ENEA
Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile

Ministero della Transizione Ecologica

Servizi informatici per l'ampliamento della piattaforma software 'ECListener' con la realizzazione di una ontologia delle Energy community ed il relativo webcrawler

Alberto Botti, Gabriele Di Segni

arakne
do I.T. better

Report RdS/PTR2021/042

SERVIZI INFORMATICI PER L'AMPLIAMENTO DELLA PIATTAFORMA SOFTWARE 'ECLISTENER' CON LA REALIZZAZIONE DI UNA ONTOLOGIA DELLE ENERGY COMMUNITY ED IL RELATIVO WEBCRAWLER

Alberto Botti, Gabriele Di Segni (Arakne S.r.l)

Dicembre 2021

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Triennale di Realizzazione 2019-2021 - III annualità
Obiettivo: Tecnologie
Progetto: Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali
Work package: Local Energy District
Linea di attività 1.48: Energy Communities: Implementazione servizi LEC
Responsabile del Progetto: Claudia Meloni, ENEA
Responsabile del Work package: Claudia Meloni, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno del Contratto "Servizi informatici per l'ampliamento della piattaforma software ECListener con la realizzazione di una ontologia delle Energy community ed il relativo webcrawler"
Responsabile Unico del Procedimento ENEA: Gregorio D'Agostino
Responsabile del Contratto per il Contraente: Alberto Botti

Indice

SOMMARIO 5
1 INTRODUZIONE 6
2 ONTOLOGIA 7
2.1 RELAZIONI TASSONOMICHE 7
2.1.1 Estrazione dei concetti 7
2.1.2 Costruzione della Tassonomia 7
2.2 RELAZIONI NON-TASSONOMICHE 9
2.3 SCRIPTS 11
2.3.1 Overview 11
2.3.2 Setup 11
2.3.3 virtualenv 11
2.3.4 FastText model 12
2.3.5 Spacy model 12
2.3.6 Run 12
2.3.7 Input 12
2.3.8 Output 12
3 WEBCRAWLER 13
3.1 ARCHITETTURA 13
3.2 FLUSSO PRINCIPALE 13
3.3 COMPONENTI 14
3.3.1 Frontera Overview 14
3.3.2 Frontera Customizations 14
3.3.2.1 Message bus 14
3.3.2.2 Interazione con giro ESA 15
3.3.2.3 Crawling Strategy 15
3.3.2.4 Salvataggio dati su DB 15
3.3.3 Scrapy 16
3.3.3.1 Spider 16
3.3.3.2 Invio per l’analisi semantica 16
3.3.4 Apache Kafka 17
3.3.5 EsaScore 17
3.3.6 Esempio di messaggi JSON per le code ESA 18
3.3.7 Esempio di dati salvati nel DB 20
3.3.8 Esempio di pagina salvate nel file output/results.jl 22
3.4 INSTALLAZIONE E CONFIGURAZIONE 23
3.4.1 Installazione 23
3.4.1.1 Frontera 23
3.4.1.2 virtualenv 23
3.4.1.3 requirements 23
3.4.1.4 files e directories 23
3.4.1.5 Configurazione 24
3.4.2 Apache Kafka 24
3.4.3 Docker EsaScore 24
3.4.3.1 Configurazione 25
3.4.3.1.1 Categorie 27
3.4.4 Altro 28
3.5 ESECUZIONE SPIDER 29
3.6 START/STOP/RESET 29
3.6.1 Apache Kafka 29

3.6.2 Docker EsaScore *29*
3.6.3 Componenti Frontera *29*
3.6.4 Logging *30*
3.7 OUTPUT 30
4 RIFERIMENTI BIBLIOGRAFICI 31
5 ABBREVIAZIONI ED ACRONIMI 31

Sommario

Le attività svolte nell'ambito del contratto consistono nella realizzazione di alcuni "Servizi informatici per l'ampliamento della piattaforma software ECListener". Le attività preminenti sono due: la realizzazione di una "ontologia" delle Energy community e la realizzazione di un "web crawler" selettivo in grado di selezionare e memorizzare le pubblicazioni sui media relative alle comunità energetiche, basandosi sull'ontologia definita.
La realizzazione dell'ontologia è stata conseguita utilizzando un "corpus linguistico" realizzato dall'ENEA nei mesi precedenti al contratto e fornito tramite una banca dati relazionale. Tale banca dati a sua volta è il risultato di una campagna di raccolta selettiva svolta utilizzando gli algoritmi già in uso nel webcrawler ENEA. L'ontologia è stata resa fruibile tramite due documenti in formato "owl" (Web Ontology Language), contenti rispettivamente le relazioni di tipo "tassonomico" (specializzazione ed astrazione) e quelle "ontologiche", ovvero non gerarchiche o non tassonomiche
Il webcrawler selettivo è stato realizzato basandosi sulla libreria del tool python "Frontera". Il software è stato realizzato sulla piattaforma ENEA "babylon" in modalità "docker" per consentirne la portabilità sulla piattaforma ECListener indipendentemente dalle librerie installate e dalle loro versioni. I costituenti base sono di tre tipologie: spider, stratego e "db worker". Gli spider sono dei microservizi che prelevano una specifica risorsa da un url assegnato. Gli stratego definiscono le priorità degli url da visitare e distribuiscono agli spider il lavoro. I bersagli degli stratego sono stati definiti partendo da un insieme di MEDIA (cosiddetti seed) individuati e forniti da ENEA basandosi sulla frequenza delle acqusizioni pregresse. I "db worker" si occupano della interoperabilità con la banca dati per l'immagazzinamento dei dati. Un filtro di pertinenza è definito utilizzando l'ontologia per immagazzinare solo i documenti inerenti al dominio "Comunità Energetiche" formalizzato tramite l'ontologia. Questo filtro si basa su una funzione obiettivo definita da Arakne e data in uso all'ENEA, ma può essere sostituita con altre funzioni obiettivo definite in altre parti dell'attività ECListener. Tutto il software è stato sviluppato utilizzando il linguaggio python e garantendo portabilità (docker), interoperabilità e disponibilità dei sorgenti (open source).

1 Introduzione

Questo documento descrive le attività di Fase 1 e Fase 2 eseguite da Arakne per ENEA nell'ambito nel contratto "Servizi informatici per l'ampliamento della piattaforma software ECListener con la realizzazione di una ontologia delle Energy community ed il relativo webcrawler", condotto nella Linea di Attività 1.48.

Fase 1

La sezione 2, che si riferisce alla Fase 1, è organizzata in due sottosezioni principali che illustrano la generazione automatica in formato OWL di un'ontologia di dominio: la prima si concentra sulla definizione dei nodi dell'ontologia e sulla generazione delle relazioni tassonomiche che intercorrono tra di essi; la seconda spiega il processo utilizzato per la generazione delle relazioni non-tassonomiche.
In ognuna delle sottosezioni sono indicate sia le tecnologie che gli algoritmi adottati.

Fase 2

Lo scopo di questa fase del progetto, descritta nella sezione 3, è la realizzazione di un web-crawler selettivo con un filtro semantico tarato sulla ontologia individuata nella fase 1 del progetto.

2 Ontologia

2.1 Relazioni Tassonomiche

Un'ontologia si compone di nodi, o "concetti", tra loro legati da relazioni. Questa sezione descrive la procedura implementata per l'apprendimento, a partire da un set di documenti forniti in input da Enea (corpus), dei concetti e delle relazioni gerarchiche tra di essi, ossia delle loro relazioni tassonomiche.

2.1.1 Estrazione dei concetti

Per "concetto" si intende l'unità elementare del significato, che utilizzano gli esseri umani per organizzare e condividere la conoscenza.
L'estrazione di concetti mira a ridurre la dimensionalità dei dati in ingresso, raccogliendo le informazioni pertinenti ed eliminando ridondanze e fonti di rumore.
L'approccio adottato per estrarre i concetti dai documenti del corpus si basa sull'idea che un testo possa essere interpretato come una "miscela pesata" di un insieme di *concetti naturali*, ossia definiti da esseri umani e facili da spiegare. Per la determinazione di tali entità vengono utilizzate le voci di Wikipedia, in quanto rappresentano attualmente il più grande archivio di conoscenze sul Web (codificato in linguaggio naturale) e sono sottoposti a continuo sviluppo, aumentando di ampiezza e profondità costantemente nel tempo.
Le voci di Wikipedia più rilevanti e comuni a tutti i testi del corpus saranno utilizzate come base per la costruzione dei concetti dell'ontologia: in particolare a partire dal contenuto delle voci di Wikipedia, che svolgono una funzione paragonabile a dei "contenitori di concetti naturali pesati", verrà estratto l'identificatore (*label*) del nodo.
La mappatura di un testo del corpus in una sequenza ponderata di voci di Wikipedia è ottenuta con i seguenti passaggi:

1. ogni testo viene tokenizzato in modo eliminare le stopwords e convertire i token in lemmi;
2. Poiché preventivamente, ogni lemma è stato rappresentato nello spazio delle voci di Wikipedia (~162K voci per la versione italiana) assegnando a ciascuna voce un peso pari al tf-idf del lemma per quella determinata voce, il vettore ~162K dimensionale rappresentativo di ciascun testo è calcolato come la somma pesata dei vettori dei lemmi che lo compongono.
3. viene costruita una matrice Testi (609) x Voci (~162K) in cui le righe sono tutti i testi e le colonne sono le voci di Wikipedia.
4. a partire da questa matrice, per isolare le voci più significative, sono state mantenute le colonne il cui valor medio su tutti i testi ecceda una soglia scelta euristicamente (in questo caso 0.012). Questo passaggio ha ridotto la dimensionalità della matrice Testi x Voci, lasciando 34 voci significative.

Queste voci saranno gli elementi dai quali verranno costruiti i nodi dell'ontologia.

