



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA IN INFORMATICA

TESI DI LAUREA
IN
BASI DI DATI

Integrazione di KGs in ambito
giuridico per i casi di violenza sulle
donne

RELATORI:

Prof.ssa Claudia D'Amato

LAUREANDO:

Sara Padovano

CORRELATORI:

Prof. Nicola Fanizzi

Dott.ssa Fatima Zahra Amara

ANNO ACCADEMICO 2024 - 2025

Indice

1	Introduzione	1
1.1	Ambito	1
1.2	Scopo	2
1.3	Organizzazione tesi	2
2	Definizioni preliminari	5
2.1	Knowledge Graph	6
2.2	Web Semantico e Linguaggi di Rappresentazione e Interrogazione .	7
2.2.1	Web Semantico	7
2.2.2	RDF	7
2.2.3	RDFS	11
2.2.4	OWL	12
2.2.5	SPARQL	15
2.3	Ontologia	17
2.4	Linked e Open Data	18
2.5	Large Language Models	19
3	Stato dell'arte	21
3.1	Due KGs iniziali	21
3.1.1	KG con metodologia bottom-up	22
3.1.2	KG tramite LLM	24
3.2	Metodologie di integrazione di KGs	26
3.2.1	Primo approccio: Allineamento delle ontologie	26
3.2.2	Secondo approccio: Integrazione di KGs	27
3.3	AutoAlign	29

3.3.1	Modulo di embedding dei predicati	31
3.3.2	Modulo di embedding degli attributi	32
3.3.3	Modulo di embedding della struttura	33
3.3.4	Allineamento dell'entità	34
3.3.5	Risultati di AutoAlign	34
4	Approccio proposto	37
4.1	Pipeline	38
4.2	Applicazione	39
4.2.1	Reperimento KGs	39
4.2.2	Creazione del prototipo del grafo di prossimità	41
4.2.3	Creazione del prompt	43
4.2.4	Uso LLM	45
4.2.5	Allineamento dei tipi	47
5	Valutazione	49
6	Conclusioni e sviluppi futuri	53
	Bibliografia	55

Capitolo 1

Introduzione

Uno degli sviluppi più importanti e interessanti dato dall'intelligenza artificiale è proprio quello nell'ambito della giustizia predittiva. Si tratta di un settore in cui si sviluppano sistemi per supportare il giudice nel prendere decisioni, con l'obiettivo di velocizzare il processo decisionale, aumentarne l'efficienza e ridurre i costi.

Attualmente, le leggi e i casi sono fondamentalmente disponibili in formato testuale o HTML. Questo rende complicato sia il loro processing per supportare l'esperto nel processo decisionale sia la loro interrogazione.

Per questo la creazione di Knowledge Graph per rappresentare le fonti normative in modo strutturato risulta fondamentale. Tali grafi consentono di interrogare i dati strutturati, per esempio tramite SPARQL e SPARQL endpoint, oppure anche di associarli con altri dati esistenti in modo da poter ottenere più informazione rispetto a quella semplicemente pubblicata con il KG.

Il primo passo è quindi la costruzione di un KG. Nell'ambito di questa tesi, l'attenzione è stata rivolta al tema della violenza sulle donne, concentrandosi in particolare sulle sentenze messe a disposizione dalla Corte europea dei diritti dell'uomo.

1.1 Ambito

La violenza sulle donne è un fenomeno sociale intrinseco della nostra epoca. Le sue fondamenta si radicano in secoli di disuguaglianza di genere in cui la donna veniva posta in una condizione di inferiorità dal punto di vista sociale, economico e legislativo rispetto all'uomo.

Essa può assumere diverse sfaccettature: dalla violenza fisica e sessuale a quella psicologica ed economica, fino ad arrivare allo stalking e, in casi estremi, anche all'omicidio.

Gli effetti della violenza sulle vittime possono avere conseguenze devastanti dal punto di vista psicologico, non considerando eventuali percosse fisiche e danni economici. La stessa dignità della vittima viene lesa. Si arriva quindi a una violazione dei diritti di base della persona e alla perdita della sicurezza che la società dovrebbe garantire.

1.2 Scopo

Nell'ambito della giustizia predittiva, con particolare attenzione ai casi di violenza sulle donne, sono già stati creati due Knowledge Graphs (abbreviazione KGs), a partire dalle pronunce emesse dalla Corte europea dei diritti dell'uomo (ECHR).

Lo scopo di questa tesi è quello di integrare i due Knowledge Graphs creati precedentemente in questo contesto. I due Knowledge Graphs sono stati costruiti con due metodologie differenti: uno con un approccio bottom-up, l'altro mediante l'impiego di Large Language Models (LLM).

Nonostante entrambi i KGs si basino sugli stessi dati di partenza, l'adozione di approcci diversi ha portato ad avere due KGs con due ontologie diverse. Al fine di avere un unico KG interrogabile e onnicomprensivo, in questa tesi si è investigato il problema circa l'integrazione/fusione dei due KGs in un'unica risorsa.

1.3 Organizzazione tesi

L'organizzazione dei capitoli è la seguente. Nel Capitolo 2 si forniranno tutte le definizioni preliminari e di base per la comprensione della tesi. Ossia si definiranno i concetti di Knowledge Graph e ontologia in quanto alla base dei KGs da integrare, oltre che dello studio dell'integrazione dei KGs stessi. Si tratteranno poi RDF, RDFS, OWL per la creazione di risorse e SPARQL per la loro interrogazione. Infine si accenneranno agli LLMs visto il loro ruolo centrale all'interno di questa tesi.

Nel Capitolo 3 si descriveranno le metodologie con cui sono stati creati i KGs da noi utilizzati per avere una panoramica di quello che è stato fatto e di qual è

il punto di partenza. Anche per capire in modo più completo sul perchè questi due KGs necessitino di integrazione. Si andranno poi ad analizzare i due approcci proposti per l'integrazione dei KGs, spiegando il problema dell'integrazione e qual è il sistema scelto per raggiungere il nostro scopo.

Nel Capitolo 4 si andrà a dire di quale step ci si è effettivamente occupati all'interno del sistema. Se ne descriveranno i vari step per arrivare al risultato della nostra applicazione, ossia il grafo di prossimità dei predicati.

Nel Capitolo 5 si valuterà quello che è stato fatto dando due valutazioni, abbastanza collegate tra di loro, che permettano di capire il livello di quello che è stato fatto, traendone le conclusioni.

Infine nel Capitolo 6 si andranno a tirare le conclusioni di quanto detto all'interno di questa tesi, riassumendo quello che è stato fatto e proponendo degli sviluppi futuri.

Capitolo 2

Definizioni preliminari

In questo capitolo verranno fornite le definizioni fondamentali per la comprensione della tesi. In particolare, nella sezione 2.1 si presenterà il concetto di Knowledge Graph, la struttura dati usata in questo progetto. Nella sezione 2.2 si presenterà il concetto di tripla, di RDF con la sua estensione RDFS e altri linguaggi di rappresentazione e interrogazione come OWL e SPARQL. Il concetto di tripla è fondamentale poiché alla base di RDF che, a sua volta, è il linguaggio di standardizzazione usato nel Web Semantico [7]. Si passerà quindi alla visione di RDFS come estensione di RDF. Si vedrà poi OWL come linguaggio per la descrizione di ontologie spiegate poi nella sezione 2.3. Infine si vedrà SPARQL come linguaggio di interrogazione per la creazione di query interrogabili per grafi RDF. Nella sezione 2.3 si tratterà il concetto di ontologia come schema alla base dei Knowledge Graph, con il linguaggio OWL per la descrizione di ontologie. Nella sezione 2.4 si darà una definizione di Linked Open Data, che definiscono il Web Semantico, e gli standards per la pubblicazione di dati che hanno ispirato anche una maggiore standardizzazione e accessibilità dei Knowledge Graphs. Infine nella sezione 2.5 si parlerà di Large Language Models (LLMs) dandone una breve definizione e introduzione vista l'importanza e il sempre crescente utilizzo di questi modelli legati a Knowledge Graphs e ontologie.

2.1 Knowledge Graph

Un **Knowledge Graph (KG)** è un modo per rappresentare una base di conoscenza. Esso si basa sulle relazioni che intercorrono tra gli individui e gli individui stessi. Un KG è un grafo orientato con gli archi etichettati. Formalmente viene espresso sotto forma di tripla $G = (V, E, L)$ dove [15]:

- **V**: è l'insieme dei nodi. Un nodo rappresenta un'entità, oppure altri valori come stringhe e numeri;
- **E**: è l'insieme degli archi. Rappresentano quindi le relazioni che connettono le entità;
- **L**: è l'insieme delle etichette associate a nodi e archi e che definiscono le relazioni tra di essi.

Nella figura 2.1 viene riportato un esempio di Knowledge Graph.

In particolare, possiamo notare come questo KG descriva degli eventi, rappre-

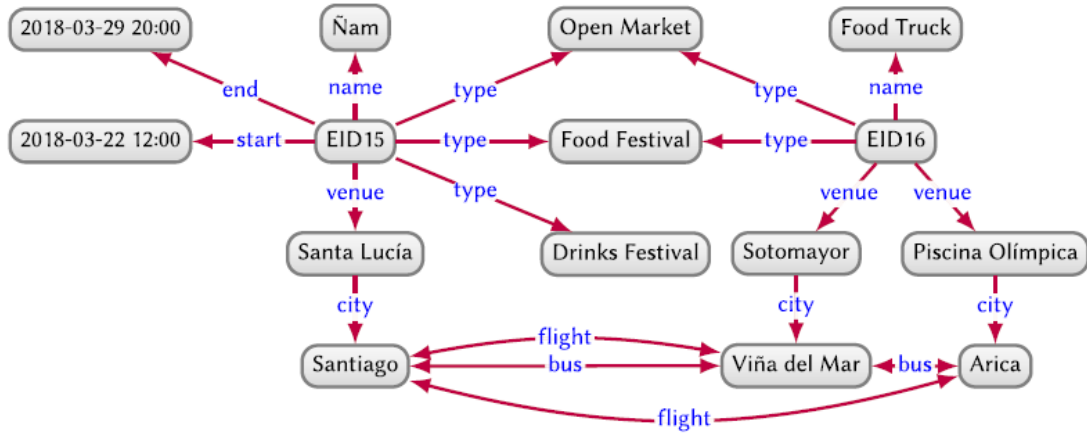


Figura 2.1: Knowledge Graph orientato etichetta che descrive eventi e la loro sede [15]

sentati dalle entità *EID15* e *EID16*, collegati ad altre entità, come *Santa Lucia* o *2018-03-22 12:00*, attraverso degli archi orientati dove l'etichetta indica la relazione che intercorre tra le due entità; un esempio di etichetta è *end* o *type*.

