuntive e delle modifiche;
- gli strumenti usati per la documentazione di SparkER;
- il workflow necessario per inserire la libreria su PyPi.

# 2 SparkER

## 2.1 Cos'è SparkER

SparkER è una libreria open-source scritta in python sviluppata in UNIMORE, il cui obiettivo è quello di fornire algoritmi volti a risolvere in modo efficiente il problema dell'entity resolution che diventa sempre più pressante con l'aumento esponenziale dei dati generati (si arriverà in breve tempo a dover lavorare con gli exabyte e oltre).
La libreria in questione non è standalone, ma necessita della presenza di Apache (Py)Spark sul calcolatore; in questo senso, SparkER può essere considerato come un estensione di Spark che fornisce funzionalità non presenti di default su Spark.

## 2.2 Apache Spark e i big data

Apache Spark è un framework sviluppato per operare in modo efficiente con i big data. Il problema dei big data è che sono di dimensione troppo elevata per essere usati e contenuti in un solo calcolatore/macchina; la possibile soluzione sarebbe distribuirli tra vari cluster di calcolatori. Facendo così, si può utilizzare commodity hardware su tali cluster, ovvero utilizzare del hardware specifico per il compito di lavorare sui big data, consentendo quindi di risparmiare su cpu e dischi, e si interconnettono i cluster attraverso il network, così che l'aggiunta di nuovi cluster sia facile. Anche per questo approccio però si presentano problemi: avere più macchine significa avere un aumento della probabilità di guasti, il fatto che il network sia più lento rispetto alla velocità di scrittura dei dischi causa un aumento della latenza e spesso capita che le macchine non abbiano le stesse prestazioni. Questo va risolto attraverso il software.
L'implementazione MapReduce permette la ripartizione dei dati tra le varie macchine: la funzione map prende in input una serie di dati, applica una funzione fornita dall'utente e produce in output coppie nella forma <chiave,valore>; è presente uno step intermedio detto shuffle, che ridistribuisce i dati tra le varie macchine, così che le stesse chiavi siano assegnate alla stessa macchina; infine lo step di reduce aggrega tutti i valori che hanno la stessa chiave, producendo in output coppie <chiave,[valori]>. Dopo ogni step del MapReduce i risultati intermedi vengono salvati su disco.
Tale implementazione, usata su Apache Hadoop, presenta comunque dei problemi, dovuti ai possibili guasti su una delle macchine, MapReduce più o meno lenti in base alle funzioni che implementano e per questo è complesso bilanciare il carico tra i vari nodi.
Apache Spark nasce in un periodo storico in cui il costo delle RAM è molto diminuito, dunque utilizza, a differenza di Hadoop, un approccio ibrido in cui per salvare i risultati si usano le memorie RAM, aumentando le prestazioni in maniera esponenziale, e nel caso in cui le RAM non siano sufficienti, Spark può abbassare le sue prestazioni utilizzando i dischi proprio come Hadoop.

L'astrazione principale all'interno di Spark è il resilient distributed dataset, o RDD; un RDD è un insieme di elementi immutabile e partizionata che può essere usato in parallelo e a livello implementativo è una classe astratta che contiene tutti i metodi di base (map,reduce,...). Una volta creato, un RDD non può essere modificato e nel caso venga applicata una trasformazione, come un map, viene creato un nuovo RDD; l'RDD tiene traccia di tutte le trasformazioni che vi sono state applicate (utile in caso di danneggiamento dei dati). Inoltre in fase di creazione, si può specificare in quante partizioni dividerlo, consentendo maggiore parallelismo in base al numero di partizioni (bisogna comunque limitare il numero di partizioni, un numero troppo altro potrebbe impattare le prestazioni, si consiglia un numero di partizioni pari a due o tre volte il numero di core nel cluster).
Un programma Spark implementa all'inizio un oggetto di tipo SparkContext; questo comunica a Spark come e dove accedere alle macchine del cluster e attraverso di esso è possibile parallelizzare i dati nel cluster.
SparkER dipende pesantemente da Spark e dalle sue strutture, la maggior parte dei suoi metodi sono stati implementati sfruttando l'implementazione MapReduce del framework e la parte di parallelizzazione è gestita attraverso lo SparkContext.

## 2.3 Entity resolution

Il concetto preponderante alla base di SparkER è l'entity resolution(ER), ovvero l'insieme di tecniche volte a identificare tutte le istanze di un'entità da un data source o da multipli data source che fanno riferimento allo stesso oggetto nella realtà. Molto spesso capita che uno stesso oggetto di cui facciamo una ricerca, su due siti o database diversi, sia rappresentato in modo differente, talvolta capita anche che lo stesso oggetto su una terza fonte sia stato scritto male, e quindi per la presenza dell'errore non si riesca a ottenerlo dalla ricerca; l'entity resolution permetterebbe di bypassare queste differenze o questi errori.
Il problema dell'ER è però la complessità di tale problema, che è quadratica; se si pensa che un dataset può contenere molti terabyte di dati (dell'ordine dei peta/exabyte se si considera aziende veramente grandi) non è pensabile confrontare tutti i profili tra due dataset, serve che queste tecniche eseguano il loro compito in un tempo ragionevole.

### 2.3.1 Data Deduplication

Diamo alcune definizioni per spiegare i prossimi concetti; quando un dataset non contiene duplicati, esso viene definito clean dataset, al contrario, qualora contenga anche solo un elemento duplicato viene definito dirty dataset. La distinzione tra i tipi di dataset è necessaria per differenziare il tipo di operazione che verrà eseguita.

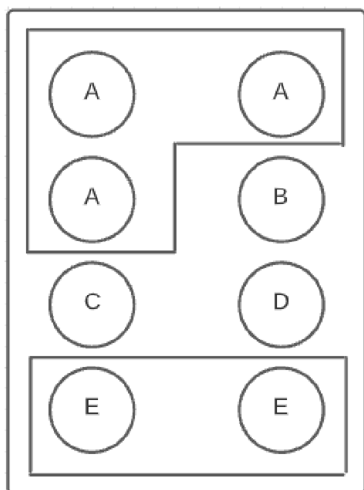


Figura 1: Dirty dataset

Nel caso in cui l'input sia un dirty dataset (anche nel caso di due dataset dirty-dirty o dirty-clean) si parla di data deduplication. Questa operazione consiste in, dato in input un insieme di elementi, trovare ed eliminare tutti i duplicati tra essi presenti e ritornare in output l'insieme clean.
L'esempio in figura 1 mostra un dataset contenente dei duplicati; l'algoritmo che implementa l'operazione di deduplication deve riconoscere tutte le occorrenze degli elementi duplicati (in questo caso ci sono 3 elementi A e 2 elementi E) e in output dovrebbero ritornare il dataset senza duplicati (in questo caso il dataset conterrà A,B,C,D,E).

#### 2.3.2 Data Linkage

L'ultimo possibile input rimasto rispetto ai casi coperti dal data deduplication è il caso del clean-clean. In questo caso l'operazione verrà definita data o record linkage. Dati due o più clean dataset, il data linkage è quell'operazione che permette di trovare gli elementi tra i vari dataset che corrispondono allo stessa persona o oggetto e ritorna un dataset unico in cui ogni elemento corrisponde a un'entità unica; in questo caso non si tratta di duplicati, ma magari di differenze di formato o presenza di errori nei dataset, differenze già citate parlando dell'entity resolution.