2.1.2 Costruzione della Tassonomia

Col fine di ottenere una tassonomia di concetti in forma gerarchica, è stato necessario utilizzare la rappresentazione vettoriale delle voci nello spazio dei lemmi: ossia, ogni voce è stata rappresentata nella base dei lemmi unici presenti nei 34 voci di Wikipedia selezionate secondo la procedura descritta nel paragrafo precedente.
Si è così costruita una matrice Voci (34) x Lemmi (4093). Questa matrice è stata ridotta dimensionalmente utilizzando i seguenti criteri:

1. sono stati mantenuti i lemmi comuni ad almeno il 10% delle voci (righe)
2. sono stati mantenuti i lemmi che hanno un tf-idf maggiore del tf-idf medio di riga.

Alla fine di questa operazione le 34 voci sono state rappresentative in uno spazio di 700 lemmi.

Il passo successivo è stato quello di creare uno spazio ordinato dei "vettori-concetto" [1]. A questo scopo è stato utilizzato lo strumento della SOM: ossia una rete neurale costituita da una griglia di neuroni artificiali - *nodi* - che vengono ordinati adattando la loro topologia ai dati passati in fase di apprendimento non

supervisionato [4, 5]. La SOM rappresenta un metodo per il clustering, la riduzione della dimensionalità e la ricerca di un ordinamento topologico dei dati. In fase di apprendimento, la SOM aggiorna la sua griglia di *nodi* "adattandoli", tramite l'aggiornamento di pesi relativi ad ogni nodo, ai vettori passati in input. La mappa risultante è organizzata in modo tale per cui vettori in input simili si localizzano l'uno vicino all'altro sulla mappa.
In particolare, è stata implementata una SOM quadrata, ossia una griglia con stesso numero di righe e colonne $n = m = 5$, con un numero complessivo di nodi pari a $5\sqrt{N} \simeq 25$, dove $N = 34$, ossia numero di dati (voci) usate per addestrare la SOM.
La tassonomia viene creata dall'alto (livello zero) verso il basso (k-esimo livello) clusterizzando gerarchicamente i nodi della SOM. Il livello zero consiste di tutti i vettori delle voci appartenenti ad un singolo insieme.
Per ottenere il primo livello, i vettori vengono raggruppati in base al seguente procedimento:

1. a partire da essi viene addestrata una SOM, ottenendo così un insieme di nodi di cardinalità inferiore ai dati di input;
2. viene applicato un algoritmo di clusterizzazione sui nodi della mappa, utilizzando il metodo DBSCAN - scelto poichè non richiede di sapere il numero di cluster a priori ed è in grado di trovare cluster di forme arbitrarie, in quanto si basa sulla distanza tra gli elementi e sulla loro distribuzione nello spazio;
3. ad ogni cluster afferiscono tutti i vettori vicini ai nodi che lo compongono, ottenendo quindi una clusterizzazione dei dati di partenza.

A tal punto, per ottenere il secondo livello della tassonomia, viene applicato nuovamente il procedimento appena descritto sui vettori voce appartenenti ad un cluster formatosi nel primo livello: nel dettaglio, viene addestrata una SOM separatamente per ogni gruppo di vettori di un cluster, dalla quale vengono formati nuovi "sotto-cluster".
Il processo si conclude quando i cluster dell'ultimo livello sono composti da un solo elemento, ottenendo così una tassonomia di sottoinsiemi di vettori.
Nel nostro caso il procedimento si ferma al 4 livello.
Per ogni cluster, viene definita l'etichetta, il *label,* che andrà a rappresentare il concetto dell'ontologia: scorrendo le componenti dei vettori, viene selezionata la componente più rilevante, ricordando che ad ogni componente del vettore corrisponde un lemma, ossia un candidato a rappresentare un concetto naturale. I criteri di selezione adottati sono

1. un valore tf-idf maggiore del valor medio nel cluster;
2. apparire in un numero di articoli superiore ad una certa soglia euristica.

Così facendo, ogni cluster ha un *label*, che racchiude l'informazione più rilevante comune a tutte le voci Wikipedia presenti nell'insieme specifico: viene sfruttato un maggior numero di informazioni presenti in ogni voce, arricchito e filtrato dalle combinazioni con gli altri elementi del cluster.
Nella Tabella 1 sono riportati i risultati ottenuti nei primi due livelli di questo procedimento: per ogni concetto del primo livello vengono estratti i concetti figli al secondo livello, per esempio il concetto "energia" ha come figli al secondo livello "valutazione, combustibile, consumo, fotovoltaico...".

In questa maniera si ottiene la tassonomia di concetti, a partire dalla quale viene costruita l'ontologia: le *labels* formeranno i nodi dell'ontologia e la struttura gerarchica fornisce le relazioni tassonomiche tra di essi, portando alla creazione di un file OWL.
L'ontologia ottenuta tramite questa procedura è composta da 46 concetti - o *classi* - legati attraverso la gerarchia individuata tramite il clustering, sotto forma di proprietà ontologiche [3].

Tabella 1 concetti estratti nei primi due livelli della tassonomia.

Livello 1	**Livello 2**
energia	valutazione, combustibile,

	consumo, fotovoltaico [...]
edificio	società, quantità, gestione, intervento [...]
strategia	fattore, riduzione
certificazione	impatto, incentivo
valore	
reattore	distribuzione, biomassa
fabbisogno	

2.2 Relazioni Non-Tassonomiche

Con la procedura illustrata precedentemente, si è in grado di costruire, a partire da un corpus di testi, sia i concetti che le relazioni tassonomiche tra di essi.

Scoprire i tipi di relazioni ontologiche non tassonomiche, ossia non gerarchiche, è strettamente correlato a metodi che identificano nei corpora delle relazioni semantiche tra i concetti.

In particolare, i verbi che compaiono congiuntamente - in prima istanza nella stessa frase - con i concetti per i quali si vuole stabilire una relazione sono candidati ottimali per identificare la relazione semantica tra due concetti.

Dunque per arricchire di relazioni non tassonomiche l'ontologia così come definita nel paragrafo precedente, si è proceduto come di seguito descritto [2]:

1. il corpus fornito da Enea è stato tokenizzato, pulito dalla stopword e lemmatizzato;
2. vengono considerate tutte le possibili coppie formate dai concetti dell'ontologia, evitando di considerare coppie identiche, ad esempio (enegia, energia). Nel nostro caso vengono individuate 1035 coppie;
3. il corpus viene suddiviso in frasi;
4. per ogni coppia di concetti, vengono scorse tutte le frasi e qualora in una frase siano presenti entrambi nell'ordine prestabilito dalla coppia, usando un riconoscitore di POS in italiano, vengono estratti e salvati tutti i verbi della frase;
5. per ogni coppia viene così costruita una *nube* di verbi, di grandezza variabile. Un esempio viene riportato nella seguente tabella 2.

Tabella 2 nube di verbi trovata per la coppia di concetti ('energia', 'petrolio')

coppia di concetti	nube di verbi
energia - petrolio	trattare, coniugare, vistare, produrre, derivare, riservare, limitare, impiegare, produrre, ricordare, stabilire, seguire, considerare, stabilire, considerare, realizzare, occupare,

	conseguire, comportare, cumulare, prevedere

6. per ogni nube, si prende in considerazione la rappresentazione vettoriale dei verbi -sfruttando modelli in italiano pre-testati costruiti con l'algoritmo Word2Vec [6] - e viene calcolato il *medoide,* ossia l'elemento più vicino al centroide della nube. Infine, si riconduce il vettore al verbo da esso rappresentato, così da trovare un *label* per la nube. Nel caso specifico dei concetti ('energia', 'petrolio') il verbo trovato è 'considerare', notando che nel calcolo del centroide viene tenuto conto del numero di occorrenze di un determinato vocabolo;
7. Il verbo così trovato sarà l'etichetta per la relazione che intercorre tra i due concetti.

Nel caso in cui ad una coppia di concetti non viene associata nessuna nube di verbi, se ne deduce che tra i due concetti, nel loro rispettivo ordine, non sussiste nessuna tipologia di relazione all'interno del corpus.
In questa maniera vengono trovate delle etichette per le relazioni non tassonomiche tra i concetti dell'ontologia, che nel nostro caso sono 101.
Queste ultime vengono introdotte nell'ontologia precedentemente creata come "ObjectProperties" nel file OWL. Siccome più relazioni non tassonomiche possono essere etichettate dallo stesso verbo, viene definita un'unica relazione con tale nome sull'insieme di concetti pertinenti - rispettivamente come dominio o codominio della relazione- mettendo poi le opportune restrizioni [3].
Un esempio di questo fenomeno viene riportato nella seguente tabella, senza elencare esplicitamente né l'intero dominio e codominio della relazione presa in esame, né tutte le restrizioni definite per tale relazione.

Tabella 3 struttura della relazione "considerare" nel file OWL.

considerare		
dominio	**codominio**	
settore, efficenza, impianto, consumo, energia, prezzo, sviluppo, rete [...]	valore, investimento, petrolio, edificio, fonte [...]	
restrizione 1	dominio: energia	codominio: petrolio

In conclusione, l'ontologia creata seguendo le precedenti procedure è composta da 46 concetti, legati tra loro da relazioni tassonomiche di tipo *SubclassOf*, arricchite da 101 relazioni non tassonomiche etichettate da verbi.