Il KG, quindi, ci descrive i due eventi definendone il nome $name(EID16, Food Truck)$, per *EID15* anche la data di inizio $start(EID15, 2018-03-22 12:00)$ e fine dell'evento $end(EID15, 2018-03-29 20:00)$, la tipologia di evento $type(EID15, Open$

Market), la loro sede *venue*(*EID16*, *Piscina Olimpica*), la città in cui si trova la sede *city*(*Santa Lucia*, *Santiago*) e in che modo si possono raggiungere le varie città *flight*(*Santiago*, *Arica*) e *flight*(*Arica*, *Santiago*).

Per riportare le relazioni rappresentate nel Knowledge Graph ho usato la rappresentazione tramite triple, in particolare la notazione funzionale nella forma *verb(subject, object)*.

2.2 Web Semantico e Linguaggi di Rappresentazione e Interrogazione

2.2.1 Web Semantico

Il **Web Semantico** (o **Semantic Web**) [7] nasce con l'obiettivo di rendere i contenuti leggibili dalle macchine e riutilizzabili, permettendo così una gestione più automatizzata e intelligente delle informazioni.

A livello generale, il Web Semantico può essere considerato un'estensione dell'attuale Web, progettata per consentire la creazione, la condivisione e il riutilizzo intelligente dei contenuti in un formato che le macchine possano interpretare e sfruttare autonomamente. [14]

In questo contesto, l'RDF (Resource Description Framework) è una tecnologia semantica fondamentale, in quanto fornisce un modello di dati standardizzato per rappresentare e scambiare informazioni in modo strutturato e leggibile dalle macchine.

2.2.2 RDF

Per rappresentare le relazioni (come visto sopra) si possono usare le **triple**. La rappresentazione tramite triple è nella forma *individuo-proprietà-valore* o *soggetto-predicato-oggetto*. Esse vengono rappresentate sotto forma di una relazione *prop* nella forma *prop(Ind, Prop, Val)*. [23] Ecco perché una tripla viene vista come una semplice frase composta da tre termini. Il concetto di tripla è alla base del modello RDF usato nel web semantico e standardizzato dal W3C.

RDF è un modello e linguaggio standard per la rappresentazione e lo scambio dei dati e un linguaggio per identificare le risorse sul web. Supporta la fusione

di dati anche con schemi sottostanti diversi con anche la modifica degli schemi nel tempo senza che si apportino ulteriori modifiche. [35] RDF è stato creato non tanto affinché le informazioni vengano mostrate agli utenti, ma soprattutto per contesti in cui l'informazione deve essere processata e scambiata tra le varie applicazioni senza subire perdite. L'abilità di scambiare informazioni rende l'informazione disponibile anche per applicazioni e contesti diversi da quelli per cui è stata creata. [31].

Il modello dei dati RDF rappresenta l'informazione sotto forma di tripla, quindi dalla forma *soggetto-predicato-oggetto*. Un esempio di tripla RDF può essere la frase: "Riccardo ha scritto un saggio". Dove Riccardo è il soggetto, ha scritto è il predicato e saggio è l'oggetto.

I dati in RDF possono appartenere a tre categorie [3] [14]:

- **IRI**: sono identificatori globali che servono a identificare univocamente una qualsiasi risorsa (sul web). Gli IRI, negli ultimi anni, hanno sostituito il termine *URI* generalizzandolo (ad esempio <http://dbpedia.org/resource/Lemon>);
- **Letterali**: insieme di valori lessicali che possono essere:
 - **Letterali semplici**: insieme di stringhe semplici come "Hello World";
 - **Letterali tipizzati**: che mettono insieme una stringa con un tipo come "2"^^xsd:integer;
- **Blank nodes**: identificatori usati per identificare risorse anonime che non corrispondono a risorse sul web e sono preceduti dall'underscore. Non possono essere riferiti al di fuori dello scope in cui vengono dichiarati; essi infatti servono come identificatori a livello locale. Un esempio può essere `_:bnode1`.

Nelle triple RDF il soggetto può essere un IRI o un blank node, il predicato può essere solo e soltanto un IRI e l'oggetto può essere tutte e tre le tipologie di dati. Inoltre, RDF non incorpora l'UNA (l'assunzione del nome unico, Unique Name Assumption). Quindi in RDF più nomi possono riferirsi a una stessa risorsa; questo è dovuto anche e soprattutto alla natura del web, dove due autori o siti possono riferirsi a uno stesso concetto usando nomi diversi.

RDF viene spesso usato per concettualizzare informazioni come grafi orientati etichettati dove il soggetto e l'oggetto della tripla RDF sono uniti da un arco etichettato rappresentante il predicato della tripla [14] [37].

Nell'immagine sottostante si può vedere la rappresentazione grafica di una tripla a grafo del concetto di RDF graph:

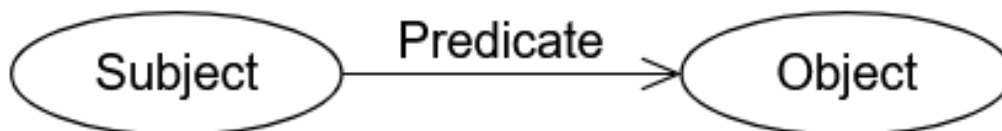


Figura 2.2: Un grafo RDF con due nodi (Soggetto e Oggetto) e una tripla che li collega (Predicato) [37]

Per rappresentare un grafo RDF in un file esistono diverse sintassi, ma una di quelle più usate è **Turtle** [14, 39]. Turtle punta a essere una sintassi precisa e intuitiva, consentendo l'utilizzo di abbreviazioni per caratteristiche frequenti di RDF. Per esempio, preso il seguente semplice knowledge graph Figura 2.2: Per

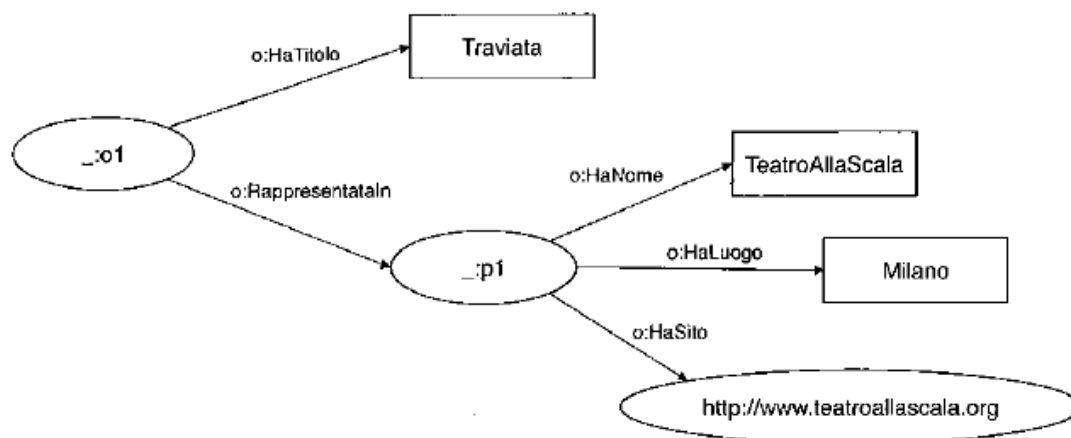


Figura 2.3: Grafo RDF [3]

representare questo knowledge graph, la sua sintassi Turtle è riportata di seguito. Si consideri che `_:o1` e `_:p1` sono blank node e corrispondono rispettivamente a un'opera e a un teatro di Milano. In Turtle le relazioni di queste due entità verranno scritte nel seguente modo¹:

```
@prefix o: <http://www.polimi.it/ceri/opera>.
```

¹Esempio preso da [3]

```
_:o1 o:HaTitolo 'Traviata';  
      o:RappresentataIn _:p1.  
_:p1 o:HaNome 'TeatroAllaScala';  
      o:HaLuogo 'Milano';  
      o:HaSito <https://www.teatroallascala.org>.
```

Si può notare come nella sintassi Turtle, se due triple condividono lo stesso soggetto si usa il separatore ";" evitando di ripetere il soggetto. Altrimenti se si deve dichiarare una nuova istanza con soggetto diverso, si usa "." oppure se si ha lo stesso soggetto e predicato si usa ",". Alla fine, quali sono i vantaggi di RDF che ci hanno portato al suo utilizzo?

I vantaggi che hanno consolidato l'utilizzo di RDF come modello dei dati largamente diffuso sono molteplici. Tra i più importanti riscontriamo [4]:

- **Interoperabilità dei dati:** vantaggio più importante di RDF è la sua interoperabilità consentendogli di catturare metadati e informazioni di risorse strutturate, semistrutturate e non strutturate. Questo lo rende universale per la rappresentazione di dati;
- **Flessibilità nello schema:** la flessibilità nella mappatura, aggregazione di dati indipendentemente dagli schemi sottostanti, porta RDF a poter lavorare con i dati senza esser legato agli schemi sottostanti;
- **Fluidità:** RDF provvede un framework di rappresentazione dati e schemi che si adatta in base alla tipologia di dati esistenti e alle strutture note. Come si scoprono nuovi dati o attributi essi possono essere aggiunti al modello esistente senza apportare alcuna modifica allo schema sottostante;
- **Adattibilità:** il punto precedente porta a notare un altro punto di forza, ossia l'adattabilità di RDF in contesti di mondo incompleto o parzialmente incompleto, oppure consente un'evoluzione della complessità del mondo mantenendo le strutture e osservazioni precedentemente fatte.

RDF però nella sua forma base consente solo la definizione di forme più semplici di conoscenza come triple. Ad esempio non è possibile tipizzare soggetti ed oggetti di predicati o specificare gerarchie di tipi/classi. Per superare tali limitazioni sono stati introdotti linguaggi più espressivi, come sue estensioni, quali RDFS, e OWL

illustrati di seguito. Per questo motivo è stata introdotta una sua estensione, RDFS, per superare queste limitazioni.