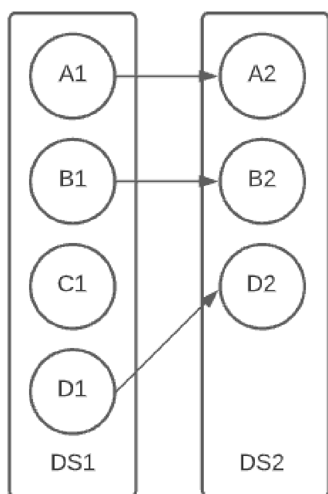


Figura 2: Data linkage

Nell'esempio si possono osservare due clean dataset che non contengono duplicati interni; l'algoritmo che implementa il data linkage deve trovare nei dataset quegli elementi che, apparentemente diversi, identificano la stessa cosa, dunque nel caso a lato bisognerebbe avere corrispondenza tra gli elementi A1-A2, B1-B2 e D1-D2; infine ritornare un dataset contenente un identificatore unico per ogni elemento (il dataset output dell'esempio sarà composto da A,B,C,D).
Il modo in cui SparkER implementa tali algoritmi parte dall'unificare i dataset su cui si intende lavorare. Si può però notare che il primo modo che viene in mente per confrontare i vari elementi dei dataset è confrontarli uno per uno (n x m numero di confronti, con n cardinalità del primo dataset e m cardinalità del secondo), dunque bisognerebbe avere un modo per semplificare tale processo, che non scala bene nell'ambito dei big data.

### 2.3.3 Blocking

SparkER fa uso di tecniche di blocking e meta-blocking per ridurre il numero di confronti che bisogna fare per produrre risultati. Il blocking si basa sul raggruppare i profili di entità simili nello stesso blocco, assegnando a ogni profilo un insieme di keys, in questo modo profili con la stessa key vengono raggruppati nello stesso blocco. Utilizzando i blocchi, i confronti vengono fatti all'interno dei blocchi, riducendo così il numero di confronti alla grandezza del blocco e non alla grandezza del dataset. Inoltre questo apre anche alla possibilità di parallelizzare questo task, confrontando più blocchi contemporaneamente (si ricorda lo SparkContext citato in precedenza, che SparkER usa per parallelizzare i task).
Il blocking si basa su alcune ipotesi:

- ogni profilo è identificato da un insieme di coppie nome-valore unici;
- ogni profilo identifica corrisponde a un oggetto o persona nella realtà;
- la corrispondenza tra due profili può esserci solo se i due profili condividono almeno un blocco;
- ogni profilo è rappresentato da almeno un blocco;
- tutti i profili che condividono una stessa key vengono messi nello stesso blocco.

Obiettivo del blocking è massimizzare le misure di richiamo e precisione, dove il richiamo è il rapporto tra match trovati e match esistenti, mentre la precisione è il rapporto tra match trovati e i confronti eseguiti (si tende a dare più rilevanza al richiamo). I risultati di questa fase possono essere soddisfacenti, ma risultano essere ulteriormente migliorabili in termini di precisione.

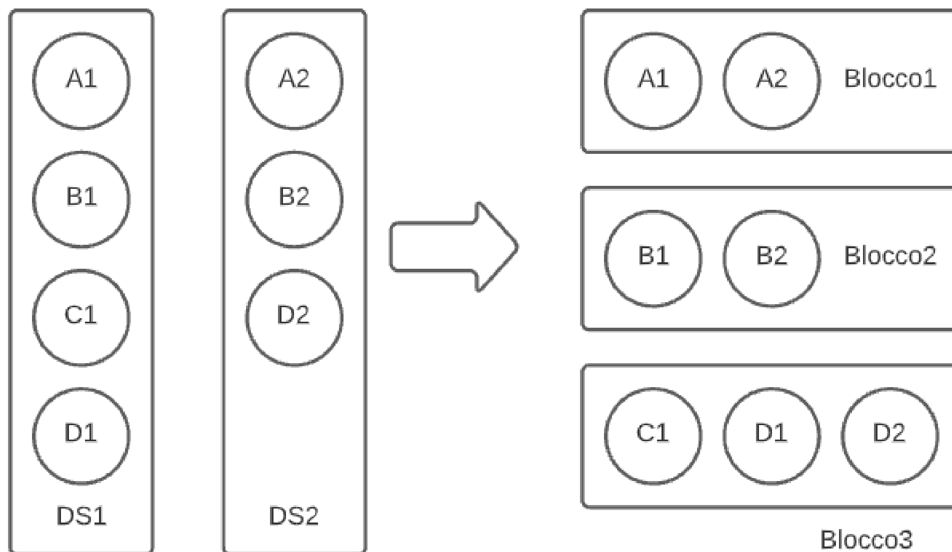


Figura 3: Creazione dei Blocchi

### 2.3.4 Token Blocking

SparkER implementa la tokenizzazione dei profili per creare i blocchi (non è l'unico metodo come si vedrà in seguito). Fornito in input il profilo di un entità, si estraggono tutti i token dai valori dei suoi attributi e, per ogni token viene creato un blocco; tale blocco sarà poi popolato dai profili che risultano generare lo stesso token (per essere creato un blocco deve contenere almeno due elementi).

SparkER fornisce due tipi di tokenizzazione: il word tokenization, ovvero divide il testo di un attributo nelle parole che lo compongono, quindi eliminando la punteggiatura e i white spaces, risultando molto semplice come metodo; il q-grams tokenization, che divide il testo in frammenti di lunghezza q. Il primo metodo è utile quando si vogliono confrontare ad esempio documenti di dimensioni elevate, ma non rileva gli errori nelle parole e in questo caso creerebbe un blocco aggiuntivo. Il secondo metodo funziona meglio nelle situazioni opposte, è in grado di creare blocchi con profili che hanno dato token simili nonostante gli errori, ma è parecchio pesante nel caso si confrontino documenti di grandi dimensioni, poichè verrebbero generati troppi token.

Dato un criterio di similarità (ovvero una misura di quanto sono simili due token o profili, per citarne una la similarità di Jaccard), viene posto un valore soglia e in base al risultato della similarità, due profili vengono tenuti per la creazione di un blocco attraverso quel token solo se soddisfano quel valore soglia.

### 2.3.5 Block Cleaning

Ulteriore step per il miglioramento delle prestazioni si ha attraverso il block-cleaning. Con questo termine si identificano tutte quei metodi che riducono il numero di blocchi che sono stati creati, per diminuire ulteriormente il numero di confronti.

Fa parte di questo gruppo il block purging, che consiste nel mettere un limite superiore alla cardinalità di un blocco (ovvero il numero di confronti che andrebbe fatto), permettendo così che i blocchi troppo grandi vengano eliminati; di solito questi blocchi sono formati da stop-words o da token così comuni che risultano essere inutili.

Altro metodo è il block filtering, che si basa sull'idea che più elementi contiene un blocco generato da un token, meno probabile è che quel blocco contenga dei duplicati unici; esso riordina i blocchi in ordine del numero di elementi, e scarta una certa percentuale dei blocchi più grandi (solitamente viene mantenuto l'80% dei blocchi).