2.3 Scripts

2.3.1 Overview

Gli script, i file txt (README e requirements) ed i dati di input custom per la costruzione dell'ontologia a partire dai vettori voci/lemmi rilevanti, sono rilasciati come archivio tgz: **eclistner_ontology_scripts.tgz**
Questo file tgz lo si può trovare nella home dell'utente dedicato sulla macchina babylon nel path ~/**eclistner_ontology_scripts.tgz** . I path ~/ richiamati nel seguito si riferiscono alla home di questo user.
L'alberatura risultante dopo aver estratto l'archivio, scaricato i file aggiuntivi (come da istruzioni di setup) e fatto il run degli script (per generare gli output/) è la seguente:

```
.
├── cc.it.300.bin [da scaricare a parte[1]]     Modello FastText italiano, cfr README
├── ClusteringMinisomToTree.py                  Script per Clustering con minisom
├── CorpusAnalysis.py                           Script Analisi/Esplorazione del Corpus
├── GetOntologyConcept.py                       Script per estrazione Ontology Concept
├── input                                       Dati di input
│   ├── corpus_eclistner.txt                    Corpus di articoli
│   ├── enea.json                               Lista dei 700 lemmi di partenza
│   ├── toAvoidWords.txt                        Blacklist di parole
│   └── voices_pandas_34_700.pickle             Vettori di input (34 voci x 700 lemmi)
├── NonTaxonomicRelation.py                     Script per Relazioni non Tassonomiche
├── OntologyLabeling.py                         Script per aggiungere le relazioni
├── OntologyTaxonomicRelation.py                Script per Relazioni Tassonomiche
├── output  [generati durante il run]           Dati di output
│   ├── categories.csv                          CSV dei termini dell'ontologia
│   ├── ClassOntology.txt                       Lista dei termini dell'ontologia
│   ├── ClusterResults.json
│   ├── ClusterResults.p
│   ├── NonTaxonomicRelation.p
│   ├── owl
│   │   ├── OntologyTaxonomic.owl               Ontologia solo relazioni tassonomiche
│   │   └── OntologyWithRelation.owl            Ontologia anche relaz. non tassonomiche
│   └── plots                                   Plot di output
│       ├── allLevelCluster.png
│       ├── bestLemmas.png
│       ├── lemmasPerArticle.png
│       ├── levelCluster.png
│       └── numSentences.png
├── README.txt                                  File README
├── requirements.txt                            Pacchetti python3 richiesti
└── RunAll.sh                                   Script per il run automatico
```

Si rimanda alle sezioni precedenti ed al README per una descrizione più dettagliata dei vari script.py e degli step da eseguire per il setup/run.

2.3.2 Setup

Sulla macchina babylon, nel path ~/eclistner_ontology_scripts è già stato estratto l'archivio tgz, preparato il virtualenv (~/eclistner_ontology_scripts/venv), installati i vari pacchetti python e scaricati i dati di input addizionali (spacy e fasttext).
Le istruzioni qui sotto elencano le procedure da eseguire in caso si voglia rimettere su il setup.

2.3.3 virtualenv

Creare un **virtualenv** python3 (gli script sono stati testati in ambiente GNU/Linux con Python 3.6.8 e Python 3.8.10), attivarlo, se necessario aggiornare pip e quindi installare i pacchetti elencati nel file requirements.txt:

numpy==1.19.5 pandas==1.1.5 fasttext==0.9.2

[1] il file cc.it.300.bin occupa circa ~4.5GB ed è quindi troppo pesante da includere "as is"; va comunque scaricato prima di girare il codice, vedi istruzioni di setup.

MiniSom==2.2.9	matplotlib==3.2.2	seaborn == 0.11.1
spacy==3.1.1	scipy==1.4.1	nltk == 3.6.2
spacy-legacy==3.0.8	Owlready2==0.33	click==7.1.2

2.3.4 FastText model

Scaricare il modello FastText italiano utilizzato per individuare le label delle relazioni non tassonomiche.
Il file è "cc.it.300.bin.gz" (~4.5GB gzip file) e si può scaricare direttamente da (per poi scompattarlo - ad esempio con gunzip - nella cartella al livello dei vari file .py):

https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.it.300.bin.gz

Oppure da codice Python

```
import fasttext.util
fasttext.util.download_model('it')
ft = fasttext.load_model('cc.it.300.bin')
```

Il riferimento da citare in proposito è [7].

2.3.5 Spacy model

Scaricare la Pipeline Spacy per l'italiano *it_core_news_lg*, ad esempio con il comando:

```
python -m spacy download it_core_news_lg
```

Si veda https://spacy.io/models/it per maggiori informazioni e dettagli su licenza / sorgenti dati ecc.

2.3.6 Run

Se non già attivo, attivare il virtualenv creato durante il setup, e.g. ~/eclistner_ontology_scripts/venv.
Utilizzare lo script RunAll.sh per creare le cartelle di output (se non già presenti), cancellare eventuali output preesistenti e chiamare in sequenza i vari script per generare le ontologie.

2.3.7 Input

- **corpus_eclistner.txt**: i 609 testi degli "articoli" che costituiscono il corpus di input
- **voices_pandas_34_700.pickle**: i vettori delle 34 voci di wikipedia "rilevanti" rispetto agli articoli di input, nello spazio dei loro 700 lemmi più "rilevanti". È un pandas dataframe salvato in formato pickle.
- **enea.json**: lista dei 700 lemmi "rilevanti"
- **toAvoidWords.txt**: lista di stopwords e parole da scartare, usata per le relazioni non tassonomiche e nello script di esplorazione del Corpus ("CorpusAnalysis.py")

2.3.8 Output

Gli output principali sono i file owl delle ontologie:

- **output/owl/OntologyTaxonomic.ow**l
  ontologia con solo le relazioni tassonomiche
- **output/owl/OntologyWithRelation.owl**
  ontologia arricchita con relazioni non tassonomiche

Per visualizzare dinamicamente i file (come un grafo "interattivo") è possibile caricarli su di una web application, ad esempio: http://www.visualdataweb.de/webvowl/
Il file **output/ClassOntology.txt** contiene la lista dei termini dell'ontologia; il file **output/categories.csv** elenca i termini dell'ontologia in formato csv ("title", "text").
La cartella output/plots contiene alcuni grafici realizzati dallo script di analisi del Corpus (lemmi più comuni, numero di frasi per articolo, lemmi più comuni per POS) e dallo script di clustering.
La cartella output/ contiene anche i file intermedi prodotti dai vari script.

3 Webcrawler

3.1 Architettura

L'architettura del tool di web-crawling è articolata in diverse componenti:

1. il framework di crawling sviluppato con il tool python **Frontera**: https://github.com/scrapinghub/frontera
2. lo spider vero e proprio implementato con la libreria python **Scrapy**: https://scrapy.org/
3. un programma di analisi semantica dei testi, **EsaScore** - basato sul tool STAG di Arakne e sviluppato con tecnologia Apache Spark - utilizzabile tramite container Docker dedicato
4. dei topics Apache **Kafka** per far comunicare le varie componenti del sistema

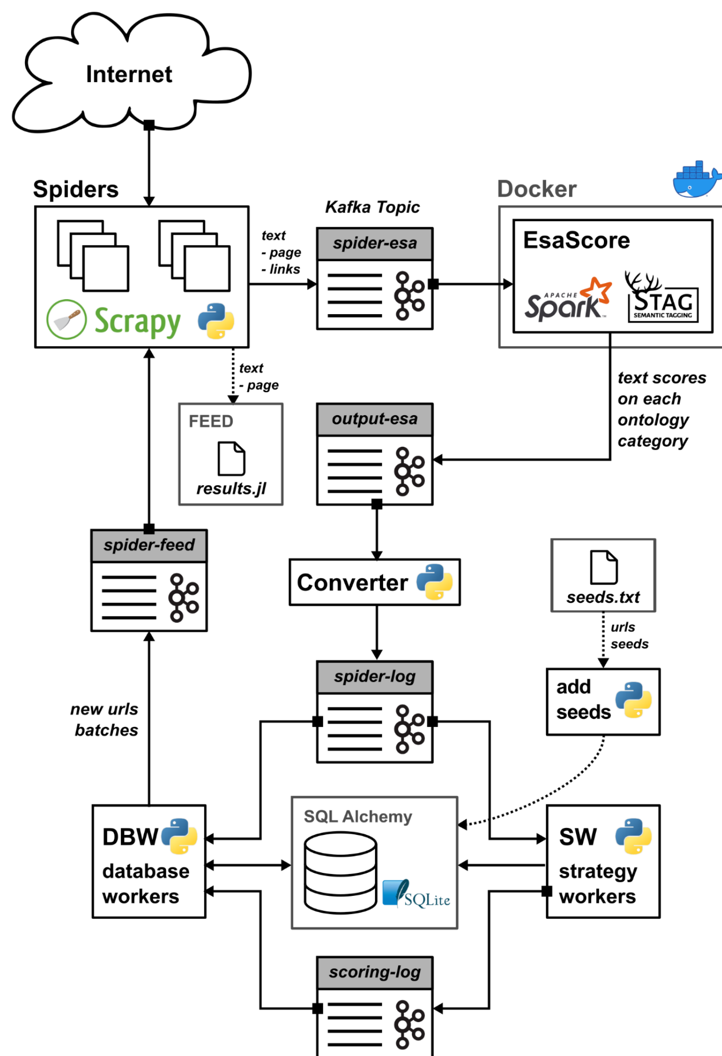

Figura 1 Componenti principali dell'architettura.

3.2 Flusso principale

Il flusso principale è il seguente:

1. l'utente specifica dei seeds urls che vengono salvati nella coda di crawling su DB (queue);
2. un componente dedicato di Frontera (DBW batches) periodicamente estrae dalla coda degli url da visitare (next_requests);

3. questi url sono inviati via topic Kafka allo Spider;
4. lo Spider visita le pagine, ne estrae il testo (che salva anche in un file dedicato results.jl) ed i link corredati da del testo “a contorno”;
5. il testo delle pagine e dei link estratti viene inviato all’analisi semantica;
6. il tool di analisi semantica analizza i testi fornendo una valutazione di pertinenza rispetto a ciascuna delle “categorie” (e.g. termini dell’ontologia);
7. i risultati dell’analisi semantica sono reimmessi nel flusso Frontera, dove un componente dedicato (SW) li utilizza per calcolare uno “score” ed un altro componente (DBW) li memorizza su Database. I link con uno score sopra soglia vengono aggiunti alla coda di crawling.