2.2.3 RDFS

RDF Schema, o **RDFS**, è un'estensione di RDF pensata dal W3C che provvede meccanismi per la descrizione di gruppi di risorse collegate e delle relazioni che intercorrono tra esse. [38] Esso aggiunge struttura e proprietà alle istanze RDF, i *metadati*. Si definisce quindi lo schema dell'istanza. [3]

RDFS quindi introduce sei costrutti fondamentali²:

1. **Classi**: definita come *rdfs:Class*. Una classe RDFS consente di definire insieme di risorse di istanza RDF che hanno caratteristiche in comuni. Per esempio considerando di voler introdurre una classe *Compositore* all'esempio dell'opera del teatro di Milano precedente possiamo definire così:

```
@prefix o: <http://www.polimi.it/ceri/opera>.
@prefix rdf: <https://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <https://www.w3.org/2000/01/rdf-schema#>.
o:Compositore rdf:type rdfs:Class.
_:Wagner rdf:type o:Compositore.
```

2. **Proprietà**: definita come *rdf:Property* che è la classe delle proprietà RDF ed è un'istanza di *rdfs:Class*. Quando si definisce una proprietà bisogna definirne anche il dominio (il tipo del soggetto) e il range (il tipo dell'oggetto);
3. **Dominio**: definito come *rdfs:domain*. È un'istanza di *rdf:Property* e viene usata per definire che qualsiasi risorsa data una proprietà è un'istanza di una o più classi;
4. **Range**: definito come *rdfs:range* è anch'esso un'istanza di *rdf:Property* e viene usato per definire che i valori di una proprietà sono istanze di una o più classi;

²Tutti gli esempi presentati per i costrutti sono presi da [3]

Vogliamo definire adesso un esempio per vedere *rdf:Property* con *rdfs:domain* e *rdfs:range* sulla riga dell'esempio precedente si aggiungono le seguenti istruzioni all'esempio della classe:

```
o:nome a rdfs:Property;
      rdfs:domain o:Compositore;
      rdfs:range rdfs:Literal.
_:Ring a o:Opera.
o:CompostoDa a rdfs:Property;
      rdfs:domain o:Opera;
      rdfs:range o:Compositore.
```

5. **SottoClasse:** definita come *rdfs:subClassOf* è sempre un'istanza di *rdf:Property* e consente di definire che tutte le istanze di una classe sono anche istanze di un'altra. Si parla quindi di generalizzazione tra classi portando in RDF anche il concetto di ereditarietà che possiamo vedere nel seguente esempio in cui la classe *Compositore* dichiarata negli esempi precedenti verrà generalizzata nella classe *Artista* (per logica e per il concetto di ereditarietà si deduce che così facendo la classe compositore avrà le proprietà della classe artista):

```
o:Artista rdf:type rdfs:Class.
o:Compositore rdfs:subClassOf o:Artista.
```

6. **SottoProprietà:** definita come *rdfs:subPropertyOf* istanza di *rdf:Property* che definiscono che tutte le risorse collegate a una proprietà sono collegate all'altra.

2.2.4 OWL

I linguaggi RDF e RDFS presentano delle limitazioni, come [2]:

- *ambito locale delle proprietà:* in RDFS non si possono applicare restrizioni sul range applicabile solo su alcune classi;
- *disgiunzione delle classi:* in RDFS possiamo solo dichiarare il concetto di sottoclasse e non anche di disgiunzione tra classi;

- *combinazioni booleane di classi*: non si può costruire una classe dalla combinazione di altre classi data unione, intersezione e complemento;
- *restrizioni di cardinalità*: non ci sono restrizioni su quanti valori una proprietà può avere;
- *caratteristiche specifiche delle proprietà*: non sono definite proprietà come quella di transitività, di unicità e inversa.

Per questo serve un linguaggio ontologico più espressivo e che superi le limitazioni date da RDF e RDFS e che consenta di descrivere formalmente il significato della terminologia usata. **OWL** è un linguaggio sviluppato per il Web ed utilizzato per rappresentare conoscenza complessa su oggetti, gruppi di oggetti e le relazioni che intercorrono tra di essi. [33]

OWL descrive domini in termini di [23]:

1. **Individui**: le cose nel mondo che vengono descritte;
2. **Classi**: insiemi di individui. Una classe è l'insieme di tutte le cose potenziali o reali che potrebbero essere in quella classe;
3. **Proprietà**: usata per descrivere relazioni binarie tra individui e altri individui o valori.

OWL è composto da tre sottolinguaggi che passano dalla versione Lite a quella Full, diventando più generali ed espressivi [2] [29]:

1. **OWL Lite**: supporta per la creazione di una gerarchia di classificazione e semplici restrizioni. Rispetto ai due successivi sottolinguaggi è meno espressivo ma dà una più facile implementazione, consente una migrazione di tassonomia più veloce ed è più facile da comprendere;
2. **OWL DL**: consente di avere una maggiore espressività contenendo la complessità computazionale sacrificando una maggiore compatibilità con RDF data invece dalla versione Full;
3. **OWL Full**: consente sempre espressività come OWL DL con la libertà sintattica di RDF ma non garantendo efficienza computazionale.

I costrutti principali introdotti da OWL sono [14] [30]:

- *owl:equivalentClass* quando due classi risultano essere equivalenti, i membri di una classe lo sono anche della seconda e viceversa;
- *owl:disjointWith* quando l'intersezione tra due classi è vuota, ossia non hanno membri in comune;
- *owl:equivalentProperty* quando due proprietà mettono in relazione esattamente gli stessi membri, sono quindi equivalenti tra di loro;
- *owl:disjointPropertyWith* quando due proprietà non potranno mai mettere in relazione gli stessi membri;
- *owl:inverseOf* quando una proprietà mette in relazione esattamente gli stessi membri di un'altra proprietà ma nel senso opposto, queste due proprietà risultano quindi inverse;
- *owl:TransitiveProperty* quando si definisce una proprietà transitiva, P. Data una coppia (x,y) istanza di P e la coppia (y,z) anch'essa istanza di P allora si può dire che anche (x,z) è istanza di P;
- *owl:SymmetricProperty* quando si definisce una proprietà simmetrica, P. Data una coppia (x,y) istanza di P, anche la coppia (y,x) sarà istanza di P;
- *owl:sameAs* quando si vuole indicare che due risorse identificano la stessa cosa;
- *owl:differentFrom* quando due risorse non possono essere la stessa cosa e che quindi si devono necessariamente riferire a individui diversi;
- *owl:FunctionalProperty* quando si definisce una proprietà funzionale dove si dice che la proprietà può avere un solo valore y per un'istanza x. Se vi sono più valori devono essere valori compresi in una relazione *owl:sameAs*;
- *owl:InverseFunctionalProperty* quando l'oggetto di una proprietà può determinare unicamente il soggetto. Ossia un valore y può essere il valore della proprietà P per una singola istanza x (non può esistere più di una coppia con y come elemento).

Come abbiamo visto, quindi, OWL è un linguaggio molto importante nel Web Semantico che consente la definizione di ontologie e una maggiore espressività, espandendo i linguaggi RDF e RDFS.

2.2.5 SPARQL

Per interrogare i dati scritti in RDF/RDFS, il W3C ha introdotto un altro linguaggio, **SPARQL**. Tramite la creazione di query SPARQL, è possibile interrogare il grafo RDF per estrarre informazioni o anche filtrarle. Per ritrovare le soluzioni, SPARQL applica un pattern matching tra le triple della query di interrogazione (graph pattern nel caso più generale) e quelle del grafo RDF.

Ad alto livello, una query SPARQL è composta da cinque parti [14] [3] [32]:

1. **Dichiarazioni *PREFIX***: consentono la definizione di prefissi per gli URI da usare come shortcuts nella query. Si dichiarano nel seguente modo:

PREFIX o:<<https://www.polimi.it/ceri/opera>>

2. **Clausola del dataset**: specifica la porzione dati su cui verrà applicata la query, dichiarata tramite le clausole *FROM* e *FROM NAMED*. La clausola *FROM* viene poi seguita da un IRI da cui leggere dati mentre la clausola *FROM NAMED* viene seguita da un URI in cui è presente un grafo RDF a cui viene anche associato un nome;
3. **Clausola di risultato**: specifica che tipo di query SPARQL viene eseguita e cosa dovrebbe essere restituito. I tipi possibili sono:
 - *SELECT*: ritorna la lista dei valori associati alle variabili. Può essere seguito anche da *DISTINCT* se non si vogliono avere duplicati nel risultato;
 - *CONSTRUCT*: consente di costruire un nuovo grafo RDF dalle triple risultanti della query;
 - *ASK*: consente di vedere se esiste almeno una tupla di binding per la query fatta, consente quindi di vedere se la query ha un risultato non nullo;

- *DESCRIBE*: consente di estrarre tutta l'informazione legata alle risorse che soddisfano la query
4. **Clausola della query**: è la definizione della parte centrale della query introdotta dalla clausola *WHERE*. Consente di esprimere condizioni di pattern matching tra la query e il grafo RDF interrogato. All'interno della clausola possono essere aggiunte variabili SPARQL introdotte dal simbolo ?. Si possono anche dichiarare delle clausole opzionali quali:
- *FILTER*: filtra le tuple di binding in base al predicato specificato, solo le tuple che lo soddisfano vengono poi restituite;
 - *UNION*: consente l'unione di binding prodotti da diversi graph patterns;
 - *OPTIONAL*: consente di dichiarare all'interno della clausola *WHERE* variabili non sempre legate ad avere un risultato e che quindi possono assumere il valore NULL
5. **Modificatori della soluzione**: consentono di ordinare, dividere o impaginare i risultati. In particolare abbiamo:
- *ORDER BY*: consente di ordinare i risultati in base a una variabile. Può essere poi seguito dai termini *ASC* per indicare un ordine ascendente o *DESC* per l'ordine discendente;
 - *LIMIT*: consente di specificare un numero intero non negativo che indica il numero massimo di risultati che si vuole restituire;
 - *OFFSET*: anche in questo caso consente di identificare un numero intero non negativo che, in questo caso, identifica il numero di risultati da ignorare

La combinazione di *OFFSET* e *LIMIT* consente di creare una forma di impaginazione.

Un esempio di query SPARQL, date le informazioni sopra, è il seguente³:

³Esempio preso da [3]

```
PREFIX o: <http://www.polimi.it/ceri/opera>
SELECT ?t, ?p
FROM <http://www.polimi.it/ceri/ring.rdf>
WHERE{
    ?o1 o:haPersonaggio ?p .
    ?o2 o:haPersonaggio ?p .
    ?o1 != ?o2 .
    ?o1 o:haTitolo ?t .
    ?o2 o:haTitolo 'Valchiria' .
    ?o2 o:haDurata ?d .
    FILTER (?d > 300)
}
ORDER BY DESC(?t)
```

Nelle versioni più recenti di SPARQL sono state introdotte anche altre operazioni utili come quella di aggregazione con le clausole *GROUP BY* e *HAVING* oppure anche le negazioni e le sotto-query.

Certamente tra gli aspetti avanzati più interessanti possiamo trovare quello dell'entailment. Nell'entailment è possibile estendere la valutazione della query includendo le triple implicate nell'istanza in base all'entailment del linguaggio. Un altro aspetto è anche quello dell'interoperabilità dato dall'introduzione della clausola *SERVICE* che consente quindi l'integrazione di più fonti di dati. [3] [34]

2.3 Ontologia

Un'**ontologia** è una specificazione dei significati dei simboli in un sistema. Specifica gli individui e le relazioni che esistono e la terminologia usata per identificarli. [23] Essa è quindi una rappresentazione concreta e formale di cosa significhino i termini nel dominio in cui vengono usati. [15] Un Knowledge Graph è quindi una realizzazione concreta della struttura e delle relazioni data un'ontologia.