A livello implementativo su SparkER, le funzioni che compiono queste azioni regolano l'aggressività dell'eliminazione usando un parametro numerico; nel caso del purging il parametro deve essere maggiore di 1 (più il valore tende a 1, maggiore sarà l'aggressività, tipicamente si usa 1.005), mentre per il filtering il parametro dev'essere compreso, esclusi gli estremi, tra 0 e 1 (corrisponde alla percentuale di blocchi da mantenere, tipicamente 0.8).

### 2.3.6 Meta-Blocking

Ulteriore tecnica utilizzata dalla libreria è quella del meta-blocking, che si basa sull'idea che più blocchi due profili condividono, più è probabile che facciano match; l'obiettivo è che dato un blocco B, lo si ristruttura in un blocco B' che mantenga circa lo stesso richiamo, ma migliorandone la precisione.

L'esempio proposto mostra il passaggio che viene fatto dalla fase di blocking a quella di meta-blocking.

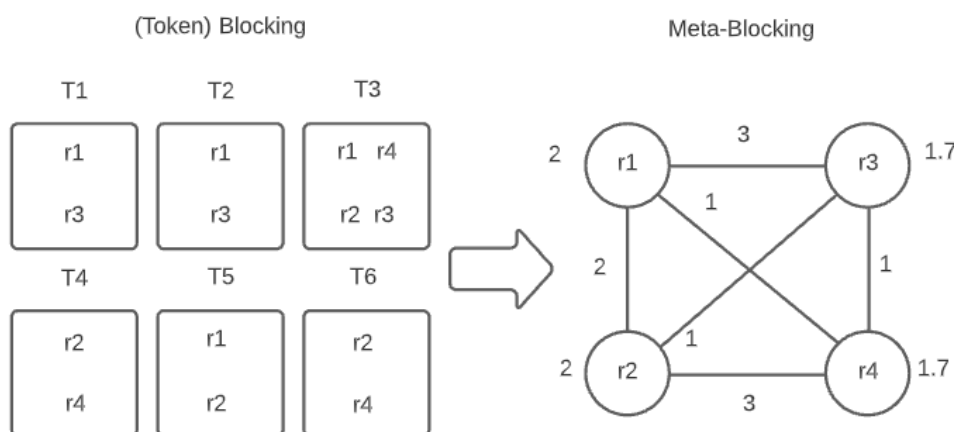


Figura 4: Meta-Blocking

Si supponga che le coppie di record r1-r3 e r2-r4 indentifichino la stessa entità. Per ogni token T ricavato dai record r viene creato un apposito blocco T*, contenente gli r* record che l'hanno generato; per ogni record r* viene creato un nodo nel grafo del meta-blocking, dove questi nodi sono tutti interconnessi attraverso archi il cui peso corrisponde al numero di blocchi che tali record condividono. Ogni nodo ha poi un valore dato dalla media degli archi uscenti (nel caso di r1 si ha (1+2+3)/3=6). Un arco viene tenuto solo se il suo peso è maggiore o uguale al peso dei due nodi che collega. Si può notare come questo permetta di eliminare le false corrispondenze che il token blocking può creare, tuttavia si può anche notare come questo metodo non sia perfetto; secondo le ipotesi che sono state fatte, r1 ed r2 non si riferiscono alla stessa entità, ma avendo il peso dell'arco uguale al peso dei due nodi risultano erroneamente un possibile match.

### 2.3.7 Entity matching e data fusion

Il risultato finale del meta-blocking fornisce delle coppie tra profili, tuttavia lo step usato non fornisce delle coppie sicuramente corrispondenti, ma solo dei possibili candidati. L'entity matching è necessario dunque per stabilire se tali coppie formano una corrispondenza reale oppure un falso positivo.

Il risultato di questo step è un insieme di coppie di profili identificati come match reali e dunque che si riferiscono alla stessa entità.
In seguito, attraverso l'entity clustering, le coppie vengono spartite in vari gruppi, in base all'entità a cui si riferiscono; un esempio di divisione in questo caso si può avere col connected components, uno dei più semplici, che consiste nel creare un grafo con tutte le coppie per poi partizionarlo in base a come sono connessi i nodi (due nodi rimangono nella stessa partizione se sono connessi attraverso un arco).
Ultimo passaggio di questa lunga serie di step consiste nel creare un unico record da ogni raggruppamento che rappresenta ogni entità, per poi unire tutti i record in un unica lista. Il data fusion si occupa di questo; per fare ciò vengono usate delle funzioni aggregative per ogni attributo. Ad esempio, nel caso si tratti del nome di un'entità, si può usare un criterio di maggioranza, dove il nome con l'occorrenza più alta viene scelto come nome nel record finale, oppure nel caso si tratti di un prezzo, si può calcolare la media dei valori di tutti i prezzi contenuti nei profili dell'entità.

### 2.3.8 BLAST

Un significativo miglioramento alla parte di blocking/meta-blocking a livello prestazionale si può avere attraverso BLAST. BLAST (Blocking with Loosely-Aware Schema Techniques) utilizza un approccio ibrido tra meta-blocking e tecniche loosely schema-aware. L'intuizione su cui si base è che se due attributi sono simili, allora anche i loro valori dovranno essere simili. Quindi informazioni importanti possono essere estratte anche da fonti di dati altamente eterogenee.
BLAST agisce raggruppando, prima del blocking, gli attributi derivanti dai token che vengono considerati "simili" in dei cluster, mentre i restanti vengono inseriti in un cluster generale. Per esempio se in due profili diversi nell'attributo città di uno è contenuto il token Roma e nell'attributo indirizzo dell'altro è contenuto sempre il token Roma (come in Piazza Roma), viene presa in considerazione l'appartenenza a due attributi diversi dei token che risultano uguali e di conseguenza verranno inseriti in cluster diversi, cosa che con il normale token blocking non avviene.
Considerata l'idea che alcuni attributi abbiano più informazioni di altri, si può utilizzare una misura dell'informazione per generare delle blocking keys molto più precise. BLAST utilizza l'entropia di Shannon per calcolare tale contenuto informativo e utilizza l'entropia dei vari attributi per attribuire un valore di entropia generale per ogni cluster, sommando le entropie dei vari attributi contenuti nel cluster; un cluster con un'entropia alta indicherà la presenza di attributi molto diversi tra loro e un match in uno di questi avrà un valore più alto rispetto a un match in un cluster a bassa entropia. Questi step verranno poi usati per creare dei blocchi più precisi e nel meta-blocking si terrà in considerazione l'entropia dei cluster e dei blocchi per generare dei pesi più precisi; facendo riferimento alla situazione di Figura 4, utilizzando l'entropia l'arco tra i nodi r1 e r2 verrebbe correttamente scartato, poichè il valore dei nodi sarebbe diverso.