3.3 Componenti

3.3.1 Frontera Overview

Il framework si articola in alcune sottocomponenti che interagiscono tramite un message bus ed uno storage Database. Il message bus utilizzato in questo progetto è Apache Kafka, mentre il database è SQLite tramite il wrapper SQLAlchemy.
Si rimanda alla documentazione del tool per maggiori dettagli: https://frontera.readthedocs.io/en/latest/
Comunque, in breve, le sotto componenti standard sono:

- ***Strategy Worker*** **[SW] -** tipologia di worker che esegue il codice di “crawling strategy”: assegna gli score ai link estratti, decide se un link debba essere “scheduled” per essere visitato, ecc.
- ***DB worker*** **[DBW] -** worker deputato alla comunicazione con lo storage DB - e.g. salvataggio dei metadata e del contenuto estratto - e alla estrazione dei nuovi batches di url da visitare.
- ***Spider*** **-** il processo effettivo che estrae il contenuto dal web, leggendo da una coda di ingresso i link “scheduled” e salvando i risultati (testi estratti, links, …) su una coda di uscita. Per questo progetto la componente di spider utilizza il tool Scrapy integrandosi con esso attraverso dei componenti “middlewares” dedicati.

Il message bus richiederebbe di default le seguenti code (vedi sezione Apache Kafka per il “nome” effettivo assegnato al momento ai vari topic):

- **spider log -** stream di messaggi encoded (e.g. binari con msgpack) con l’output degli spider
- **scoring log -** “eventi” di aggiornamento degli score e dei flag di “scheduling” per i link (prodotti dallo SW e consumati dal DBW)
- **spider feed -** stream di messaggi dal DBW agli spider contenenti i nuovi “batch” di url da visitare
- **stats log -** coda aggiuntiva su cui vengono mandate statistiche sugli eventi gestiti dal framework

3.3.2 Frontera Customizations

Per interagire con il tool esterno di analisi semantica il flusso normale di Frontera dello spider log è stato alterato aggiungendo delle ulteriori code ed un sottocomponente custom di conversione ***Converter*** (definito in ec_listner/converter.py).

3.3.2.1 Message bus

Per gestire le code in più, la classe che gestisce il Message Bus è custom (ec_listner/kafkabus.py) ed è configurata dentro config/__init__.py (chiave `MESSAGE_BUS`)

3.3.2.2 Interazione con giro ESA

L'output degli spider relativo alle *page_crawled* e *links_extracted* non viene mandato direttamente sullo spider log, ma viene convertito[2] e instradato su una coda dedicata - **spider esa** - per essere sottoposto all'analisi semantica. [3] Una volta completata l'analisi semantica viene rimandato su una coda dedicata - **output esa** - che il componente ***Converter*** legge e ritrasforma nel formato atteso prima di scriverlo sulla coda spider log e riprendere il flusso Frontera standard.

3.3.2.3 Crawling Strategy

Il calcolo dello score "finale" da assegnare ad un link (e ad una pagina) viene effettuato da una funzione custom definita dentro la classe dello Strategy Worker. Quella custom in uso per il progetto è `EcListnerCrawlingStrategy` definita in ec_listner/strategy.py e configurata tramite l'opzione --strategy quando si fa partire lo strategy worker.
La formula per il calcolo dello score (`calculate_esa_score`) su un testo estratto (link o pagina) tiene conto sia dei risultati dell'analisi di pertinenza semantica su un testo che del "radius" (vedi sezione dedicata a Scrapy) ossia di quanto è "vicino" il testo al link in analisi

A. Si estraggono dai dati di input gli score di pertinenza del testo
B. Si identifica il più alto score di pertinenza per quel testo
C. Si fa la media degli score di pertinenza
D. Si fa la media tra B e C
E. Si penalizza lo score sulla base del radius, dividendo D per il logaritmo in base 2 di 2+radius. Il "radius" vale 0 (se è il testo di una pagina, o se tutto il testo è accanto al link in analisi) o più

Il numero E viene salvato come score e, nel caso dei link, viene confrontato con il valore della chiave di configurazione `LINK_SCORE_THRESHOLD` per decidere se visitare il link o meno.

3.3.2.4 Salvataggio dati su DB

La parte di salvataggio dei dati estratti (pagine e links) è deputata principalmente al componente DBW. Per semplificare l'estrazione delle informazioni sulle pagine analizzate i dati delle "crawled pages" (con i loro score e metadata compresi i risultati di pertinenza semantici) vengono salvati anche su una tabella dedicata del database (*pages*).
Questa personalizzazione è realizzata tramite una classe custom `EsaDistributed`, definita in ec_listner/distributed.py e richiamata in configurazione dentro al file config/worker.py tramite la chiave `BACKEND`. La classe, a sua volta richiama altre classi custom per l'interazione con il database:

- definizione e gestione tabella "pages" (ec_listner/models.py)
- coda di scheduling custom `EsaQueue` (ec_listner/queue.py) per personalizzare l'ordine di estrazione dei nuovi url da visitare (next_requests) in ordine di score decrescente.

Configurazioni

I file di configurazione dei componenti Frontera sono

- config/__init__.py: configurazioni trasversali a tutti i componenti, e.g. topics Kafka
- config/worker.py: configurazioni trasversali ai worker (SW e DBW)
- config/sw.py: configurazioni del SW

---

[2] Trasformato in formato JSON con singolo "messaggio" per link. Per semplificare la riconversione a valle del messaggio, nel JSON viene inclusa anche una chiave "request" con dentro la parte principale del payload iniziale base64 encoded per evitare problemi di serializzazione dei dati.

[3] Questa operazione è effettuata tramite una classe `BACKEND` custom `CrawlMessageBusBackend`, definita dentro il file ec_listner/backend.py e specificata per l'utilizzo da parte dello spider dentro il file di configurazione dedicato config/spider.py; per disabilitare questo instradamento/processing custom si può specificare la classe `BACKEND` di default nel file di configurazione.

- config/dbw.py: configurazioni del DBW
- config/spider.py: configurazioni integrazione Scrapy dentro Frontera
- logging.conf: configurazione dei livelli di logging dei componenti base di Frontera.

Le chiavi di configurazione custom (oltre a quelle dei topics Kafka aggiuntivi) sono:

- `LINK_SCORE_THRESHOLD` (config/__init__.py): livello di soglia sullo score di un link (per poterlo aggiungere alla coda di crawling)
- `WORKER_LOG_LEVEL` (config/worker.py): livello dei log per le classi “worker” custom

Si rimanda alla documentazione di Frontera per la descrizione delle altre chiavi di configurazione.

3.3.3 Scrapy

3.3.3.1 Spider

Lo spider vero e proprio è realizzato utilizzando il tool Scrapy, ed è contenuto in una classe custom `EcListnerSpider` (ec_listner/spiders/EcListner.py).
Lo spider è in ascolto sul topic spider feed in attesa di istruzioni su pagine da visitare. Quando ne riceve, per prima cosa estrae il testo (senza i tag) del body della pagina filtrando via <script> e <style>, lo invia per il salvataggio su file (output/results.jl) tramite il meccanismo dei `FEED` (vedi relative chiavi di configurazione nella documentazione di Scrapy) e lo invia anche nel flusso per l’analisi semantica (come se fosse un link, vedi più avanti).
Quindi cerca i link (tag `<a>`) presenti in pagina e per ogni link:

1. Scarta il link se è “chiaramente” non valido; e.g. se non ha href, o l’href è troppo corto, se punta alla stessa pagina in esame, se non comincia con http, ecc.
2. Scarta il link se corrisponde ad uno dei pattern elencati tra le `BLACKLIST_REGEXPS`
3. estrae i testi di “accompagnamento” livello per livello, risalendo nel DOM html (fino al livello del body, escluso) fino ad avere un minimo di `MIN_LINK_TEXT` caratteri totali[4]
4. concatena i testi così raccolti in un unico testo di accompagnamento, calcolando una variabile “radius” per tenere conto di quanto è lontano il testo dal link, come media pesata della frazione di caratteri per livello rispetto al totale caratteri. Più in dettaglio, data la lista dei testi intorno al link (un testo per ciascun livello, a partire dallo 0 che è il testo dentro al tag `<a>` del link)
    a. per ogni livello calcola la lunghezza
    b. calcola la lunghezza totale sommando quella dei vari livelli
    c. per ogni livello calcola la frazione di caratteri (lunghezza testo livello / lunghezza totale) e la pesa moltiplicandola per il numero d’ordine crescente del livello
    d. la somma dei c. per tutti i livelli è il “radius”
5. scarta il link se non ha abbastanza caratteri
6. invia il link per le successive analisi (yield)
7. esce dal loop di estrazione una volta che ha estratto `MAX_LINKS_PER_PAGE` validi

N.B. Quando lo spider ha raggiunto il numero di `CLOSESPIDER_PAGECOUNT` richieste fatte, **si chiude automaticamente** (e va rilanciato di nuovo se si vuole riprendere l’analisi delle nuove pagine in coda).

3.3.3.2 Invio per l’analisi semantica

La gestione dell’invio dei testi all’analisi semantica è gestita dalla classe `CrawlMessageBusBackend` già descritta nella sezione Interazione con giro ESA.

[4] Risalendo di livello, il “tag” sottostante viene rimosso, in modo che l’estrazione del testo del livello padre non contenga il testo del “tag” già estratto in precedenza.