Un'ontologia solitamente consiste in [23]:

- Un vocabolario di categorie delle cose che una base di conoscenza vorrebbe rappresentare;

- Un'organizzazione delle categorie (come *subClassOf* e *subPropertyOf*);
- Un insieme di assiomi che restringono la definizione di alcuni simboli per riflettere meglio il loro vero significato.

Quando un'ontologia fa riferimento a un particolare dominio si parla di **ontologia di dominio** o **domain ontology** (che rappresentano la maggior parte delle ontologie esistenti). Per scrivere ontologie che consentano la condivisione di conoscenza e l'interoperabilità [15] ci sono delle linee guida i cui punti più importanti sono [23]:

- Se possibile usare ontologie già stabilite;
- Se esiste un'ontologia che non corrisponde completamente, importarla e aggiungere ciò di cui si ha bisogno;
- Un'ontologia si deve integrare con quelle vicine;
- Seguire convenzioni sui nomi.

Il linguaggio usato per la descrizione delle ontologie è OWL, descritto nella sotto-sezione 2.2.4.

2.4 Linked e Open Data

Per **linked data** ci riferiamo a un insieme di risorse disponibili sul web descritte da triple RDF e collegate tra loro tramite link. Si riferisce quindi a un insieme di best practices per la pubblicazione dei dati sul Web. [3] [36].

Le pratiche più importanti per rendere i dati interconnessi e navigabili sul Web sono [6]:

1. Identificare le risorse tramite URI. Fondamentale per il concetto stesso di Semantic Web;
2. Usare URI HTTP per ricercare quei nomi;
3. Ricercare URI fornendo informazioni tramite gli standards (SPARQL, RDF);
4. Includere links ad altre URI. Fondamentale per poter approfondire, espandere un argomento e per connettere i dati nel Web.

Altri principi per i Linked Open Data sono:

1. Dati disponibili sul Web con licenza libera;
2. Dati strutturati disponibili per essere letti dalle macchine;
3. Usare formati non proprietari (come CSV);
4. Usare gli standards del W3C per identificare le risorse e renderle navigabili;
5. Linkare i propri dati a quelli di altri utenti per creare contesto.

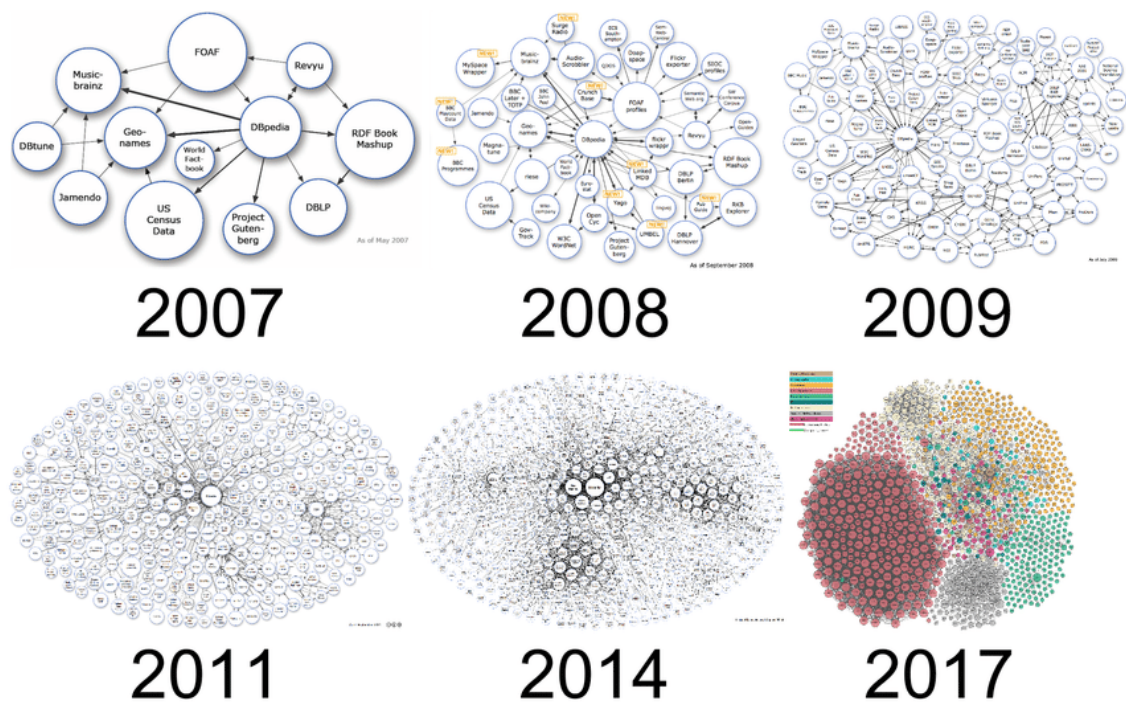


Figura 2.4: Crescita dei Linked Open Data dal 2007 al 2017 [13]

Come si può vedere dalla Figura 2.4, i Linked Open Data hanno acquisito sempre maggiore importanza negli ultimi anni, risultando essenziali per connettere il web semantico e gettare le basi per knowledge graph standardizzati e accessibili.

2.5 Large Language Models

I Large Language Models (LLM) sono modelli linguistici statistici di grandi dimensioni, pre-addestrati e basati su reti neurali. Essi si riferiscono specialmente

ai modelli linguistici basati su trasformatori. Rispetto ai modelli linguistici pre-addestrati (PLMs) molto più generali rispetto ai primi modelli linguistici neurali, gli LLMs risultano essere molto più grandi in dimensioni, hanno una maggiore comprensione del linguaggio e abilità generative. Inoltre, hanno delle capacità emergenti come (limitato) apprendimento in contesto anche con piccoli esempi oppure la capacità di affrontare nuovi compiti senza esempi specifici. Tutto questo rende possibile la creazione di agenti di intelligenza artificiale basati su LLM. [20]

Negli ultimi anni, grazie anche all'avvento di ChatGPT, sono diventati molto diffusi e il loro utilizzo è stato sperimentato anche in ambiti come la creazione di ontologie e, in particolare, per l'integrazione di KGs esistenti.

Capitolo 3

Stato dell'arte

In questo capitolo si introdurranno i KGs iniziali da integrare, dandone una breve spiegazione delle metodologie utilizzate per costruirli. Ricordiamo che entrambi i Knowledge Graphs sono derivati dalle sentenze della Corte europea dei diritti dell'uomo (ECHR) con focus sui casi di violenza sulle donne.

Nella sezione 3.1 daremo un breve accenno alle metodologie utilizzate per la costruzione dei due KGs. Rispettivamente nella sottosezione 3.1.1 e nella sottosezione 3.1.2, verranno trattate la metodologia bottom-up per la creazione del primo KG e l'utilizzo di LLM per il secondo. Nella sezione 3.2 si andrà ad esplorare le varie metodologie di integrazione di KGs considerate, andando poi nella sezione 3.3 a definire in maniera generale l'approccio scelto.

3.1 Due KGs iniziali

In questa sezione si descriveranno le metodologie utilizzate per la creazione dei due KGs iniziali: bottom-up e tramite LLM. Le ontologie prese in considerazione per la standardizzazione e l'interoperabilità dei due KGs per i casi di violenza sulle donne sono [10]:

- **European Law Identifier (ELI)**: è lo standard creato per identificare i documenti legislativi per gli stati dell'Unione Europea (preferibile ad ECLI, descritto sotto, per testi legislativi);

- **European Case Law Identifier (ECLI)**: definisce un identificatore standard per la giurisprudenza europea. L'identificatore è provvisto di un codice identificativo comune tra tutti gli Stati membri (preferibile ad ELI per decisioni legali);
- **EuroVoc**: non è un'ontologia come le precedenti citate ma un thesaurus che per classificare i documenti dell'Unione Europea in categorie per facilitarne la ricerca.

3.1.1 KG con metodologia bottom-up

Il primo KG preso in considerazione è stato realizzato con la metodologia *bottom-up*, processo proposto da Tamašauskaitė e Groth, in cui la conoscenza viene estratta dai dati per crearne l'apposita ontologia. Questa metodologia è opposta alla *top-down* in cui si ha il processo inverso, ossia si definisce un'ontologia e da questa si estrae conoscenza [28].

Il processo di sviluppo del Knowledge Graph si divide in sei step fondamentali: identificazione dei dati, costruzione dell'ontologia, estrazione della conoscenza, processing della conoscenza, costruzione del KG e mantenimento.

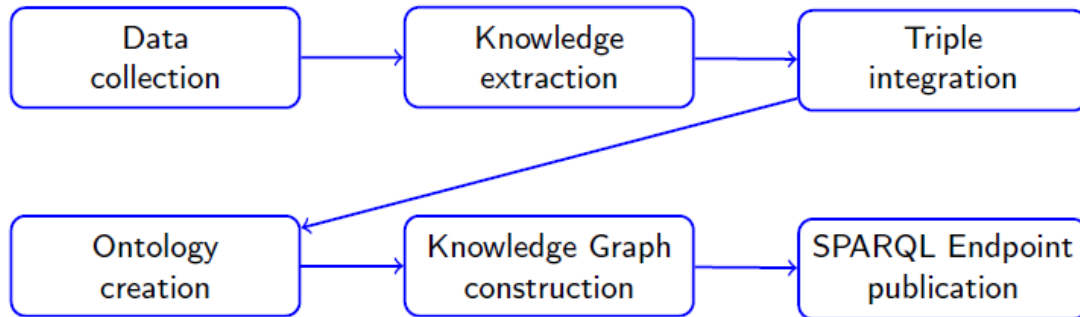


Figura 3.1: Pipeline utilizzata per la costruzione del primo KG

Da questi step fondamentali è poi scaturita l'effettiva pipeline, ossia le fasi, per la costruzione del primo Knowledge Graph. Le fasi adottate, visibili in Figura 3.1, sono [24]:

1. **Collezione dei dati**: vengono presi in considerazione i documenti del sito di riferimento selezionati da esperti di diritto internazionale, in particolare

quelli in inglese. Si sono quindi reperiti i file PDF e il codice HTML del sito tramite Selenium per ottenere dettagli aggiuntivi non presenti nei file PDF. Ogni sentenza è poi elaborata per la fase successiva;

2. **Estrazione della conoscenza:** in questa fase si considera una sentenza per volta sfruttando i metadati presenti nella sezione Case Details del sito web della Corte europea dei diritti dell'uomo (ECHR), in particolare si considerano solo i file HTML poichè contengono tutti i metadati necessari mentre non è possibile estrarre triple dal documento. Infatti i dati estratti in questa fase vengono usati per generare triple RDF. Da qui l'utilizzo della libreria RDFLib consente la serializzazione in file Turtle;
3. **Integrazione delle triple e eliminazione delle ridondanze:** il passo successivo è quello poi di eliminare ridondanze e consentire integrazione delle triple. Quest'operazione non è stata necessaria grazie a RDFLib che in fase di combinazione delle triple ha eliminato automaticamente le ridondanze;
4. **Creazione dell'ontologia:** questa fase inizia con la formulazione delle *competency questions*, ossia domande guida usate nella progettazione di ontologie o knowledge graph per chiarire e verificare quali informazioni il sistema deve essere in grado di rappresentare e rispondere. Si tratta di domande in linguaggio naturale che delineano e limitano l'ambito della conoscenza rappresentata in un'ontologia. [40] Visto che però non a tutte le competency questions è stato possibile rispondere in base ai dati che erano stati raccolti, sono stati introdotti nuovi concetti e proprietà;
5. **Costruzione e visualizzazione del KG:** il Knowledge Graph finale è infine stato costruito dalla combinazione di triple derivate da varie sentenze e dal collegamento con risorse esterne come Wikidata;
6. **Querying tramite Endpoint SPARQL:** come fase finale è stato infine creato un Endpoint SPARQL per consentire l'interrogazione del grafo ottenuto.