### 2.3.9 Esempio real-life

Facciamo un esempio che renda chiaro il problema che può capitare in alcuni contesti lavorativi, come in aziende e imprese. Supponiamo che un'azienda di logistica che lavora con la gestione di prodotti medici acquisisca nuovi clienti; questi clienti hanno già il loro catalogo di prodotti, che ne comprende a migliaia. In ogni catalogo lo stesso prodotto è salvato con un identificatore diverso, e una descrizione simile ma non uguale. L'azienda vorrebbe unificare i due cataloghi dando un identificatore unico ai prodotti, in modo da poter organizzare al meglio il magazzino.
Catalogo A:

| ID | Nome | Descrizione |
|---|---|---|
| 1 | Siringhe 10ml | Siringhe 10ml x 10p |
| 2 | Siringhe 10ml formato extra | Siringhe 10ml x 100p |
| 3 | Antibiotici 10g | AB curamax tonsille 10g x 20p |
| ... | ... | ... |

Catalogo B:

| PID | Nome | Descrizione |
|---|---|---|
| P123X | Siringhe 10x10 | Siringhe 10 ml - 10 pezzi |
| P123Y | Siringhe 10x100 | Siringhe 10ml - 100 pezzi |
| P456A | AB CuraMax 10x20 | Antibiotici per tonsillite 10g - 20 pezzi |
| ... | ... | ... |

Per risolvere questo problema si può agire in due modi.
Il primo prevede l'unificazione manuale dei due cataloghi; questo metodo fa già intuire di essere una soluzione alquanto scomoda in quanto: il numero di persone che ci lavorerebbe non sarebbe irrilevante (utilizzo di personale non indifferente); essendo l'uomo propenso a commettere errori, probabilmente il catalogo finale conterrà degli errori (alta probabilità di errori); il lavoro richiederebbe comunque molto tempo.
Il secondo modo prevede invece un approccio ibrido tra uomo e calcolatore, utilizzando uno strumento di deduplication che permetta di sfruttare le descrizioni dei vari oggetti per riconoscere l'uguaglianza tra i vari oggetti. I benefici di questo approccio dovrebbero essere già evidenti: è più veloce, più preciso e richiede molte meno persone per il lavoro.

## 2.4 Workflow di SparkER

In questa piccola sezione si elencheranno a grandi linee i vari step che, dati in input i dataset, portano come risultato un elenco di profili senza duplicati:

- si prendono in input uno o più dataset(a seconda di se bisogna fare data deduplication o linkage) e si crea una lista di profili (si uniscono le liste dei profili qualora ci siano due dataset)
- se si vuole utilizzare BLAST, si creano i cluster degli attributi;
- si creano i blocchi, utilizzando i profili e i cluster(se in BLAST);
- si effettua block cleaning con le tecniche sopra citate;
- si effettua il meta-blocking, utilizzando uno degli algoritmi di pruning possibili.

## 2.5 Mancanze della libreria

SparkER non è esattamente una libreria molto intuitiva da utilizzare, complice la presenza di molti metodi e la mancanza di una documentazione apposita di questi metodi.
SparkER non è "casual-user friendly", i metodi della libreria richiedono spesso molti parametri; il workflow di SparkER prevede una serie di step ben precisi e la varietà che viene proposta per eseguire ognuno di questi può causare confusione; manca un metodo che permetta, dati uno o due dataset, di eseguire senza ulteriori comandi l'intero processo di blocking/meta-blocking.
SparkER presenta anche delle limitazioni dei formati di dataset compatibili per il caricamento dei profili, sono presenti il formato .csv e .json, ma se un utente volesse caricare un dataset da un file excel o da una query sql dovrebbe scrivere lui il codice che permetta questa fase preliminare.
La libreria inoltre non è presente su PyPi; PyPi è un repository (contenitore) online contenente un grandissimo numero di librerie open-source python e consente l'installazione attraverso il gestore di pacchetti pip (comando su shell di python "pip install <nome libreria>"); chiunque volesse installare SparkER dovrebbe scaricarne il repository da GitHub e farne l'installazione manuale, processo che può essere tedioso.

# 3 Aggiunta di funzionalità alla libreria

In questa sezione si mostrerà cosa è stato aggiunto nella libreria al fine di ampliarne le funzionalità e di migliorarne le già esistenti.

## 3.1 Aggiunta di nuovi formati

Una delle parti cruciali del workflow della libreria è il caricamento dei profili. SparkER per utilizzare i propri metodi ha bisogno che il contenuto dei dataset su cui si vuole effettuare data linkage/deduplication venga inserito in un RDD e che tale contenuto sia convertito in tanti profili; i metodi che effettuano questa operazione vengono detti wrappers.

SparkER implementa wrappers per 3 tipi di formato: json, csv, pandas. Per i formati json e csv, la funzione prende in input direttamente il file del dataset, mentre il wrapper di pandas funziona solo se viene fornito in input un dataframe di pandas, ovvero un oggetto definito nella libreria di pandas. Questi wrappers effettuano il parsing dei dataset che gli vengono forniti e restituiscono un RDD di profili, che può essere utilizzato negli step successivi di blocking e block-cleaning.

Esempio di wrapper di dataset in formato csv:

```
import sparker as sp
profiles = sp.CSVWrapper.load_profiles('dataset_da_caricare.csv',
                                       real_id_field = "id")
```

Nonostante i formati json e csv siano tra i più usati per i dataset, alcuni utenti potrebbero voler leggere dati da altri tipi di formato. La presenza di un wrapper che lavori anche con pandas aiuta in questo senso, data la flessibilità di tale libreria, ma è comunque un procedimento aggiuntivo che un utente deve preparare a parte prima di poter creare le strutture per SparkER.

### 3.1.1 Creare dataframe da formato excel/xlsx

Si è deciso di fornire una funzione ausiliaria che permetta il caricamento di dataset dal formato excel (xlsx) in un dataframe di pandas. Il motivo per cui si è deciso di non realizzare un wrapper esclusivo per il formato xlsx è dato dal fatto che un dataset in tale formato ha un limite sulla dimensione abbastanza basso (non raggiunge un gigabyte). Essendo già presente un wrapper per i dataframe di pandas, l'unica cosa che va fatta è utilizzare la funzione già presente su pandas per leggere gli xlsx.

Il problema di tale funzione è data dal parametro engine; fornendo in input semplicemente il file xlsx, la funzione darà errore. Questo è dovuto al fatto che pandas utilizza di default l'engine "xlrd" per leggere gli excel, che da alcuni aggiornamenti non supporta più il formato xlsx ma solo xls. Per risolvere il problema, si è scelto di utilizzare un altro engine: "openpyxl", un engine open-source.

La funzione è stata implementata come segue:

```
import pandas
class PandasDFMaker(object):
    def xlsx_df(xlsx_path):
        pandas_df = pandas.read_excel(xlsx_path, engine='openpyxl')
        return pandas_df
```

Utilizzando il metodo di SparkER, un utente generico non deve preoccuparsi di trovare l'engine giusto e può ottenere direttamente il dataframe che gli serve.

### 3.1.2 Creare dataframe da query sql

Approccio simile è stato utilizzato anche nel prendere i risultati di una query sql e inserirli in un dataframe. In questo caso il passaggio aggiuntivo rispetto alla funzione precedente è quello di stabilire una connessione con il dbms che si utilizza. Si è deciso di utilizzare la libreria "sqlalchemy" per gestire tutti i passaggi di collegamento, sottomissione query e chiusura della connessione.

```
import pandas
import sqlalchemy
class PandasDFMaker(object):
    def dbms_df(host, database, user, password, port="5432", query=""):
        if(host==None or database==None or
            user==None or password==None):
                raise ValueError("...")
        if (query == ""):
                raise ValueError("...")
        try:
            engine = sqlalchemy.create_engine(
                                    'postgresql://'+user
                                    +':'+password+'@'+host+
                                    ':'+port+'/'+database)
            conn=engine.connect()
            pdf=pandas.read_sql(query, conn)
        except (Exception) as error:
            print(error)
        finally:
            if conn is not None:
                conn.close()
        return pdf
```

La funzione riportata è stata modificata rispetto all'originale implementata su SparkER; questa versione mostra la gestione della connessione con il dbms postgresql e la creazione del dataframe

direttamente dalla query che si passa come parametro. La funzione implementata su SparkER consente di utilizzare diversi dbms, utilizzando un parametro aggiuntivo "dialect" che specifica quale dbms l'utente vuole usare. Si è ritenuto di mettere solo la versione troncata della funzione, dato che la maggior parte del codice aggiuntivo sarebbe stato composto da controlli ripetitivi e non molto estetici.