I links extracted vengono spacchettati e trasformati in singoli messaggi JSON da mandare alla coda ESA, corredandoli - per semplicità di riconversone a valle - della “richiesta” originale (base64 encoded). In caso di link uguali, viene tenuto solo quello con radius minore.
Per quanto riguarda la page_crawled, il suo meccanismo standard di invio (che non avrebbe il testo a corredo ma solo il messaggio nel formato corretto atteso per la coda spider_log) viene intercettato, salvato temporaneamente in memoria e posposto a quando la classe procede alla analisi dei links_extracted. Tra i links infatti è stato aggiunto anche un “link” in più corrispondente alla pagina stessa; si può così preparare un messaggio JSON che ha sia il testo della pagina (preso da questo link extra) che la richiesta originale base64 encoded (che era stata temporaneamente salvata in memoria).

Configurazioni
I file di configurazione dei componenti Frontera sono
- scrapy.cfg
- ec_listner/settings.py: configurazioni Scrapy
- config/scrapy.py: configurazioni Frontera per Scrapy

Le chiavi di configurazione custom (oltre a quelle dei topics Kafka aggiuntivi) sono:
- `BLACKLIST_REGEXPS`: lista di pattern regular expression per escludere link dall'estrazione
- `MAX_LINKS_PER_PAGE`: massimo numero di link validi da estrarre per ciascuna pagina
- `MIN_LINK_TEXT`: lunghezza minima (caratteri) del testo di accompagnamento di un link

Si rimanda alla documentazione di Scrapy (e di Frontera) per la descrizione delle altre chiavi di configurazione. Tra le chiavi che sarebbe preferibile impostare prima di cominciare c’è il “nome” identificativo del crawler, impostando la variabile `USER_AGENT`; infatti il valore di default dovrebbe essere tipo: "`Scrapy/VERSION (+https://scrapy.org)`"

3.3.4 Apache Kafka
Per maggiori informazioni su Apache Kafka si veda: https://kafka.apache.org/
I seguenti topics Kafka (descrizione: nome_topic) - per semplicità creati tutti con partitions 1 - sono utilizzati dalle componenti del web-crawler per comunicare tra di loro
- scoring log: enea-scoring-log
- spider feed: enea-spider-feed
- spider log: enea-spider-log
- stats log: enea-stats-log
- spider esa: enea-spider-esa
- output esa: enea-output-esa

I nomi di questi topic possono essere cambiati avendo l’accortezza di aggiornali anche nei vari file di configurazione dei componenti; e.g. nello script reset.sh che crea i topic Kafka, nel file config/__init__.py per Frontera (e facendo partire di nuovo i componenti), nel file esascore.env per il docker EsaScore (facendo poi partire nuovamente il container e.g. con run.sh).

3.3.5 EsaScore
Il componente EsaScore - disponibile tramite una Docker image / container dedicato - si occupa di fare l’analisi semantica dei testi passati, utilizzando l’approccio Explicit Semantic Analysis. Per maggiori informazioni si veda https://en.wikipedia.org/wiki/Explicit_semantic_analysis e relative referenze.
È un tool che utilizza “STAG” una soluzione di analisi semantica sviluppata da Arakne basata su tecnologia Apache Spark e che usa le relative librerie di Machine Learning (Spark MLLib https://spark.apache.org/mllib/) e processing dei testi (Spark NLP di John Snow Labs https://github.com/JohnSnowLabs/spark-nlp https://nlp.johnsnowlabs.com/).
Il “modello” ESA utilizzato per l’analisi è stato addestrato con un dump di Wikipedia Italia; la matrice TF-IDF risultante contiene 524288 “lemmi” e 162463 voci “rilevanti” (voci di Wikipedia).

All’avvio del programma, EsaScore legge un file (locale al container Docker) con una lista di “categorie”, e.g. i termini dell’ontologia, con il loro “title” ed il loro “text”: quest’ultimo è la rappresentazione testuale della categoria individuata dal “title” ed è il testo effettivamente usato per eseguire il confronto semantico; in prima approssimazione il “text” corrisponde al “title” stesso.
I testi di input delle categorie sono “trasformati” e rappresentati nello spazio vettoriale dei lemmi, quindi si utilizza la matrice TF-IDF per “trasformarli” nello spazio vettoriale delle voci.
I “nuovi” testi da analizzare sono invece passati come JSON su un topic/coda Kafka dedicata (spider esa) da cui EsaScore li legge e li processa in parallelo con micro batch (e.g. 20 testi per volta; valore configurabile). Ciascun testo di input viene anch’esso “trasformato” come le categorie, rappresentato dapprima nello spazio vettoriale dei lemmi e poi “trasformato” nello spazio vettoriale delle voci.
La “pertinenza” di ciascun testo di input con ciascuna categoria si calcola tramite la *cosine similarity* dei rispettivi due vettori nello spazio delle voci.
Spark permette di eseguire queste operazioni in maniera matriciale / parallela per migliorare le prestazioni. Il risultato di questi punteggi di pertinenza - sulle varie categorie per ciascun testo - viene riassemblato in un messaggio JSON e mandato su una coda/topic Kafka di uscita (output esa).
Quando si crea il container, tramite le variabili di ambiente (e.g. il file esascore.env o i parametri --env) è possibile ridefinire una serie di parametri lato Kafka (e.g. nomi topic, timeout, ecc.) e lato Spark (numero cores, memoria da usare, ecc.) . Con il meccanismo dei Docker “volumes” è possibile anche scavalcare il file di input delle categorie dentro al container con un file della macchina host, così da utilizzare una lista custom di categorie.

3.3.6 Esempio di messaggi JSON per le code ESA

Per semplicità di visualizzazione i campi lunghi (e.g. “text”, “request”) sono stati tagliati e solo un sottoinsieme dei risultati di pertinenza semantica (le prime tre e l’ultima categoria) è mostrato per i messaggi della coda “output esa”.

Coda spider esa (uscita dello Spider ed ingresso dell’analisi semantica)

page crawled


```
{
	"url": "https://www.rinnovabili.it/",
	"from_url": "https://www.rinnovabili.it/",
	"text": "Energia Eolico Biomassa Efficienza Energetica Finanziamenti Fotovoltaico Geotermia Idroelettrico Idrogeno Moto Marino Politiche Energetiche Sistemi di accumulo Termico Termodinamico Comunità Energetiche [...]",
	"radius": "0",
	"type": "page",
	"fingerprint": "4a5a4195d7a920fa2d83475dc728b3260bd7dd4a",
	"request": "ksQCcGOVu2h0dHBzOi8vd3d3LnJpbm5vdmFiaWxpLml0L8z[...]",
	"key": "e4bfe21afd14ad3b0d7f9a5a49e03581d01c55f6"
}
```


link extracted


```
{
	"url": "https://www.rinnovabili.it/energia/comunita-energetiche-rinnovabili/comunita-energetiche-fotovoltaiche-lombardia/",
	"from_url": "https://www.rinnovabili.it/",
	"text": "Comunità energetiche fotovoltaiche, oltre 6.000 in Lombardia 5 Agosto 2021",
	"radius": "0.5342465753424658",
	"type": "link",
	"fingerprint": "4a5a41954bd21fb079875e651f86dfeb9cecdf02",
	"request": "k8QCbGWVu2h0dHBzOi8vd3d3LnJpbm5vdmFiaWxpLml0L8Q[...]",
	"key": "e4bfe21afd14ad3b0d7f9a5a49e03581d01c55f6"
}
```


Coda output esa (uscita dell’analisi semantica, ingresso converter e quindi spider-log)

- page crawled


```
{
	"idx": 0,
```

*"fingerprint": "4a5a4195d7a920fa2d83475dc728b3260bd7dd4a",
"request": "ksQCcGOVu2h0dHBzOi8vd3d3LnJpbm5vdmFiaWxpLml0L8z[...]",
"url": "https://www.rinnovabili.it/",
"text":"energia eolico biomassa efficienza energetica finanziamenti fotovoltaico geotermia idroelettrico idrogeno moto marino politiche energetiche [...]",
"radius": "0",
"key": "e4bfe21afd14ad3b0d7f9a5a49e03581d01c55f6",
"type": "page",
"size": 11433,
"results": [
{
"score1": 0.37072070151777675,
"c1": "energia"
},
{
"score1": 0.3399872774371581,
"c1": "sviluppo"
},
{
"score1": 0.32905563960694795,
"c1": "produzione"
},
{
"score1": 0.28182414015735646,
"c1": "settore"
},
{
"score1": 0.05590340783341793,
"c1": "incentivo"
}
],
"model_label": "model",
"categories_label": "default",
"received_date": "2021-08-05 11:47:01",
"completion_date": "2021-08-05 11:47:16",
"status": "success",
"process_time": 0.81
}*

- link extracted

*{
"idx": 4,
"fingerprint": "4a5a41954bd21fb079875e651f86dfeb9cecdf02",
"request": "k8QCbGWVu2h0dHBzOi8vd3d3LnJpbm5vdmFiaWxpLml0L8Q[...]",
"url": "https://www.rinnovabili.it/energia/comunita-energetiche-rinnovabili/comunita-energetiche-fotovoltaiche-lombardia/",
"text":"comunità energetiche fotovoltaiche, oltre 6.000 in lombardia 5 agosto 2021",
"radius": "0.5342465753424658",
"key": "e4bfe21afd14ad3b0d7f9a5a49e03581d01c55f6",
"type": "link",
"size": 74,
"results": [
{
"score1": 0.5917729299977252,
"c1": "fotovoltaico"
},
{
"score1": 0.26725520113750173,
"c1": "energia"
},
{
"score1": 0.1968428814634685,
"c1": "efficienza"
},
{
"score1": 0.13063726204068035,*

```
                "c1": "impianto"
        },
        {
                "score1": 0.03126693089281827,
                "c1": "certificato"
        }
        ],
        "model_label": "model",
        "categories_label": "default",
        "received_date": "2021-08-05 11:47:01",
        "completion_date": "2021-08-05 11:47:16",
        "status": "success",
        "process_time": 0.81
}
```


3.3.7 Esempio di dati salvati nel DB

Per migliorare la leggibilità, i JSON sono stati manipolati come descritto nella sezione precedente.