3.1.2 KG tramite LLM

Il secondo KG preso in considerazione è invece stato costruito sfruttando gli LLM poiché questi ultimi hanno saputo dimostrare di avere degli strumenti potenti per capire e generare testo. In particolare, gli LLM hanno mostrato delle capacità più avanzate per l'estrazione dei dati combinati. Per assicurare precisione, sono state usate delle tecniche in combinazione con gli LLM, quelle di *NLP* (Natural Language Processing), dove Natural Language Processing non è altro che un campo dell'informatica che si focalizza sul consentire ai computer di capire il linguaggio naturale. Solitamente alcuni task sono analizzare, comprendere e generare testo. [8]

L'approccio proposto per generare questo secondo Knowledge Graph, cercando di sfruttare le capacità avanzate degli LLM per la generazione di ontologie e l'estrazione dei dati e le tecniche NLP per garantire chiarezza e precisione, comprende l'utilizzo di due pipelines, una per gli LLM e una per le tecniche NLP. La pipeline per gli LLM è la seguente [11]:

1. **Preparazione del documento:** questa prima fase si focalizza sull'estrazione del testo da due fonti principali per assicurare l'efficacia di entrambe. La prima sono file interi PDF da cui estrarre testo, la seconda è invece un insieme di specifiche sottoparti di sentenze selezionate da esperti del dominio. Poiché questa seconda fonte risulta variare in struttura e complessità rispetto alla prima, l'estrazione è stata effettuata manualmente. Il testo estratto è stato poi formattato in un documento per le fasi successive;
2. **Creazione di RAG:** in questa fase si sono concentrati sulla creazione di modelli *RAG*, Retrieval-Augmented Generation, ossia dei modelli che si basano non solo su conoscenza ottenuta in fase di addestramento, ma anche su conoscenza che vanno a ritrovare su un corpus esterno consultabile dinamicamente. [18] Sono stati introdotti per migliorare le prestazioni degli LLM che possono avere problemi nel rispondere a task specifici e soffrire di *allucinazioni*. Le allucinazioni sono delle informazioni che possono sembrare vere ma che in realtà sono false e inesistenti. [16];
3. **Creazione dell'ontologia base:** : a questo punto si sono iniziate a gettare le fondamenta per l'ontologia creando le classi e con l'aggiunta di Object-Property e DatatypeProperty. Per perfezionare l'ontologia, su documenti

specifici del dominio è stata applicata una tecnica di *zero-shot prompting*, che è una tecnica che consente agli LLM di eseguire compiti specifici senza fornire esempi espliciti nel prompt. Il modello di svolge un'attività basandosi esclusivamente sulle istruzioni fornite, sfruttando la conoscenza acquisita durante il pre-addestramento. [27]. Dall'utilizzo di questa tecnica con più iterazioni casuali, si sono riscontrate delle somiglianze strutturali tra i documenti, permettendo così di identificare schemi comuni. Dopo aver ottenuto l'ontologia si sono eliminati gli elementi superflui dello schema;

4. **Creazione del KG:** dall'ontologia creata nella fase precedente e dal modello RAG, si sono generati per ogni documento il rispettivo KG. I vari KGs ottenuti sono poi stati fusi in un unico KG preservando le entità distinte per garantire consistenza tra i vari KGs;
5. **Creazione e risposta delle competency questions:** come fase finale della pipeline per l'LLM sono state generate le competency questions tramite *Mixtral*, un LLM, che ha creato le domande dato l'input dell'ontologia creata. Sempre utilizzando *Mixtral* sono poi state create le risposte alle domande selezionate, il tutto poi controllato e validato manualmente.

Infine, la pipeline NLP per complementare l'uso degli LLM è stata la seguente [11]:

1. **Preprocessing:** fase utilizzata per valutare e comparare i nuovi approcci LLM, essa è stata applicata dopo l'estrazione del testo dai documenti. Questa fase ha coinvolto vari step come: la *rimozione della punteggiatura*, la *tokenizzazione* (suddivisione delle parole in token), *rimozione delle stopwords* (un insieme di parole prefissate che il sistema sa di non dover considerare come articoli, proposizioni,...). Il tutto per preparare il testo per le fasi successive;
2. **Part of Speech:** dopo la tokenizzazione, è stata applicata la tecnica di *POS tagging*, ossia una tecnica che categorizza i componenti di una frase. In questo caso, è stato utilizzato per estrarre le triple dal documento categorizzando i token in soggetto, predicato e oggetto in base al ruolo grammaticale;
3. **Creazione delle triple:** in quest'ultima fase le triple vengono composte associando al soggetto il rispettivo oggetto e predicato. Il processo risulta

infine in una rappresentazione strutturata del testo che cattura le relazioni tra le entità nel testo.

3.2 Metodologie di integrazione di KGs

Dati quindi i due Knowledge Graphs creati con le metodologie descritte nella sezione 3.1, l'obiettivo di questa tesi è quello di unificarli in un unico Knowledge Graph per avere una rappresentazione dell'informazione più robusta. Per fare ciò si sono presi in considerazione due approcci. Il primo approccio considera l'allineamento delle ontologie, mentre il secondo l'integrazione di Knowledge Graphs.

3.2.1 Primo approccio: Allineamento delle ontologie

In questo primo approccio l'integrazione dei KGs avviene tramite l'allineamento delle ontologie sottostanti ai Knowledge Graphs considerati. Consideriamo le ontologie perché forniscono la struttura semantica di base dei dati all'interno del KG, descrivendo le classi, le relazioni e le proprietà degli enti nel grafo. Allineandole si risolverebbero i problemi riguardanti i conflitti semantici, le diverse rappresentazioni e inoltre faciliteremmo la fusione dei dati.

L'idea è quella di sfruttare gli LLM per effettuare l'allineamento delle ontologie. Si è però visto che gli LLM da soli possono portare a risultati non soddisfacenti o persino a output non usabili [21], così si era pensato di sfruttare la modularità dell'ontologia per fare in modo che i Large Language Models possano focalizzarsi su moduli più piccoli e ben definiti, consentendo quindi un miglioramento delle performance, dove un modulo non è altro che l'insieme delle classi, proprietà e assiomi all'interno dell'ontologia rilevanti per una nozione chiave. [25]

La metodologia usata per ottenere ontologie modulari è *MOMo*, che guida la progettazione di ontologie attraverso un processo modulare e iterativo. Si basa sull'idea di suddividere l'ontologia in moduli distinti, ciascuno dei quali rappresenta un concetto o una relazione specifica. Questi moduli sono costruiti utilizzando *Ontology Design Patterns (ODP)*, che sono schemi riutilizzabili per rappresentare concetti comuni in modo coerente e standardizzato. [25] [26]

Il problema fondamentale di questo approccio risiede nel fatto che *MOMo* risulta essere un sistema di creazione di ontologie modulari, non funziona quindi su

ontologie già esistenti, quale risulta essere il nostro caso. Inoltre, il suo utilizzo non è completamente automatizzato e richiede il coinvolgimento di esperti. [26]

Si è persino considerata l'opzione di utilizzare metodologie più semplici e tradizionali di alignment delle ontologie, ma queste non riuscirebbero a carpire la complessità dei dati del nostro dominio specifico. [1]

Visto che l'allineamento delle ontologie non risultava essere possibile data la natura non modulare dei nostri KGs e delle ontologie sottostanti e, vista la non possibilità di utilizzare metodologie più semplici, si è optato per la diretta integrazione di Knowledge Graphs come vedremo nella sottosezione 3.2.2

3.2.2 Secondo approccio: Integrazione di KGs

In questo secondo approccio l'integrazione dei Knowledge Graphs avviene direttamente senza passare per l'allineamento delle ontologie come nel primo approccio.

Anche in questo caso si sfruttano gli LLMs per effettuare l'integrazione. La ricerca ha infatti evidenziato alcuni vantaggi di questi modelli come la conoscenza estesa, la comprensione del linguaggio e la generalizzabilità. Insieme ad essi sono emersi anche alcuni limiti come la mancanza di capacità di pianificazione, le allucinazioni e l'inferenza incerta, in particolare, perché può ostacolare la comprensione e la capacità di formulare previsioni stabili in risposta a domande complesse. [41]

I Knowledge Graphs, invece, presentano vantaggi nella memorizzazione della conoscenza accurata, nel supporto all'inferenza esplicita e nella facilità di aggiornamento e modifica. Nonostante ciò, presentano anche questi svantaggi come un grande sforzo umano nella creazione, conoscenza incompleta causata dai rigorosi requisiti di standardizzazione dei dati e alla quantità limitata di questi ultimi. [41] Come si può vedere dall'immagine in Figura 3.2, che descrive quanto detto sopra, possiamo notare come effettivamente LLMs e KGs abbiano vantaggi complementari. Proprio grazie a questa loro caratteristica sono stati fatti tanti studi che combinano questi due strumenti per sfruttarne i rispettivi vantaggi e sopperire così alle loro limitazioni. Tra i tanti campi in cui sono usati, quello di nostro interesse è sicuramente quello per l'integrazione dei Knowledge Graphs.