### 3.1.3 Vantaggi dell'usare pandas

La scelta di pandas per la realizzazione delle funzioni viste in precedenza non è stata casuale. Pandas presenta molti vantaggi per i programmatori python per la gestione di grandi quantità di dati in modo abbastanza veloce e visivamente gradevole.
La flessibilità di questo framework quando si tratta di raccogliere dati da formati diversi è di grande aiuto, permettendo di non dover scrivere metodi ad-hoc che possono rubare tempo e righe di codice, e la rappresentazione di tali dataset può essere cambiata a seconda delle cose che si vogliono evidenziare; pandas inoltre permette il caricamento di larghi dataset in modo efficiente, permettendone quindi l'utilizzo in modo efficiente anche con calcolatori meno potenti; infine, questa libreria è stata scritta in python, dunque può utilizzare al meglio tutte le funzionalità di python e sinergizza con tutto il codice che è stato scritto per SparkER.
Come svantaggio (nell'utilizzo previsto con SparkER), pandas presenta un limite alla dimensione del proprio dataframe (50 GB), ma per SparkER non è un problema, dato che il formato xlsx ha una dimensione massima molta più bassa e il dataframe di sql non è pensato per giganteschi dataset, che verrebbero piuttosto trattati come csv.

```
xlsx_path="D:\Downloads\water-quality-performance-results-end-of-january-2022.xlsx"
pdf=sparker.PandasDFMaker.xlsx_df(xlsx_path)
pdf
```

| | Region Name | System Name | Parameter | Units | Health Guideline * | Aesthetic Guideline * | Average Value | C |
|---|---|---|---|---|---|---|---|---|
| **0** | Eyre | Coffin Bay | Alkalinity | mg/L | NG | NG | 237 | |
| **1** | Eyre | Coffin Bay | Aluminium - acid soluble | mg/L | NG | ≤ 0.2 | <0.001 | |
| **2** | Eyre | Coffin Bay | Antimony | mg/L | ≤ 0.003 | NG | <0.0005 | |
| **3** | Eyre | Coffin Bay | Arsenic | mg/L | ≤ 0.01 | NG | 0.0008 | |
| **4** | Eyre | Coffin Bay | Barium | mg/L | ≤ 2 | NG | 0.0167 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |

Figura 5: Rappresentazione dataset excel con pandas

## 3.2 Metodo facciata

Come già detto in precedenza, SparkER richiede molti passaggi per effettuare blocking e meta-blocking. Questo è scomodo sia per utenti non esperti, che rischiano di dimenticare step fondamentali per il successo dell'operazione o il passaggio di alcuni parametri che regolano le varie funzioni, ma anche per chi utilizza SparkER da più tempo, che non vorrebbe sempre dover riscrivere le funzioni per fare gli stessi passaggi con diversi dataset.

Si è deciso di implementare un metodo basato sul design pattern facade (facciata); l'idea alla base è quella di racchiudere all'interno di un unico metodo le varie chiamate alle funzioni che vengono utilizzate per eseguire tutto il workflow. Così facendo l'utente non deve necessariamente conoscere tutte le funzioni che vengono chiamate, ma semplicemente passare in input uno o due file di dataset e alcuni parametri, tra cui alcuni opzionali (per gli utenti che sanno cosa succede in certi passaggi e vogliono usare ad esempio un algoritmo di pruning diverso) e ottenere come risultato la lista di profili su cui è stato fatto blocking/meta-blocking, il risultati del meta-blocking e le statistiche dell'ultimo step effettuato.

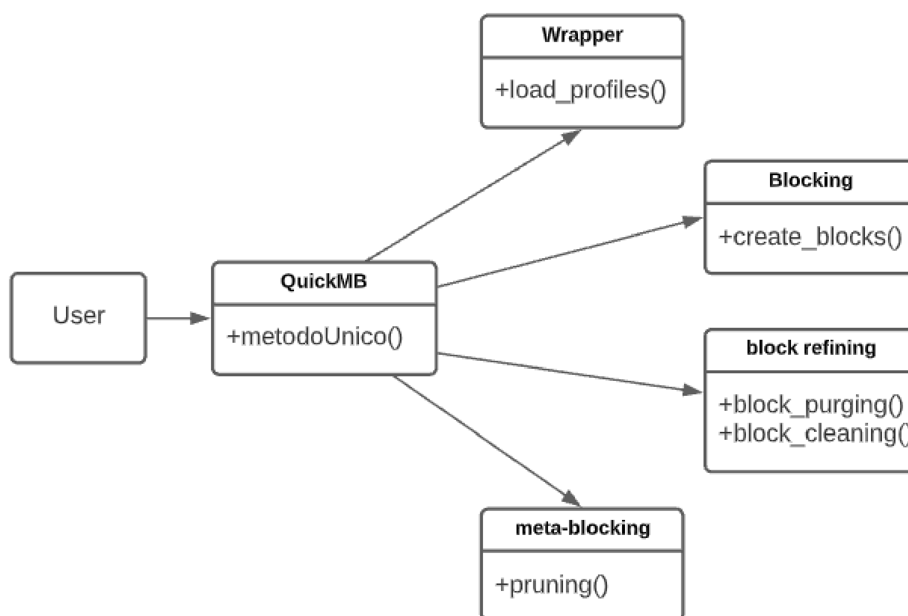


Figura 6: Schema semplificato del pattern

Ne sono state implementate 3 versioni diverse:

- data deduplication, che prende in input un singolo dataset;
- data linkage normale, che prende in input due dataset;
- blast, un metodo specifico per il workflow di BLAST.

Non verrà mostrato l'intero codice di ogni funzione, dato che si tratta solo di un insieme di varie chiamate a funzione, ma verrà mostrata solo la dichiarazione dei tre metodi.

```
def blast(profiles1_fp, profiles2_fp, real_id_f,
          groundtruth_fp=None, file_extension="json",
          chi_divider=2.0)
def data_deduplication(profiles_fp, real_id_f,
                       groundtruth_fp=None, file_extension="csv",
                       weighting_type=WeightTypes.CBS)
def standard_metablocking(profiles1_fp, profiles2_fp, real_id_f,
                          groundtruth_fp=None, file_extension="json",
                          weighting_type=WeightTypes.CBS,
                          mb_schema="wnp",
                          comp_type=ComparisonTypes.AND)
```

Partendo dai parametri comuni a tutte e tre le funzioni:

- profiles-fp, il percorso del file contenente il/i dataset;
- real-id-f, campo contenente l'identificatore del record;
- groundtruth-fp, il percorso file che contiene gli id corrispondenti alle entrate dei dataset una volta che esse vengono convertite da Spark (parametro opzionale, serve per controllare richiamo e precisione dopo ogni step di blocking);
- file-extension, il tipo di formato del dataset.