- **page** salvata nella DB table “pages” e poi estratta in output/pages.json tramite script parsedb.py


```
{
        "fingerprint": "4a5a4195d7a920fa2d83475dc728b3260bd7dd4a",
        "url": "https://www.rinnovabili.it/",
        "text": "energia eolico biomassa efficienza energetica finanziamenti fotovoltaico geotermia idroelettrico idrogeno moto marino politiche energetiche sistemi di accumulo termico termodinamico comunità energetiche [...]",
        "score": 0.2773262050974584,
        "meta": {
        "scrapy_meta": {
                "download_timeout": 60.0,
                "download_slot": "www.rinnovabili.it",
                "download_latency": 2.998445987701416,
                "depth": 0
        },
        "domain": {
                "netloc": "www.rinnovabili.it",
                "name": "www.rinnovabili.it",
                "scheme": "https",
                "sld": "",
                "tld": "",
                "subdomain": "",
                "fingerprint": "e4bfe21afd14ad3b0d7f9a5a49e03581d01c55f6"
        },
        "fingerprint": "4a5a4195d7a920fa2d83475dc728b3260bd7dd4a",
        "state": 1,
        "jid": 0,
        "encoding": "utf-8",
        "esa": {
                "idx": 0,
                "fingerprint": "4a5a4195d7a920fa2d83475dc728b3260bd7dd4a",
                "url": "https://www.rinnovabili.it/",
                "text": "energia eolico biomassa efficienza energetica finanziamenti fotovoltaico geotermia idroelettrico idrogeno moto marino politiche energetiche sistemi di accumulo termico termodinamico comunità energetiche [...]",
                "radius": "0",
                "key": "e4bfe21afd14ad3b0d7f9a5a49e03581d01c55f6",
                "type": "page",
                "size": 11433,
                "results": [
                {
                        "score1": 0.37072070151777675,
                        "c1": "energia"
                },
                {
                        "score1": 0.3399872774371581,
                        "c1": "sviluppo"
                },
```

```
			{
				"score1": 0.32905563960694795,
				"c1": "produzione"
			},
			{
				"score1": 0.28182414015735646,
				"c1": "settore"
			},
			[...]
			{
				"score1": 0.05590340783341793,
				"c1": "incentivo"
			}
			],
			"model_label": "model",
			"categories_label": "default",
			"received_date": "2021-08-05 11:47:01",
			"completion_date": "2021-08-05 11:47:16",
			"status": "success",
			"process_time": 0.81
		},
		"link_score": 1.0
		}
}
```


- **link** salvato nella DB table “metadata” e poi estratto in output/metadata.json tramite script parsedb.py


```
{
	"fingerprint": "4a5a41954bd21fb079875e651f86dfeb9cecdf02",
	"url":			"https://www.rinnovabili.it/energia/comunita-energetiche-rinnovabili/comunita-energetiche-fotovoltaiche-lombardia/",
	"score": 0.3434975497872048,
	"meta": {
	"scrapy_callback": null,
	"scrapy_errback": null,
	"scrapy_meta": {
		"url":		"https://www.rinnovabili.it/energia/comunita-energetiche-rinnovabili/comunita-energetiche-fotovoltaiche-lombardia/",
		"from_url": "https://www.rinnovabili.it/",
		"radius": 0.5342465753424658,
		"type": "link",
		"fingerprint": "4a5a41954bd21fb079875e651f86dfeb9cecdf02"
	},
	"origin_is_frontier": true,
	"domain": {
		"netloc": "www.rinnovabili.it",
		"name": "www.rinnovabili.it",
		"scheme": "https",
		"sld": "",
		"tld": "",
		"subdomain": "",
		"fingerprint": "e4bfe21afd14ad3b0d7f9a5a49e03581d01c55f6"
	},
	"fingerprint": "4a5a41954bd21fb079875e651f86dfeb9cecdf02",
	"esa": {
		"idx": 4,
		"fingerprint": "4a5a41954bd21fb079875e651f86dfeb9cecdf02",
		"url":		"https://www.rinnovabili.it/energia/comunita-energetiche-rinnovabili/comunita-energetiche-fotovoltaiche-lombardia/",
		"text": "comunità energetiche fotovoltaiche, oltre 6.000 in lombardia 5 agosto 2021",
		"radius": "0.5342465753424658",
		"key": "e4bfe21afd14ad3b0d7f9a5a49e03581d01c55f6",
		"type": "link",
		"size": 74,
```

```
            "results": [
            {
                  "score1": 0.5917729299977252,
                  "c1": "fotovoltaico"
            },
            {
                  "score1": 0.26725520113750173,
                  "c1": "energia"
            },
            {
                  "score1": 0.1968428814634685,
                  "c1": "efficienza"
            },
            {
                  "score1": 0.13063726204068035,
                  "c1": "impianto"
            },
            [...]
            {
                  "score1": 0.03126693089281827,
                  "c1": "certificato"
            }
            ],
            "model_label": "model",
            "categories_label": "default",
            "received_date": "2021-08-05 11:47:01",
            "completion_date": "2021-08-05 11:47:16",
            "status": "success",
            "process_time": 0.81
      }
      }
}
```


3.3.8 Esempio di pagina salvate nel file output/results.jl


```
{
   "url": "https://www.rinnovabili.it/",
   "from_url": "https://www.rinnovabili.it/",
   "text": "Energia Eolico Biomassa Efficienza Energetica Finanziamenti Fotovoltaico Geotermia Idroelettrico Idrogeno Moto Marino Politiche Energetiche Sistemi di accumulo Termico Termodinamico Comunità Energetiche [...]",
   "radius": 0,
   "type": "page"
}
```

3.4 Installazione e configurazione

3.4.1 Installazione

I vari componenti sono installati sulla macchina babylon all'interno nella home dell'utente dedicato. Nel seguito i vari path ~/ si riferiscono alla home di questo utente

3.4.1.1 Frontera

Il tool richiede Python 3 ed è stato testato con la versione 3.6.8 in ambiente GNU/Linux Centos 8.

3.4.1.2 virtualenv

Un virtualenv dedicato è già stato creato dentro ~/frontera/venv installandoci, dopo averlo attivato (ed aver aggiornato pip), i vari pacchetti necessari con il comando:

```
pip install -r ~/frontera/ec_listner/requirements.txt
```

Il virtualenv va attivato per poter utilizzare Frontera.

3.4.1.3 requirements

più in dettaglio i requirements python3 sono i seguenti:[5]

```
attrs==21.2.0
Automat==20.2.0
cachetools==4.2.2
cffi==1.14.6
cityhash==0.2.3.post9
confluent-kafka==1.7.0
constantly==15.1.0
cryptography==3.4.7
cssselect==1.1.0
frontera==0.8.1
greenlet==1.1.0
h2==3.2.0
hpack==3.0.0
hyperframe==5.2.0
hyperlink==21.0.0
idna==3.2
importlib-metadata==4.6.1
incremental==21.3.0
itemadapter==0.3.0
itemloaders==1.0.4
jmespath==0.10.0
kafka-python==2.0.2
lxml==4.6.3
msgpack==0.6.2
msgpack-python==0.5.6
parsel==1.6.0
priority==1.3.0
Protego==0.1.16
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycparser==2.20
PyDispatcher==2.0.5
PyMySQL==1.0.2
pyOpenSSL==20.0.1
pyzmq==22.1.0
queuelib==1.6.1
Scrapy==2.5.0
service-identity==21.1.0
six==1.16.0
SQLAlchemy==1.4.21
Twisted==21.2.0
typing-extensions==3.10.0.0
w3lib==1.22.0
zipp==3.5.0
zope.interface==5.4.0
```

3.4.1.4 files e directories

Cartella principale ~/frontera/ec_listner.

```
ec_listner
├── config                    cartella di configurazioni
│   ├── dbw.py                configurazione DBW
│   ├── __init__.py           configurazione comune
│   ├── spider.py             configurazione interazione Spider-Frontera
│   ├── sw.py                 configurazione SW
│   └── worker.py             configurazione comune SW e BDW
├── db
│   └── ec_listner.sqlite    storage DATABASE (cancellato da reset.sh)
├── ec_listner
│   ├── backend.py            classe backend custom per lo spider
│   ├── converter.py          conversione messaggi da output-esa a spider-log
│   ├── distributed.py        classe custom per personalizzare models e queue
│   ├── __init__.py
│   ├── kafkabus.py           classe MESSAGE_BUS custom
```

[5] il pacchetto confluent-kafka (e relative dipendenze) è richiesto solo dallo script logtopics.py (i.e. lo script di debug per salvare su log tutti i messaggi che passano nei vari topic Kafka). Se questo script non è necessario, si può evitare di installare quel pacchetto.

```
│   ├── models.py          modelli custom per gestione tabella DB pages
│   ├── queue.py           classi custom per gestione coda di crawling
│   ├── settings.py        settings di Scrapy
│   ├── spiders
│   │   ├── EcListner.py   crawler Scrapy
│   │   ├── __init__.py
│   └── strategy.py        classe Crawling Strategy custom
├── feed.sh                script per mettere seeds nella coda di crawling
├── follow.sh              script per concatenare i vari file di log
├── logging.conf           configurazione livelli di log Frontera base
├── logs                   cartella dei logs (svuotata da reset.sh)
│   ├── dbw_batches.log
│   ├── dbw_scoring.log
│   ├── sw.log
│   ├── scrapy.log
│   └── topics.log         c'è solo quando si fa partire anche topics
├── logtopics.py           script per salvare su log i messaggi dei topics
├── output
│   ├── results.jl         output testo pagine visitate (scritto da Scrapy)
│   ├── metadata.json      dump decodificato (pickle) tabella DB metadata
│   └── pages.json         dump decodificato (pickle) tabella DB pages
├── parsedb.py             script per il dump delle tabelle del database
├── pids                   cartella ausiliaria con i PIDS dei processi
├── README.md             README file
├── requirements.txt       pacchetti python richiesti da Frontera &co
├── reset.sh               cancella il DB, svuota i log e rimette i seeds
├── scrapy.cfg             file di configurazione di Scrapy
├── seeds
│   └── seeds.txt          file con gli url seeds
├── start.sh               script di lancio di tutti/un componente
└── stop.sh                script di stop di tutti/un componente
```

3.4.1.5 Configurazione

Si veda la sezione Frontera Customizations per il dettaglio dei file di configurazione e delle chiavi da cambiare / personalizzare. In particolare si ricorda di configurare e impostare un “nome” identificativo per il crawler impostando la variabile `USER_AGENT`.
I topics Kafka sono configurati con i nomi descritti Apache Kafka puntando al Kafka server locale.