L'integrazione di Knowledge Graphs è tutt'ora un ambito di ricerca molto importante, dovuto a un aumento dei KGs nel tempo che però non ha portato a

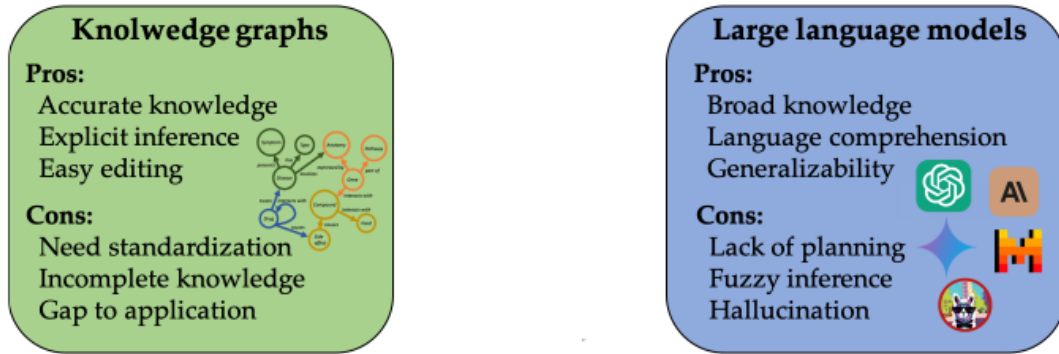


Figura 3.2: Vantaggi e svantaggi di LLMs e KGs a confronto

scoprire altrettante nuove tecniche di integrazione per avere una rappresentazione della conoscenza più completa e robusta.

Con il sempre maggiore aumento e diversità dei Knowledge Graphs si cercano tecniche di integrazione efficienti ed affidabili che vadano a superare i seguenti due problemi delle più classiche tecniche di integrazione, come quelle basate su string-matching [41]:

1. **Eterogeneità semantica:** KGs differenti spesso utilizzano terminologie, definizioni e contesti diversi per rappresentare concetti simili;
2. **Diversi livelli di granularità:** i KGs possono variare nel livello di dettaglio con cui descrivono entità e relazioni, influenzando la coerenza e l'utilizzabilità dei dati integrati.

Gli LLMs hanno risolto i problemi descritti sopra, apportando anche i seguenti vantaggi [41]:

1. **Capacità di comprensione:** gli LLMs hanno una forte capacità di comprensione del linguaggio naturale, permettendo così le relazioni presenti tra concetti che sfuggono ai metodi tradizionali;
2. **Vasta conoscenza pregressa:** possono attingere a vasta conoscenza pregressa acquisita durante il pre-training per supportare la disambiguazione delle entità e il mapping delle relazioni tra KGs differenti;
3. **Domini specializzati:** utili in domini specializzati grazie alla loro abilità di *few-shot learning* che si riferisce al problema di imparare il pattern sottostante

a dei dati solo da alcuni esempi, ossia l'abilità di generalizzare dati pochi esempi di training. [22]

Un esempio di efficacia dell'approccio LLM all'integrazione di Knowledge Graphs sono sistemi come **HiPrompt**, specializzato per l'integrazione di entità in ambito biomedico. Il sistema usa l'etichetta dell'entità del KG come query e cerca nel testo delle entità della tassonomia generando così una lista di top-k entità più simili, rappresentanti i candidati che l'LLM valuta usando prompt gerarchici, cioè includono informazioni sulla posizione dell'entità nella tassonomia, per selezionare il match migliore. [19]

Il problema di un sistema come HiPrompt è il suo stretto legame con il dominio biomedico per cui è stato progettato. Si è quindi cercato un sistema che potesse avere i vantaggi dell'utilizzo degli LLM e che potesse essere applicato al nostro dominio di interesse. Tra gli esempi di sistemi che sfruttano LLM utili al nostro scopo è stato trovato e usato **AutoAlign**.

3.3 AutoAlign

AutoAlign è un sistema di integrazione di Knowledge Graphs che si basa sull'allineamento delle entità. Quest'ultimo mira a identificare coppie di entità tra i due Knowledge Graphs che rappresentano lo stesso concetto. I metodi esistenti tradizionali fanno affidamento sull'*allineamento manuale dei seed*, che non sono altro che corrispondenze tra entità appartenenti ai due KGs distinti, ritrovate da esperti del dominio e utilizzate per l'addestramento o l'inizializzazione dei modelli di allineamento. [12]

Fare affidamento però su questi seed presenta dei notevoli svantaggi come [43]:

- **Tempo e competenze:** la loro creazione necessita di intervento umano esperto, il che è costoso e difficilmente scalabile per grandi dataset;
- **Scarsa riutilizzabilità:** i seed non sono generalizzabili; ogni nuovo compito di allineamento necessita di un nuovo set manuale;
- **Bassa affidabilità:** La qualità dei seed può variare significativamente a causa di bias introdotti dai diversi annotatori umani, ovvero esperti incaricati di

identificare manualmente le corrispondenze tra entità, e dalla possibilità di errore umano.

Per evitare i problemi sopra citati, AutoAlign risulta essere un metodo che automatizza l'allineamento di KGs senza la necessità di annotazioni manuali. Come si può

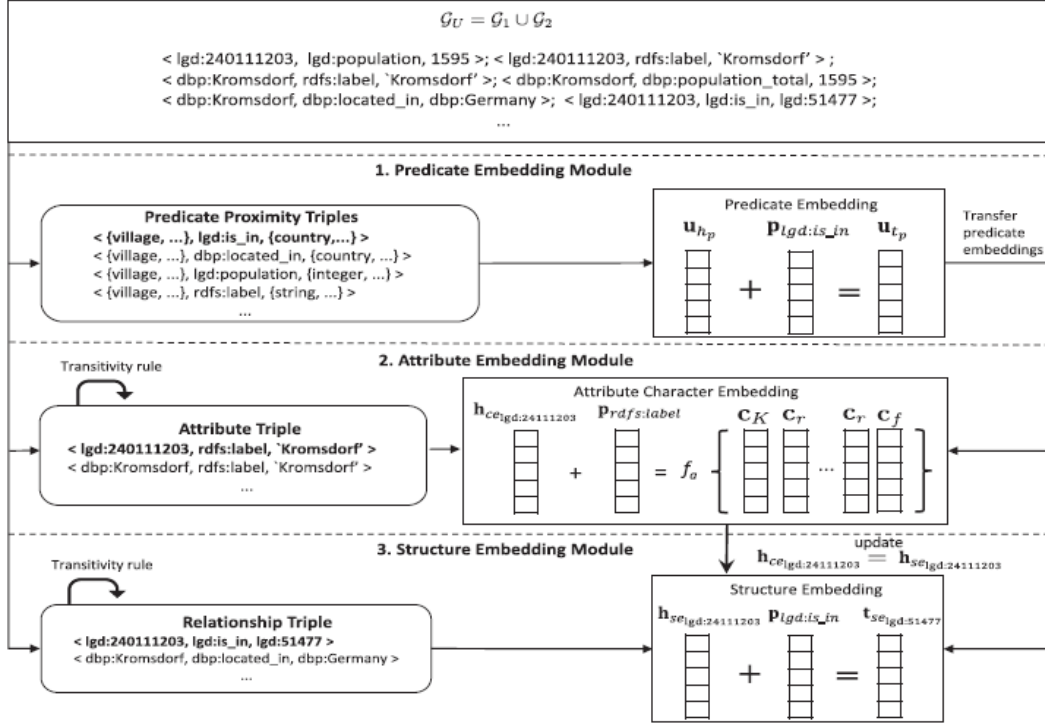


Figura 3.3: Metodo proposto da AutoAlign per l'allineamento delle entità [43]

vedere dalla Figura 3.3, il modello utilizza tre moduli di embedding principali, dove un *embedding* non è altro che una rappresentazione vettoriale continua appresa di elementi discreti, come parole o token, che permette agli algoritmi di lavorare su dati simbolici catturando le similarità semantiche tra gli elementi. [17]

Prima dell'applicazione dei moduli di embedding, per assicurarsi che sia gli embedding dei predicati sia quelli delle entità dei due KG coesistano nello stesso spazio vettoriale, si uniscono il primo Knowledge Graph G_1 e il secondo G_2 in un unico formando $G_U = G_1 \cup G_2$. Si hanno quindi tutte le triple dei due Knowledge Graphs nella loro forma originale, non ancora allineate. Le triple che si vanno a formare sono: le triple T_p di prossimità dei predicati(es: $\langle \text{village}, \text{dbp}:\text{located_in}, \text{country} \rangle$)

e $\langle \text{village, lgd:is_in, country} \rangle$), le triple di relazione Tr dove anche l'oggetto denota un'entità e le triple di attributi Ta se l'oggetto è un letterale. [43]

3.3.1 Modulo di embedding dei predicati

Il concetto base è che gli stessi predicati di due Knowledge Graphs diversi connettono gli stessi tipi di entità. Per rendere la procedura dell'allineamento dei predicati automatica si apprende l'embedding dei predicati da un grafo di prossimità dei predicati dei due KGs. Il *grafo di prossimità dei predicati* è un grafo in cui al posto delle entità sono presenti i loro rispettivi tipi. [43]

Le fasi di questo modulo sono [43]:

- **Costruzione del grafo di prossimità dei predicati:** un grafo di prossimità dei predicati non è altro che un grafo che rappresenta le relazioni tra i tipi di entità rispetto alle entità stesse, dove per tipi di entità ci si riferisce alle categorie generali di entità. Per costruire il grafo di prossimità si parte da Gu descritto come il grafo derivato dall'unione dei due Knowledge Graphs e contenente tutte le loro triple in forma originale. Prese le triple di Gu si sostituisce ogni soggetto e oggetto con il rispettivo tipo. Le sottofasi sono:
 - **Estrazione dei tipi di entità:** si estrae il valore del tipo dell'entità prendendo i valori associati al predicato *rdfs:type*;
 - **Allineamento del tipo tramite LLMs:** dopo aver ottenuto i tipi delle entità dei due Knowledge Graphs, li si allinea. Questo avviene perchè due KGs possono utilizzare diversi nomi per esprimere lo stesso significato. Si realizza così un prompt da dare in input all'LLM specificando il task che deve compiere e su quali set di dati deve lavorare. Si definiscono infatti due set ciascuno dei quali contiene i tipi delle entità che si riferiscono per il *Set1*, ai tipi delle entità del primo Knowledge Graph preso in considerazione e *Set2* per il secondo. Si sfruttano quindi gli LLMs per ottenere le coppie di sinonimi tra i due insiemi e si ha in output delle coppie di sinonimi nella forma $(type1, type2)$ sostituendo poi i sinonimi per avere tipi simili sotto la stessa forma.