Nel caso delle funzioni riguardanti BLAST e il data deduplication, si può notare che sono le due con meno flessibilità, lasciando solo la possibilità di decidere il valore del parametro chi-divider nella prima, che regola l'aggressività dell'algoritmo di pruning usando uno schema basato sul chi quadro, e il weighting type della seconda, che lascia la possibilità di scegliere il metodo di pesatura degli archi durante lo step del pruning. La terza funzione invece è quella che lascia più spazio alle personalizzazioni dell'utente, consentendo di modificare oltre al weighting type, anche l'algoritmo usato per il pruning mb-schema (che possono essere weighted node/edge pruning, "wnp" e "wep", oppure cardinality node/edge pruning, "cnp "e cep") e il comparison type (serve per scegliere se usare il weighted/cardinality node pruning o il reciprocal weighted/cardinality node pruning). C'è da sottolineare però che tali funzioni sono limitate dal non poter regolare le misure usate per i vari step di blocking e block cleaning, visto che le tre funzioni sopra mostrate hanno dei valori di default con cui operare durante quegli step. Dunque queste funzioni sono ottime per risparmiare tempo, per gli utenti che non hanno compreso bene il workflow di SparkER ma che necessitano di utilizzare tali metodi e per quelli che non hanno bisogno di utilizzare valori specifici in certi step. D'altra parte, qualora si volesse andare a provare valori diversi, visto che valori diversi talvolta possono dare risultati diversi, tra cui anche prestazioni migliori in termini di richiamo e precisione, allora si consiglia di effettuare in proprio tutti gli step necessari.

# 4 Documentazione della libreria

Nella seguente sezione si approfondirà come è stata scritta la documentazione della libreria, spiegando cosa è stato utilizzato nella sua realizzazione, come si è deciso di strutturare le varie sezioni della documentazione e come è stata poi pubblicata.

## 4.1 Strumenti utilizzati

### 4.1.1 ReadTheDocs

ReadTheDocs (rtd) è un servizio di hosting gratuito che permette in modo semplice la pubblicazione online delle documentazioni dei software o delle librerie open source. Il servizio consente ai vari programmatori la possibilità di superare le limitazioni dovute alla necessità di un servizio di hosting.
Rtd utilizza GitHub, BitBucket o GitLab come fonte per caricare il progetto e, in seguito, il servizio compila automaticamente il progetto. In questo modo, qualora venisse modificato il progetto sul proprio repository, non servirebbe alcun tipo di cambiamento su ReadTheDocs che automaticamente caricherebbe la build più recente, dando anche la possibilità di caricare build differenti in base alla versione del progetto.
Inoltre, il servizio consente anche di scaricare la documentazione in formato pdf o eBook, qualora si volesse scaricare in formato portabile la documentazione.
Per il servizio di archiviazione del progetto si è scelto di utilizzare GitHub.

### 4.1.2 Sphinx e reStructuredText

Sphinx è uno strumento che permette la realizzazione in modo semplice di una documentazione con anche funzionalità che normalmente richiedono più di qualche riga di codice per essere implementate in un documento online.
Sphinx permette di generare in automatico la rappresentazione html della documentazione, il formato pdf, ePub, ma anche il manuale o la semplice rappresentazione testuale.
Sphinx, a livello implementativo, consente la facile generazione di indici e la gestione di una struttura ad albero delle varie pagine della documentazione; inoltre consente l'inclusione di frammenti di codice di vari linguaggi. Sphinx utilizza reStructuredText (rst) come linguaggio di markup. rst è un linguaggio che segue la filosofia what-you-see-is-what-you-get, molto semplice da leggere e da implementare; il parser di rst è contenuto nel pacchetto Docutils di python, disponibile dall'installazione di default di python.

### 4.1.3 Sintassi e costrutti utilizzati

A livello di programmazione, la sintassi è molto intuitiva, la maggior parte dei costrutti di rst sono chiamati in modo molto intuitivo rispetto a quello che fanno. Nella documentazione si è fatto un uso esteso di alcuni di questi.

Rst utilizza i caratteri "+" e i caratteri "-" per dividere una pagina nelle varie sezioni, tali caratteri vanno messi sotto ai titoli delle varie sezioni, il carattere "+" è usato per il titolo dell'intera pagina mentre il "-" è usato per le sezioni.

Per il titolo della documentazione invece viene usato il carattere "=", e va scritto all'interno del file index.rst, ovvero il file che rappresenta la pagina principale quando si accede alla documentazione. Sono presenti ulteriori caratteri per dividere ulteriormente in sottosezioni e paragrafi, ma non sono stati utilizzati nella documentazione. Tale divisione nella pagina consente ad altre pagine della documentazione di fare riferimento esattamente alla sezione.

Il costrutto che è stato utilizzato per creare l'indice (Figura 10 a lato) è il costrutto "..toc-tree:", questo va inserito sempre nel file index.rst; nella documentazione è stato utilizzata l'opzione ":hidden:", che non inserisce l'indice anche nel corpo della pagina, ma solo a lato.

(a) Label

(b) Toc-Tree

Figura 7: Costrutti di sphinx

Il costrutto per mostrare i frammenti di codice di SparkER è "..code-block:"; per usare code-block basta semplicemente scrivere il codice al suo interno (rispettando l'indentazione giusta), l'output che ne risulterà è molto simile al risultato che si ottiene su jupyter notebook usando markdown. Di default il sphinx rileva in automatico il codice python, e provvede dunque ad evidenziarne la sintassi (ad esempio evidenziando il comando import in verde), qualora però si volesse evidenziare un altro tipo di linguaggio basta aggiungere il nome di tale linguaggio a lato del costrutto ("..codeblock: javascript").

```
.. code-block::

    print("Number of blocks",
```

(a) Costrutto

```
print("Number of blocks",blocks.co
```

(b) Toc-Tree

Figura 8: Implementazione di code-block

Sono state inserite anche parecchie references a varie sezioni della documentazione (ad esempio nella sezione BLAST parlando di un metodo presente anche nella sezione del meta-blocking si rimanda alla sezione della seconda) attraverso il costrutto
”:doc:‘linkword <percorso/della/sezione>‘”. Per gli hyperlink invece il costrutto da utilizzare è ”‘linkword <link-al-sito>‘_”, ed è stato utilizzato molto nella prima sezione, dove sono contenuti i link per scaricare tutto il necessario per installare SparkER.
Sphinx permette in diversi modi di implementare le tabelle, il costrutto che in questo caso si è scelto è il ”..csv-table:”; rispetto agli altri costrutti, csv-table è molto più comodo da implementare, dato che in alcuni degli altri metodi bisognerebbe, quasi letteralmente, disegnare la tabella usando i caratteri ”=” e ”—”; in questo modo invece si può facilmente impostare il nome delle colonne e la larghezza delle colonne, rispettivamente con ”:header:” e ”:widths:”, mentre per popolare la tabella basta mettere il contenuto di ogni casella nella riga separato da una virgola.

```
.. csv-table:: Catalog B
    :header: ID,Name,Description
    :widths: 5,15,20

    1,Syringes 10 ml,Syringe 10 ml
    2,Syringes 10 ml big pack,Syri
    3,Small needle,Needle for insu
    ...,...,...
```

(a) Costrutto

*Catalog B*

| ID | Name | Description |
| --- | --- | --- |
| 1 | Syringes 10 ml | Syringe 10 ml x 10 pieces |
| 2 | Syringes 10 ml big pack | Syringe 10 ml x 100 pieces |
| 3 | Small needle | Needle for insulin 4 mm 10 pieces |
| ... | ... | ... |

(b) Output

Figura 9: Implementazione di csv-table

## 4.2 Struttura della documentazione

La documentazione è stata redatta in lingua inglese ed è divisa in 5 macro-sezioni.
La prima sezione, Getting Started, affronta i passi preliminari per l’utilizzo della libreria. Vengono elencati i vari pre-requisiti software, la dipendenza della libreria da python, java, Spark e git, e la spiegazione per l’installazione di SparkER. Inoltre sono stati inseriti link ai vari articoli che spiegano i vari studi teorici su cui si basa SparkER.