3.4.2 Apache Kafka

Per maggiori informazioni su Apache Kafka si veda: https://kafka.apache.org/
Kafka è già installato nella home dell'utente, sottocartella kafka_2.13-2.8.0, non è una installazione di sistema (quindi non parte all'avvio della macchina), ma va lanciato manualmente.
Alcuni script ausiliari per lo start (di Kafka e di Zookeeper), stop ed il reset (i.e. ricreare i topics) sono disponibili nella cartella ~/frontera/kafka ed utilizzano i comandi della sottocartella bin/ dell'installazione Kafka e le configurazioni della sottocartella config/.
Le configurazioni sono quelle di default con l'unica modifica custom dell'aggiunta della chiave `clientPortAddress=localhost` alla configurazione di zookeeper.
N.B. Lo script start.sh al momento lancia i due processi in background (solo con un & alla fine): quindi se si chiude la shell dovrebbero venire interrotti. Per renderli "longevi" va modificato lo script mettendo tipo il nohup (vedi per esempio la sintassi commentata dentro allo script).
Per passare ad un altro server Kafka (e.g. di sistema oppure ospitato su di un'altra macchina). non si deve fare altro che cambiare i puntamenti a Kafka dei vari componenti; cfr. la sezione Apache Kafka

3.4.3 Docker EsaScore

Il file tgz dell'immagine è attualmente disponibile dentro la home dell'utente dedicato.
L'immagine è già stata importata (con docker image load) [la created date qui sotto non è aggiornata).

```
$ docker images
REPOSITORY         TAG          IMAGE ID     CREATED      SIZE
localhost/esascore  latest     bda77909f90d  30 hours ago  2.82 GB
```

Table 1 Dettagli immagine Docker e sue Componenti SW.

SW	Descrizione	Versione
esascore	docker-esascore_20210803.tgz (~2.8GB) immagine docker completa creata con il comando `docker image save` e caricabile con il comando `docker image load -i <file.tgz>` la entrypoint è: /opt/stag/esascore/esascore.sh	IMAGE ID bda77909f90d
openjdk:8	immagine docker di partenza	
Spark	scaricato con wget da https://archive.apache.org/dist/spark/spark-${SPARK_VERSION}/spark-${SPARK_VERSION}-bin-hadoop${HADOOP_VERSION}.tgz ed estratto e rinominato in /opt/stag/spark	SPARK_VERSION=2.4.8 HADOOP_VERSION=2.7 SCALA_VERSION=2.11
EsaScore	/opt/stag/esascore/esascore.sh /opt/stag/esascore/esascore_2.11-1.0.jar /opt/stag/esascore/esascore.conf	2.11_1.0
Modello ESA e file ausiliari	costruito utilizzando STAG /opt/stag/model/document_index /opt/stag/model/esa_model /opt/stag/model/lv_data_df /opt/stag/model/matrix_vl	wiki_2021_IT_NER
Packages Spark	com.johnsnowlabs.nlp:spark-nlp_2.11:2.6.0 org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.4 questi pacchetti e le relative dipendenze vengono scaricati dinamicamente quando viene fatto il run del container (e parte automaticamente esascore.sh)	
Porte	L'immagine espone la porta 4040 (SPARK WEB UI)	4040

3.4.3.1 Configurazione

Per cambiare le configurazioni del Docker (e.g. configurazioni Kafka e Spark) è possibile modificare il file ~/frontera/docker/esascore.env per poi ricreare un nuovo container con lo script custom ~/frontera/docker/run.sh (che usa esascore.env)
Le chiavi configurabili sono (il file esascore.env ne elenca esplicitamente solo le più comuni):

chiave	valore (esempio)	descrizione

Configurazioni relative a Kafka utilizzate per configurare il KafkaConsumer ed il KafkaProducer della libreria *org.apache.kafka.clients*		
KAFKA_SERVER	localhost:9092	<indirizzo del server>:<porta> *kafka.bootstrap.servers*
KAFKA_TOPIC_IN	enea-spider-esa	coda di ingresso *kafka.consumer.topic*
KAFKA_TOPIC_IN_GROUP_ID	esascore-docker	gruppo del consumer coda ingresso *kafka.group.id*
KAFKA_OFFSET_RESET	earliest latest	se cominciare a consumare i topic dalla coda di ingresso a partire dal primo non ancora assegnato al proprio consumer group (earliest) o dal successivo che arriva (latest) *kafka.auto.offset.reset*
KAFKA_TOPIC_OUT	enea-output-esa	coda di uscita *kafka.producer.topic*
KAFKA_MAX_POLL_RECORDS	20	massimo numero di record da processare in parallelo *kafka.max.poll.records* questo valore va **calibrato** sulla base delle caratteristiche della macchina, della lunghezza dei testi, numero link per pagine ed in generale delle performance che si vogliono ottenere
KAFKA_MAX_POLL_INTERVAL_MS	400000	*kafka.max.poll.interval.ms*
KAFKA_SESSION_TIMEOUT_MS	300000	*kafka.session.timeout.ms*
KAFKA_ENABLE_AUTO_COMMIT	false	*kafka.enable.auto.commit*
KAFKA_JSON_FIELDS	fingerprint,url,text,radius,type,request,key	lista separata da virgole senza spazi field del json di input da mantenere (`text` è sempre richiesto) e poi rimandare indietro nell’output è una chiave custom, non di libreria **N.B. tutti questi field dentro ai messaggi JSON devono essere *stringhe*, altrimenti il programma non riesce a fare il parsing del messaggio e restituisce errori.**

Configurazioni relative a Spark da modificare in base alle caratteristiche della macchina e alle “performance” richieste per il programma EsaScore cfr https://spark.apache.org/docs/2.4.8/configuration.html		
`SPARK_DRIVER_MEMORY`	`10G`	memoria assegnata al driver
`SPARK_DRIVER_CORES`	`2`	numero di core assegnati al driver
`SPARK_EXECUTOR_MEMORY`	`10G`	memoria assegnata all’executor
`SPARK_TOTAL_EXECUTOR_CORES`	`4`	numero di core assegnati all’executor
`SPARK_MEMORY_OFFHEAP_SIZE`	`10G`	*spark.memory.offHeap.size*
`SPARK_PARTITIONS`	`16`	numero “partizioni” dei dataframes imposta anche la variabile: *spark.default.parallelism*
`SPARK_DRIVER_MAXRESULTSIZE`	`0`	*spark.driver.maxResultSize*
`SPARK_RPC_MESSAGE_MAXSIZE`	`512MB`	*spark.rpc.message.maxSize*
Configurazioni relative ad EsaScore		
`ESA_DEBUG_MODE`	`true` `false`	se visualizzare messaggi di debug aggiuntivi sullo stdout
`ESA_FORCE_LOWERCASE`	`true` `false`	se forzare la trasformazione in lowercase del testo di input. Mantenere attiva questa opzione dovrebbe migliorare i risultati
`ESA_OUTPUT_TEXT`	`true`	se mantenere la colonna text nel JSON output. Da tenere attiva per poter popolare la DB table con le pages

Personalizzazioni ulteriori possono essere effettuate “scavalcando” con delle copie locali modificate i file esascore.conf ed esascore.sh dentro la cartella /opt/stag/esascore dentro al container.

3.4.3.1.1 Categorie

Le “categorie” corrispondono ai vari termini dell’ontologia e sono usate per l’analisi semantica (vedi sopra le sezioni dedicate). Il contenuto del file csv delle categorie caricato di default nell’immagine (/opt/stag/categories/categories.csv) è [presentato qui su tre colonne per renderlo più leggibile]

"title","text"		
"efficienza","efficienza" "produzione","produzione"	"obiettivo","obiettivo" "fotovoltaico","fotovoltaico"	"fattore","fattore" "impatto","impatto"

"consumo","consumo" "tecnologia","tecnologia" "sviluppo","sviluppo" "edificio","edificio" "settore","settore" "energia","energia" "investimento","investimento" "sostenibilità","sostenibilità" "intervento","intervento" "strategia","strategia" "mercato","mercato" "economia","economia" "gestione","gestione"	"fonte","fonte" "impianto","impianto" "costo","costo" "incentivo","incentivo" "misura","misura" "rete","rete" "riduzione","riduzione" "riscaldamento","riscaldamento" "risparmio","risparmio" "fabbisogno","fabbisogno" "società","società" "valore","valore" "biomassa","biomassa"	"valutazione","valutazione" "prezzo","prezzo" "distribuzione","distribuzione" "petrolio","petrolio" "emissione","emissione" "quantità","quantità" "combustibile","combustibile" "elettricità","elettricità" "certificazione","certificazione" "carbone","carbone" "certificato","certificato" "gas","gas" "servizi","servizi"

Questo file di categorie viene letto ogni volta che il container parte. Per scavalcare la lista delle categorie è quindi possibile sfruttare il meccanismo dei docker volumes, montando sul container un file locale della macchina host per scavalcare il file di categorie del container.
Ad esempio aggiungendo l'opzione -v quando di lancia docker container run::

```
-v local_categories_file.csv:/opt/stag/categories/categories.csv
```

Nello script ~/frontera/docker/run.sh c'è un esempio commentato di comando dedicato.