- **Apprendimento del modulo:** per catturare al meglio la similarità dei predicati, ci si dovrebbe concentrare sui tipi di dati più distintivi. AutoAlign, in questo caso, propone due modalità per aggregare più tipi di entità:
 - **Funzione di somma pesata:** combina più tipi di entità in un'unica rappresentazione vettoriale, dando più importanza ai tipi più distintivi. Si crea un embedding che rappresenti efficacemente un'entità, pesando i suoi tipi secondo la loro specificità e rilevanza. Per ogni tipo associato a un'entità si calcola un peso che tiene conto di fattori come specificità del tipo in WordNet, frequenza, consistenza e numero di attributi. I pesi vengono poi normalizzati con una funzione softmax per trasformarli in una distribuzione di probabilità e si calcola infine la somma pesata degli embedding dei tipi;
 - **Funzione basata sull'attenzione:** la funzione in questo caso si focalizza sul dare più peso ai tipi più rilevanti e a ignorare quelli troppo generici. Ogni tipo collegato ad un'entità ha un peso di *attenzione*, che è un meccanismo che permette a un modello di concentrarsi selettivamente su parti rilevanti dell'input [5], che quindi misura quanto quel tipo è utile per capire il significato del predicato. Dopo aver calcolato il peso di ogni tipo si fa la somma pesata degli embedding dei tipi, usando i pesi di attenzione.

Le due funzioni di aggregazione dei tipi vengono impiegate per calcolare le pseudo-embedding dell'entità di testa e di coda nelle triple di prossimità. L'addestramento degli embedding dei predicati avviene mediante la minimizzazione di una funzione obiettivo di ranking, che riduce la distanza tra la somma dell'embedding dell'entità di testa e del predicato e l'embedding dell'entità di coda per le triple corrette, e aumenta la distanza nel caso di triple corrette, generate sostituendo casualmente il predicato o una delle entità.

3.3.2 Modulo di embedding degli attributi

Questo modulo serve a rappresentare in uno spazio vettoriale continuo le triple di attributi di Gu . L'oggetto della tripla è quindi un valore letterale. Anche in questo caso però uno stesso valore può essere espresso in modi differenti. Per

risolvere questo problema AutoAlign usa una funzione composta per codificare il valore dell'attributo in vettore. [43]

Le funzioni di composizione proposte sono [43]:

- **SUM**: è la somma dei vettori dei caratteri dell'attributo valore. Il problema di questa funzione è che ignora l'ordine dei caratteri;
- **LSTM**: sfrutta una rete neurale Long Short-Term Memory, ossia capace di mantenere informazioni nel tempo, per codificare la sequenza di caratteri in un singolo vettore;
- **N-Gram**: alternativa proposta per catturare informazioni composizionali di una stringa.

Si addestra poi il modello per minimizzare la distanza tra triple vere e triple false dove le triple false si ottengono sostituendo casualmente il soggetto e il valore dell'attributo.

3.3.3 Modulo di embedding della struttura

TransE è un modello di embedding per rappresentare entità e relazioni in un Knowledge Graph. Data la tripla $\langle h, r, t \rangle$, TransE, e modelli simili, suggeriscono che l'embedding dell'oggetto dovrebbe essere vicino all'embedding del soggetto più l'embedding del predicato: $h + r \approx t$. Esso mira a preservare l'informazione strutturale dell'entità. [43] [9]

Il modulo di structure embedding di AutoAlign si basa su TransE, ma lo estende per attribuire pesi diversi ai vicini di un'entità in base all'affidabilità del predicato che li collega. L'intuizione è che le entità collegate tramite predicati già allineati forniscono informazioni più utili per l'allineamento rispetto a quelle collegate da predicati non allineati. [43]

I predicati vengono così suddivisi in tre categorie [43]:

1. **Predicati già allineati**: seguono convenzioni comuni e sono presenti in entrambi i KGs;
2. **Predicati implicitamente allineabili**: non esplicitamente uguali ma semanticamente simili;

3. **Predicati non allineati:** compaiono in un solo KG e introducono quindi rumore.

Per limitare l'influenza del rumore, AutoAlign amplia TransE attraverso un peso α per dare più importanza alle triple con predicati che appaiono spesso nel grafo G_u .

3.3.4 Allineamento dell'entità

Le informazioni derivate dagli embeddings dei predicati e degli attributi consentono al sistema di imparare a rappresentare entità di due KG diversi in uno spazio di embedding unificato. Il risultato è che entità simili nei due KG avranno vettori molto vicini nello spazio di embedding, facilitando il loro allineamento. La corrispondenza finale tra entità è calcolata attraverso una ricerca del massimo valore di similarità coseno, applicando una soglia di similarità β , scartando le coppie con somiglianza inferiore a questo valore. [43]

3.3.5 Risultati di AutoAlign

Per valutare il sistema sono stati usati seed dal 10% al 50% (per gli altri sistemi). La valutazione si basa su Hits@k ($k=1,10$), ovvero la percentuale di entità correttamente allineate che si trovano nelle prime k posizioni delle predizioni. Valori più alti indicano prestazioni migliori. [43]

Come si può vedere dalla Figura 3.4 AutoAlign, in entrambe le sue versioni, *AutoAlign-W* e *AutoAlign-A* la cui differenza risiede nel fatto che utilizzano rispettivamente per l'aggregazione la funzione di somma pesata e l'altra la funzione basata sull'attenzione, ha ottenuto risultati superiori agli altri sistemi. Si vede anche come AutoAlign-A ha ottenuto risultati ancora superiori a AutoAlign-W. [43]

AutoAlign ha quindi dimostrato robustezza e alta efficacia anche senza dati di allineamento preliminari e migliori prestazioni rispetto a metodi tradizionali, specialmente in scenari con pochi o nessun seed alignment. Per questi motivi, e anche grazie al fatto che AutoAlign è un sistema automatizzato e che non necessita di seed alignment manuali, si è optato per utilizzare AutoAlign per effettuare l'integrazione dei nostri KGs.

3.3 – AutoAlign

TABLE III
EFFECT OF THE AMOUNT OF SEED ENTITY ALIGNMENTS ON EA PERFORMANCE IN TERMS OF HITS@K (%)

Method		Seed: 0%		Seed: 10%		Seed: 20%		Seed: 30%		Seed: 40%		Seed: 50%	
		Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10
DW-NB													
Translation-based	MTransE	N/A	N/A	2.82	10.45	5.42	18.72	7.88	25.75	10.42	31.44	12.98	36.00
	IPTransE	N/A	N/A	5.98	13.45	7.54	18.78	12.90	24.61	16.32	32.86	23.54	35.98
	BootEA	N/A	N/A	8.12	16.15	12.54	20.13	17.92	28.38	21.46	35.16	25.44	37.57
	TransEdge	N/A	N/A	22.98	48.12	38.29	56.22	45.27	68.95	49.26	75.25	54.85	79.68
	JAPE	N/A	N/A	4.62	7.87	8.62	14.43	12.57	19.96	17.20	27.32	19.91	30.63
	MultiKE	N/A	N/A	80.25	87.58	82.56	88.92	84.06	90.05	84.87	91.24	85.21	95.06
	AttrE	87.98	95.80	87.98	95.80	87.98	95.80	87.98	95.80	87.98	95.80	87.98	95.80
	AutoAlign-W	87.81	95.86	87.81	95.86	87.81	95.86	87.81	95.86	87.81	95.86	87.81	95.86
AutoAlign-A	88.73	96.91	88.73	96.91	88.73	96.91	88.73	96.91	88.73	96.91	88.73	96.91	
GNN-based	MuGNN	N/A	N/A	13.49	37.79	20.96	49.28	26.92	56.77	31.09	61.43	34.41	64.96
	AliNet	N/A	N/A	14.58	31.46	18.55	35.84	24.34	50.46	28.39	55.46	35.31	58.22
	KECG	N/A	N/A	18.95	34.17	24.32	40.78	30.24	48.66	35.29	52.12	39.40	62.31
	GCN-Align	N/A	N/A	12.40	30.18	20.04	41.56	24.76	48.52	29.02	53.43	31.80	56.20
	HGCN	N/A	N/A	58.08	62.15	63.14	68.15	78.97	86.51	84.25	90.75	88.54	91.54
	GMNN	N/A	N/A	71.32	74.24	75.34	79.23	80.98	82.23	82.67	85.87	84.59	88.64
	RDGCN	N/A	N/A	59.22	62.98	64.22	68.98	79.02	87.12	85.34	90.45	88.21	93.23
	CEA	N/A	N/A	50.13	52.31	63.25	64.12	80.32	84.21	84.34	85.54	86.58	88.34
	MRAEA	N/A	N/A	53.75	54.74	64.58	66.12	81.54	85.97	83.54	86.02	84.06	87.55
	NMN	N/A	N/A	51.45	59.78	68.21	72.54	84.03	88.21	85.65	90.54	88.69	95.46
	AttrGNN	N/A	N/A	45.79	78.28	51.67	82.85	54.65	84.30	59.48	86.18	62.08	88.74
	UPLR	0.34	1.62	0.34	1.62	0.34	1.62	0.34	1.62	0.34	1.62	0.34	1.62
DY-NB													
Translation-based	MTransE	N/A	N/A	0.01	0.15	0.01	0.39	0.08	0.68	0.08	1.39	0.13	1.89
	IPTransE	N/A	N/A	1.54	9.87	5.67	25.76	14.55	36.45	15.77	45.81	17.33	52.18
	BootEA	N/A	N/A	2.15	14.19	8.47	38.15	15.77	48.32	17.22	57.15	19.24	58.14
	TransEdge	N/A	N/A	22.98	47.50	37.85	64.85	48.98	72.15	58.95	76.54	62.49	78.54
	JAPE	N/A	N/A	0.70	1.83	1.57	3.37	1.40	3.27	1.37	1.77	2.37	4.97
	MultiKE	N/A	N/A	81.87	88.05	82.11	89.26	84.97	90.84	87.22	92.05	89.25	93.58
	AttrE	90.44	94.23	90.44	94.23	90.44	94.23	90.44	94.23	90.44	94.23	90.44	94.23
	AutoAlign-W	90.42	94.35	90.42	94.35	90.42	94.35	90.42	94.35	90.42	94.35	90.42	94.35
AutoAlign-A	91.27	95.62	91.27	95.62	91.27	95.62	91.27	95.62	91.27	95.62	91.27	95.62	
GNN-based	MuGNN	N/A	N/A	19.16	51.41	27.40	62.69	31.60	68.56	34.73	71.24	37.15	74.07
	AliNet	N/A	N/A	13.54	28.53	14.25	31.69	25.39	58.31	28.98	56.12	34.59	64.12
	KECG	N/A	N/A	11.19	23.65	14.89	27.25	20.95	34.48	22.81	35.44	24.71	37.15
	GCN-Align	N/A	N/A	8.56	25.09	17.88	43.88	24.36	53.43	31.29	62.44	33.56	67.88
	HGCN	N/A	N/A	52.54	64.51	65.87	77.40	71.14	85.64	71.45	85.64	74.54	87.48
	GMNN	N/A	N/A	62.34	70.34	64.32	67.34	75.57	77.47	78.65	82.65	82.34	85.62
	RDGCN	N/A	N/A	53.13	65.30	67.28	78.21	74.54	85.22	77.45	87.43	78.67	89.45
	CEA	N/A	N/A	55.24	58.97	64.35	65.42	74.56	78.42	77.78	80.95	78.91	83.24
	MRAEA	N/A	N/A	52.46	53.20	60.33	64.54	73.71	78.52	74.25	78.66	76.22	80.15
	NMN	N/A	N/A	55.74	64.78	62.54	70.54	75.87	80.54	84.55	88.69	90.78	94.77
	AttrGNN	N/A	N/A	77.21	88.03	79.44	89.76	80.16	90.19	81.31	90.84	83.98	91.95
	UPLR	89.84	93.11	89.84	93.11	89.84	93.11	89.84	93.11	89.84	93.11	89.84	93.11

* Methods that use attribute triples are underlined. The rest tables and figures follow this convention.