La seconda sezione, Scope, fornisce alcuni accenni di teoria sull'entity resolution e il data deduplication, non troppo estesi data la presenza delle reference agli articoli nella prima sezione e siccome l'obiettivo della documentazione è soprattutto la spiegazione dell'utilizzo dei metodi si è deciso di non dilungarsi troppo sulla seconda; viene anche riproposto l'esempio riportato nella sezione 2.3.9 dell'elaborato.

SparkER
latest
Search docs

GETTING STARTED:
Installation and Requirements
External References

SCOPE
Understanding the Purpose
Real life example

CODE WITH SPARKER
Quick Application
Loading Profiles
Blocking
What's a block?
Creating blocks
Block Purging and Cleaning
Checking performances (Optional)
Example data linkage - Part 2
Meta-Blocking

BLAST
A Different Approach to Entity Resolution
How to use BLAST

and you can understand that, if we are working with peta/exabytes of data, ER doesn't scale very well. But if we can only reduce the comparisons for the number of profiles in every block, then the comparisons number is much much lower, and since comparing can't take forever since "time is money", blocking is a really useful tool.

## Creating blocks

SparkER offers different techniques for blocks creation, we're going to show them all, so you can use which ever you prefer; they may differ in performance, but that's up to the dataset composition or other factors.

```
blocks = sparker.Blocking.create_blocks(profiles, separator_ids=None, keys_to_exclude=None,
                attributes_to_exclude=None,
                blocking_method=sparker.BlockingKeysStrategies.token_blocking
                )
```

The function above takes as input the set of profiles obtained as a result of the loading profiles step, the separator ids (you may or may not have to input it, depends if you are doing deduplication or linkage),some optional parameters to filter keys or attributes you want to bypass and returns a set of blocks, containing profiles that share the same keys. This is a standard token blocking strategy, is general purpose and the easiest and quickest to use, but the *blocking_method* parameter is the one that you can change to use a different blocking strategy.

Other strategies are:

- `BlockingKeysStrategies.token_blocking_w_attr`
- `BlockingKeysStrategies.ngrams_blocking`

While both strategies share the same parameters as the above method, n-gram blocking actually require an additional parameter `ngram_size`, which gives the size of the n-grams used for the strategy, by default it's set to 3.

Figura 10: Estratto della documentazione su ReadTheDocs

La terza sezione, Coding, è la sezione più corposa delle cinque. La parte iniziale di questa sezione si dedica a dare le definizioni di dataset clean e dirty e di profili e introduce i metodi visti nella sezione 3.2 per il data deduplication e il data linkage normale. Dopodichè si affrontano tutte le funzioni del workflow di SparkER. Si introduce il caricamento dei profili, con l'inizializzazione delle variabili che servono per i vari step di blocking. I metodi di blocking sono descritti tutti in maniera esaustiva, lasciando all'utente la possibilità di scegliere quello che più reputa utile in tale situazione. Infine la parte sul meta-blocking mostra i vari algoritmi di pruning e le strutture di Spark che devono essere inizializzate.
La quarta sezione, BLAST, è una sezione che si focalizza esclusivamente su BLAST; viene spiegata l'idea alla base di BLAST, l'utilizzo dei cluster, l'entropia e i vari metodi per compiere i passaggi specifici di BLAST.
L'ultima sezione, Future Updates, elenca le varie feature che saranno implementate in futuro o che potrebbero essere implementate.

# 5 Aggiunta della libreria su PyPI

Nella seguente sezione si spiegherà cos'è PyPI, la sua utilità e cosa è stato fatto per caricare SparkER su quest'ultimo.

## 5.1 Perchè usare PyPI

Python Package Index, in breve PyPI, è un repository di software per Python. L'intento del repository è di rendere facile l'installazione di varie librerie open-source sviluppate per python (la quasi totalità delle librerie più utilizzate per python lo utilizza). PyPI utilizza "pip" come gestore delle installazioni dei pacchetti; tutti i programmatori python che hanno avuto necessità di installare una libreria hanno usato almeno una volta il comando "pip install nome-libreria".
Pubblicare la propria libreria sul repository ha molti vantaggi sia per chi la carica che per gli utenti che la utilizzeranno; per installare una libreria senza pip bisognerebbe andare su GitHub e clonare il repository sul proprio terminale, e in seguito utilizzare il file d'installazione direttamente dal repository scaricato, mentre utilizzando pip basta il semplice comando menzionato in precedenza e la libreria può essere importata. Inoltre alcuni step per caricare la libreria richiedono di modificare il pacchetto della propria libreria in modo da renderlo pronto per la pubblicazione; sapere come rendere il proprio codice publish-ready è importante nel caso si lavori su una libreria open-source e si debba "standardizzare" il pacchetto rispetto alle librerie pubblicate su PyPI.

## 5.2 Creazione del package

Come primo passo, va effettuata la registrazione sul sito di PyPI e, nel caso in cui pip non sia presente sulla propria installazione di Python, va scaricato e installato; le versioni più recenti di python hanno preinstallato pip di default, quindi nella maggior parte dei casi si può ignorare questo step.
La parte più lunga della procedura riguarda la creazione del pacchetto che verrà poi caricato. Anche in questo caso, va caricato il proprio software su GitHub, dato che PyPI prenderà il sorgente da li. Il nome del package dev'essere lo stesso del repository su GitHub.
La libreria dev'essere strutturata in un certo modo; bisogna creare una directory al cui interno vanno inseriti i file ".py" contenti le varie classi utilizzate. In questa directory va creato un file chiamato esattamente "__init__.py"; questo file conterrà una serie di import delle varie classi della libreria. Questo segna le classi alle quali si vuole consentire l'accesso all'utente.
Di seguito un esempio degli import contenuti nel file:

```
from .classedesempio1 import classe1, classe2
from .classedesempio2 import classe3, classe4
```

Una volta strutturata la libreria in questo modo, vanno creati tre file, due obbligatori e uno opzionale.
Il primo e più importante è il "setup.py":

```python
from setuptools import setup
setup(
    name = 'libreriaesempio',
    version = '1.0',
    description = 'Spiegazione del setup.py',
    author = 'Pinco Autore',
    author_email = 'pinco@unimore.it',
    url = 'https://github.com/autpinco/libreriaesempio',
    packages = ['libreriaesempio'],
    install_requires = ['numpy', 'networkx'],
)
```