3.4.4 Altro

La cartella ~/frontera contiene la cartella ausiliaria packages con dentro ad esempio gli archivi .tgz dei componenti installati. N.B. eventuali file di configurazione contenuti in questi archivi potrebbero non essere configurati per l'ambiente babylon ed in caso di estrazione vanno riconfigurati adeguatamente.

3.5 Esecuzione Spider

3.6 Start/Stop/Reset

3.6.1 Apache Kafka

Per prima cosa è necessario che l'Apache Kafka sia attivo e con i relativi topics configurati. Se necessario, quindi, farlo partire manualmente.
Dopo aver verificato che non stia già girando, andare nella cartella ~/frontera/kafka ed utilizzare ad esempio lo script custom **start.sh** (che invoca i comandi della cartella bin/ dell'installazione di kafka puntando alla configurazione config/) per lanciare il Kafka Server e Zookeeper.
N.B. Lo script start.sh al momento lancia i due processi in background (solo con un & alla fine): quindi se si chiude la shell dovrebbero venire interrotti. Per renderli "longevi" va modificato lo script mettendo tipo il nohup (si veda per esempio la sintassi commentata dentro allo script).
Se i topics Kafka necessari non sono presenti, ricrearli tramite lo script **reset.sh**. Lo script utilizza i comandi bin/kafka-topics.sh --delete e poi --create per cancellare e ricreare i topics. L'operazione di cancellazione dei messaggi dei topics potrebbe però non venire completata se ci sono dei processi che sono ancora attivi su quei topics o se ci sono configurazione particolari (e.g. *Topic deletion … Note: This will have no impact if delete.topic.enable is not set to true*).
Per fermare Kafka e Zooker (da fare dopo aver fermato gli altri componenti) utilizzare ad esempio lo script custom stop.sh e verificare che i due processi siano effettivamente spenti (e.g. con il comando ps). N.B. lo script stop.sh utilizza le informazioni sui PID dei processi lanciati da start.sh e salvati dentro dei file nella sottocartella ~/frontera/kafka/pids.

3.6.2 Docker EsaScore

Se il container per EsaScore non è già presente (**docker container ls -a**) o se si sono cambiate le impostazioni (e.g. file esascore.env), ricrearne uno nuovo, ad esempio con lo script ~/frontera/docker/run.sh.
Se il container è già presente ma non è attivo, farlo ripartire con **docker container start <id>**
Per visualizzare quello che sta succedendo sul container è possibile fare *attach* ad esempio con il comando **docker container attach <id>** (attenzione che poi il Ctrl-C chiude il container) oppure usare le funzionalità di "log" di docker.
Il container si può fermare facendo Ctrl-C in modalità *attach* oppure con **docker container stop <id>**
Verificare che sia effettivamente interrotto con il comando **docker container ls -a**

3.6.3 Componenti Frontera

Se non è già attivo, far partire il virtualenv di frontera e.g. con **source ~/frontera/venv/bin/activate**
A questo punto spostarsi nella cartella ~/frontera/ec_listner ed utilizzare gli script custom.
Se la coda di crawling è vuota, inserire i seeds desiderati nel file seeds/seeds.txt ed usare **feed.sh** per aggiungerli alla coda. **N.B. gli url devono essere completi di http:// o https:// davanti, altrimenti l'inserimento dei feed restituisce errori.**
Quindi usare **start.sh** per far partire i vari componenti. In particolare il comando eseguito senza parametri fa partire in nohup (dopo averle spente se già attive) le seguenti componenti (redirigendo stdout e stderr su file di log e salvandosi il PID del programma):
converter: *Converter* (configurazione config/sw.py)

```
python -m ec_listner.converter --config config.sw
```

sw: *Strategy Worker* (con CrawlingStrategy custom e configurazione config/sw.py)

```
python -m frontera.worker.strategy --config config.sw --partition-id
0 --strategy 'ec_listner.strategy.EcListnerCrawlingStrategy'
```

scrapy: *Scrapy* (configurazione recuperata da scrapy.cfg, ec_listner/settings.py e config/spider.py)

```
scrapy crawl ec_listner
```

dbw_scoring: *DB worker* per lo scoring (config/dbw.py)

```
python -m frontera.worker.db --config config.dbw --partitions 0 --no-
batches --no-incoming
```

dbw_batches: *DB worker* per batches &co (config/dbw.py)

```
python -m frontera.worker.db --config config.dbw --partitions 0 --no-
scoring
```

Se si vuole resettare l'ambiente - **attenzione! cancella il DB ed i results.jl** - utilizzare lo script **reset.sh**

N.B. lo script reset.sh NON cancella i messaggi dai topic Kafka (bisogna usare i comandi dedicati Kafka topics per farlo): se si riparte senza averli puliti, i vari componenti continueranno a prendere "in carico" le cose in sospeso dal giro precedente.

Utilizzare lo script custom **follow.sh** per visualizzare (aggregati) i vari log di frontera (si veda il file README per maggiori dettagli).

Gli script **start.sh** ed **stop.sh** possono essere usati anche per far partire / interrompere un singolo componente passando il suo identificativo come primo argomento (sw, scrapy, dbw_batches, dbw_scoring, topics [opzionale per debug: attenzione che produce molto output])

Come descritto più sopra nella sezione dedicata, Scrapy si dovrebbe chiudere automaticamente dopo aver eseguito un numero di chiamate pari a `CLOSESPIDER_PAGECOUNT`, e va fatto ripartire manualmente (e.g. con **bash start.sh scrapy**) se si vuole riprendere il crawling.

Per interrompere i vari componenti frontera, eseguire lo script **stop.sh**

N.B. i componenti DBW impiegano un po' di tempo a chiudersi: va quindi verificato (e.g. con il comando ps) che effettivamente si siano chiusi tutti i componenti. Lo script stop.sh utilizza le informazioni sui PID dei processi lanciati da start.sh e salvati dentro dei file nella sottocartella ~/frontera/ec_listner/pids.

3.6.4 Logging

La componente frontera salva i suoi log nella cartella ~/frontera/ec_listner/logs; uno script custom (**follow.sh**) permette di visualizzarli aggregati.

La componente EsaScore stampa le informazioni sul suo stdout (utilizzare le funzionalità Docker di *attach* o di *log* per visualizzarlo). Come debug/introspezione di Spark, l'immagine Docker espone la SPARK WEB UI sulla porta 4040 del container.

La componente Kafka salva i log dentro ~/frontera/kafka/logs (redirezione output dal comando start.sh), dentro ~/kafka_2.13-2.8.0/logs e potenzialmente salva altre informazioni utili anche nella sua cartella temporanea (e.g. all'interno di /tmp/kafka o nome simile).

3.7 Output

Gli output delle operazioni di crawling (arricchiti da metadati come i punteggi di pertinenza semantici e lo "score" finale) sono salvati dentro il database db/ec_listner.sqlite. Al suo interno si trovano varie tabelle, tra cui "pages" (tabella custom con dentro le informazioni sulle pagine estratte) e "metadata" (tabella contenente le informazioni anche dei links estratti). Le informazioni aggiuntive sono salvate utilizzando il formato pickle. Per semplificare l'estrazione di questi dati c'è uno script custom **parsedb.py** (da eseguire dopo aver attivato il virtualenv frontera) che fa il dump in formato json di queste due tabelle rispettivamente in output/pages.json e output/metadata.json.

La componente Scrapy fa "append" dei testi delle pagine direttamente nel file output/results.jl; viene salvato tutto il testo del body così com'è (senza i tags). Vedi sezione dedicata per maggiori dettagli.

Invece, il testo salvato dentro la tabella pages è quello ritornato dal giro Esa (e.g. se è attivo il lowercase sui testi di input il testo salvato sarà lowercase)

4 Riferimenti bibliografici

1. M. Paukkeri, A. Peréz Garcìa Plaza, V. Fresno, R.M. Unanue, T. Honkela, "Learning a taxonomy from a set of text documents" Applied Soft Computing 12 (3) (2012), 1138-1148.
2. A. Weichselbraun, G. Wohlgennant, "Discovery and evaluation of non-taxonomic relations in domain ontologies", International Journal of Metadata, Semantics and Ontologies 4 (2009), 212-222.
3. Lamy Jean-Baptiste," Ontologies with python", (2021) Apress, Berkeley, CA.
4. https://github.com/JustGlowing/minisom, "Minisom- SOM implementation in python", (2021).
5. https://users.ics.aalto.fi/jhollmen/dippa/node7.html, "Self Organizing Map", (1996).
6. https://fasttext.cc/docs/en/crawl-vectors.html, "Pre-trained italian model", (2018)
7. E. Grave*, P. Bojanowski*, P. Gupta, A. Joulin, T. Mikolov, *Learning Word Vectors for 157 Languages*

5 Abbreviazioni ed acronimi

Acronimo	**Descrizione**
DB	Database
DBSCAN	Density-Based Spatial Clustering of Application with Noise
DBW	Database Worker
ESA	Explicit Semantic Analysis
OWL	Web Ontology Language
PID	Process ID
POS	Part Of Speech
SOM	Self Organizing Map
SW	Strategy Worker oppure Software (a seconda del contesto)
TF-IDF	Term Frequency–Inverse Document Frequency