Il contenuto delle variabili name, description, author, author_email e packages sono abbastanza intuitive; la variabile version contiene il numero della release del pacchetto, quindi tutte le volte che il pacchetto viene ampliato o aggiornato si aumenterà il numero della versione; l'url fa riferimento al repository di GitHub che contiene la libreria; infine, install_requires è una lista che contiene tutte le dipendenze della nostra libreria, nel caso esempio la libreria dipende dalle librerie numpy e networkx, che sono anche le stesse dipendenze di SparkER. Questo significa che alcune delle classi della libreria utilizzano funzioni e classi di librerie esterne e tali dipendenze verranno poi installate insieme alla nostra libreria nel momento del pip install.
Il secondo è il file testuale "LICENSE.txt"; questo contiene la licenza che si intende applicare al proprio software, ovvero ciò che gli utenti possono e non possono fare con la propria libreria. Si può anche decidere di non applicare alcuna licenza specifica e dunque applicare le leggi sul copyright vigenti, ma è sempre consigliato usare una qualche licenza altrimenti nessuno può riprodurre, cambiare o ridistribuire il prodotto (praticamente il software non è più opens source). Sulla documentazione di GitHub sono presenti diversi tipi di licenza, a seconda di cosa si necessita (spesso viene copiata la licenza del MIT, la più usata).
Il terzo file, (opzionale ma se presente è buona norma) è il "README.md" (o "README.rst").
Il contenuto del file dovrebbe descrivere brevemente qual'è l'obiettivo della libreria e volendo la guida all'installazione e all'uso (se non è presente una documentazione specifica).
Alla fine di tutto la directory principale dovrà contenere la directory contenente il file init e le classi della libreria, il file setup.py, LICENSE.txt e, se preparato, il README.rst .

| | |
|---|---|
| datasets | py_sparker for python3 |
| examples | generalized supervised mb |
| sparker | pandas wrapper |
| sparker_python2 | py_sparker for python3 |
| README.md | Update README.md |
| requirements.txt | py_sparker for python3 |
| setup.py | py_sparker for python3 |

Figura 11: Repository di SparkER su GitHub

Nel repository di SparkER, mostrato in figura 11, si può vedere che non è presente il file LICENSE.txt; questo è dovuto al fatto che tale file è contenuto nella directory superiore, essendo presente nel repository anche l'implementazione per Scala di SparkER; inoltre nel repository è presente anche un file "requirements.txt", che ha funzione analoga all'install_require visto in precedenza, solo che tale file è utile qualora non si volesse installare la libreria, ma la si volesse semplicemente usare nella directory scaricata.

## 5.3 Upload su PyPI

Una volta preparato il pacchetto, la parte dell'effettivo upload della libreria può essere vista come una semplice sequenza di comandi della shell (prompt dei comandi su windows).
Posizionadosi nella directory principale, si crea la source distribution usando il comando "python setup.py sdist"; in seguito, si installa twine, un pacchetto di utility, con "pip install twine", e una volta installato si usa il comando "twine upload dist/* ". Verrà richiesto di inserire le credenziali dell'account di PyPI e fatto ciò il pacchetto sarà reso disponibile su PyPI.
Per installare il pacchetto caricato si usa il già visto "pip install libreriaesempio" , per usarlo con python si usa l'import statement "import libreriaesempio". Nel caso in cui si dovesse aggiornare o aggiungere file alla libreria, bisognerà semplicemente ricaricare la libreria su GitHub, modificare il setup.py aggiornando la versione e, se sono state aggiunte nuove dipendenze, l'install_require e rieffettuare tutti gli step della shell.

# 6 Conclusioni

## 6.1 Cosa è stato realizzato

Il lavoro svolto ha prodotto un aumento della flessibilità di SparkER, aggiungendo funzionalità alla parte dei wrapper, consentendo l'utilizzo di dataset in formato excel e l'utilizzo di query sql per ottenere dataset da database, e fornendo anche dei metodi generali che consentano di svolgere le funzioni principali di SparkER utilizzando una sola riga di codice.
Inoltre la documentazione che è stata realizzata, in lingua inglese e disponibile su https://sparker-documentation.readthedocs.io/en/latest/, ha permesso di spiegare tutti i metodi e le funzioni che devono essere utilizzati per ottenere i risultati che ci si aspetta da SparkER in modo corretto. L'upload su PyPI della libreria è stato un passo quasi obbligato nell'ambito delle librerie open source, permettendone l'acquisizione in modo rapido.

## 6.2 Difficoltà incontrate e conoscenze acquisite

Le difficoltà più grandi nello svolgimento del lavoro sono state nella parte di comprensione del framework di Spark, oltre che nel preparare tutto l'ambiente sul proprio calcolatore per pyspark (l'OS di windows non ha aiutato in questo senso); difficoltà sono state incontrate anche nel comprendere gran parte degli elementi teorici di SparkER, e ancor di più rielaborarli per poi spiegarli (in inglese) nella documentazione. Lavorare su SparkER è stato un modo per affinare le mie conoscenze del python e per introdurmi ad Apache Spark. Studiare e capire a grandi linee la teoria che ne sta alla base, l'entity resolution e i suoi problemi, è stato di grande interesse per me, in particolare gli argomenti riguardanti il blocking e BLAST.
Lavorare con readthedocs e sphinx mi è servito per approcciarmi al mondo dell'open source, così come l'utilizzo di PyPI e e GitHub, mi ha permesso di servirmi di strumenti che saranno utili anche in un futuro ambito lavorativo.

## 6.3 Ringraziamenti

Ringrazio i miei genitori che hanno sempre fatto per me enormi sacrifici per permettermi di concentrarmi solo sugli studi.
Ringrazio la mia ragazza Rebecca e i miei amici, di università e non, che mi hanno accompagnato per tutto il percorso universitario compiuto fin qui.
Ringrazio il PhD Luca Gagliardelli per la costanza con cui mi ha seguito e delucidato sul proprio software, nonchè per il tempo che ha speso per spiegarmi come installare Spark.

# 7 Bibliografia

1. Simonini, G., Bergamaschi, S., Jagadish, H. V. (2016). BLAST: a Loosely Schema-aware Meta-blocking Approach for Entity Resolution. PVLDB, 9(12), 1173–1184.

2. Papadakis, G., Koutrika, G., Palpanas, T., Nejdl, W. (2014). Meta-blocking: Taking entity resolution to the next level. IEEE TKDE.

3. Papadakis, G., Papastefanatos, G., Palpanas, T., Koubarakis, M., Green, E. L. (2016). Scaling Entity Resolution to Large , Heterogeneous Data with Enhanced Meta-blocking, 221–232. IEEE TKDE.

4. Papadakis, G., Ioannou, E., Niederée, C., Fankhauser, P. (2011). Efficient entity resolution for large heterogeneous information spaces. Proceedings of the Fourth ACM International Conference on Web Search and Data Mining - WSDM '11, 535.

5. Gagliardelli, L., Zhu, S., Simonini, G., Bergamaschi, S. (2018). Bigdedup: a Big Data integration toolkit for duplicate detection in industrial scenarios. In 25th International Conference on Transdisciplinary Engineering (TE2018) (Vol. 7, pp. 1015-1023).

# 8 Sitografia


1. https://docs.github.com/en/repositories
2. https://pypi.org
3. https://readthedocs.org
4. https://www.sphinx-doc.org/en/master/
5. https://docutils.sourceforge.io/rst.html
6. https://github.com/Gaglia88/sparker
7. https://spark.apache.org/docs/latest/