* AttrE, AutoAlign-W and AutoAlign- \bar{A} do not use any seed alignments.

Figura 3.4: Risultati ottenuti da AutoAlign [43]

Capitolo 4

Approccio proposto

L'obiettivo di questa tesi è quello di trovare una soluzione per l'integrazione di Knowledge Graphs in ambito giuridico con particolare focus ai casi di violenza sulle donne.

Nella sezione 3.3 del Capitolo 3 sono state presentate e analizzate due metodologie per l'allineamento dei KGs. Infine, nella sottosezione 3.3.5, sono stati evidenziati in particolar modo i buoni risultati di AutoAlign, anticipandolo come sistema scelto per il raggiungimento del nostro obiettivo.

In particolare, ci si è concentrati sulla creazione del grafo di prossimità e l'allineamento delle entità sinonime, quindi sulla parte descritta nella sottosezione 3.3.1 del Capitolo 3. Questa parte è fondamentale per l'effettiva integrazione dei Knowledge Graphs tramite AutoAlign poiché è il cuore pulsante per l'allineamento dei predicati e delle entità.

Come primo passo è fondamentale il reperimento dei due Knowledge Graphs da integrare. In particolare, si reperiscono i file turtle dei due KGs che verranno utilizzati per l'estrazione delle triple. I Knowledge Graphs presi in considerazione sono già stati ampiamente descritti nella sottosezione 3.1.1 e nella sottosezione 3.1.2 del Capitolo 3.

Una volta reperiti i nostri KGs si estrarranno le triple. Per ogni tripla si cercherà nel grafo locale del KG preso in considerazione, tramite una query SPARQL, i tipi dei soggetti e degli oggetti, come descritto dalla metodologia AutoAlign, con opportune modifiche in base ai Knowledge Graphs presi in considerazione. Si andrà così a formare man mano il grafo di prossimità dei predicati per ogni Knowledge

Graph.

Si andranno poi ad esplorare varie tecniche per l'allineamento dei tipi delle entità sinonime per infine andare ad effettuare la sostituzione dei sinonimi trovati.

Una volta fatto ciò si otterrà il nostro grafo di prossimità unico, cioè contenente tutte le triple dei nostri due Knowledge Graphs nella forma *<tipo soggetto, predicato, tipo oggetto>*, con i sinonimi trovati sostituiti alle triple del primo Knowledge Graph.

Il capitolo la definizione delle fasi del progetto nella sezione 4.1. Nella sezione 4.2 si descrivono le varie fasi della pipeline applicate. In particolare nella sottosezione 4.2.1 si mostrano i due Knowledge Graphs reperiti, facenti riferimento alla prima fase del progetto. Si va così a definire la seconda fase nella sottosezione 4.2.2, ossia la creazione del prototipo del grafo di prossimità. Nella sottosezione 4.2.3 si crea il prompt da passare poi all'LLM. L'output della sottosezione 4.2.3 è poi utilizzato per la sottosezione 4.2.4 in cui si utilizza l'LLM per ritrovare i sinonimi tra i tipi delle entità. Infine l'ultima fase viene descritta nella sottosezione 4.2.5 in cui le entità sinonime vengono allineate formando il grafo di prossimità dei predicati finale.

4.1 Pipeline

Come descritto nella sottosezione 3.3.1 per ottenere il grafo di prossimità dei predicati ed effettuare l'allineamento dei tipi delle entità ci sono varie fasi con cui procedere [42]. Le fasi prese in considerazione sono le stesse di AutoAlign, visto che l'obiettivo è la creazione del grafo di prossimità. Ovviamente, nelle varie fasi sono stati applicati cambiamenti in base ai Knowledge Graphs presi in considerazione.

Le fasi prese in considerazione sono:

1. **Reperire i file Turtle dei Knowledge Graphs da integrare:** è la fase introduttiva e fondamentale per effettivamente poter considerare l'integrazione;
2. **Creazione del prototipo del grafo di prossimità:** in questa fase si interrogano i Knowledge Graphs reperiti per ottenere i tipi delle entità. Per prima cosa si estraggono le triple da interrogare, ad ogni tripla si ricerca il

tipo del soggetto e dell'oggetto. Una volta reperito lo si sostituisce e la tripla, nel formato *<tipo soggetto, predicato, tipo oggetto>*, viene aggiunta alla creazione del nostro prototipo di grafo di prossimità dei predicati;

3. **Creazione del prompt:** in questa fase si crea il prompt da dare in pasto all'LLM scelto. Il suo compito sarà quello di trovare le coppie di sinonimi tra i tipi di entità ritrovati. Il risultato verrà utilizzato per l'allineamento dei tipi delle entità;
4. **Uso LLM:** si sfrutta un LLM per ottenere le coppie di sinonimi tra i tipi delle entità;
5. **Allineamento dei tipi:** una volta trovate le coppie di sinonimi, le si sfruttano per sostituire i tipi delle entità di un Knowledge Graph con quelle dell'altro. Così facendo si avrà una maggiore uniformità dei tipi per garantire i futuri allineamenti.

Nella Figura 4.1 vengono mostrate le fasi descritte sopra.

4.2 Applicazione

Di seguito viene descritta l'applicazione dell'approccio per la creazione del grafo di prossimità dei predicati. Il progetto è stato scritto in Python e si può trovare su GitHub¹.

4.2.1 Reperimento KGs

Dai rispettivi progetti da cui i due Knowledge Graphs sono stati creati, sono stati reperiti i rispettivi file Turtle.

Si ha quindi come input del progetto il file Turtle del Knowledge Graph creato con metodologia bottom-up (identificato nel progetto come *KG1*), una cui parte è rappresentata in Figura 4.2, e il file Turtle del Knowledge Graph creato tramite LLM (identificato nel progetto come *KG2*), una cui parte, a sua volta, è rappresentata in Figura 4.3. Come si può vedere, i due Knowledge Graphs sono stati creati

¹<https://github.com/SaraPadovano/KGs-Integration>

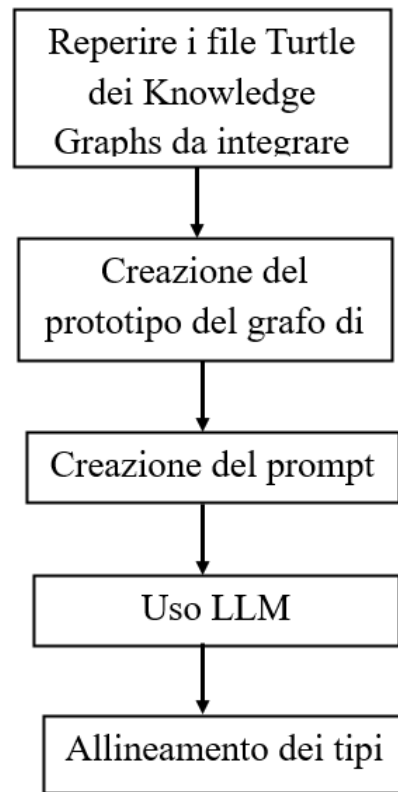


Figura 4.1: Fasi dell'approccio

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ns1: <https://github.com/PeppeRubini/EVA-KG/tree/main/ontology/ontology.owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<https://github.com/PeppeRubini/EVA-KG/tree/main/ontology/ontology.owl#10601/09> dcterms:abstract "Inadmissible"^^xsd:string ;
dcterms:contributor [ a <http://www.wikidata.org/entity/Q40348> ;
foaf:name "BOWLES P."^^xsd:string ] ;
dcterms:creator "Court (Fourth Section)"^^xsd:string ;
dcterms:date "2012-10-23"^^xsd:date ;
dcterms:identifier <https://hudoc.echr.coe.int/eng?i=001-114397> ;
dcterms:isVersionOf "ECLI:CE:ECHR:2012:1023DEC001060109"^^xsd:string ;
dcterms:type <http://www.wikidata.org/entity/Q327000> ;
ns1:hasApplicationNumber "10601/09"^^xsd:string ;
ns1:importanceLevel 3 ;
ns1:involveConventionArticle "10"^^xsd:string,
"13"^^xsd:string,
"35"^^xsd:string,
"6"^^xsd:string,
"8"^^xsd:string ;
ns1:respondentState <http://www.wikidata.org/entity/Q145> .
```

Figura 4.2: Una parte del file ttl del primo Knowledge Graph [24]

```

@prefix : <http://example.org/abuse-of-women#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

#Class

:Abuse a owl:Class ;
  rdfs:label "Abuse" ;
  rdfs:comment "The act of abusing someone, especially a woman." .

:PhysicalAbuse a owl:Class ;
  rdfs:subClassOf :Abuse ;
  rdfs:label "Physical Abuse" ;
  rdfs:comment "The use of physical force against someone, especially a woman." .

:EmotionalAbuse a owl:Class ;
  rdfs:subClassOf :Abuse ;
  rdfs:label "Emotional Abuse" ;
  rdfs:comment "The use of words or actions to hurt someone emotionally, especially a woman." .

:Perpetrator a owl:Class ;
  rdfs:label "Perpetrator" ;
  rdfs:comment "The person who commits the abuse." .

```

Figura 4.3: Una parte del file ttl del secondo Knowledge Graph [11]

per lo stesso ambito, ma rappresentano le informazioni in modo diverso. Il primo, infatti, ha informazioni dettagliate sui singoli casi, mentre il secondo ricava informazioni più generali con descrizioni a un livello più alto.

Si sono così identificati i due Knowledge Graphs su cui andare ad operare per ottenere il grafo di prossimità dei predicati.

4.2.2 Creazione del prototipo del grafo di prossimità

Una volta reperiti i file Turtle dei nostri Knowledge Graphs da integrare, lo step da compiere è quello della creazione del prototipo del grafo di prossimità.

Come primo passo si è considerato un KG per volta. Partendo dal Knowledge Graph creato con metodologia bottom-up, si carica il rispettivo grafo RDF dal file Turtle specificandone il formato e non inserendo invece il formato N-Triples per cogliere strutture RDF più articolate. Si vanno poi a creare i due set *typeset1*

e *typeset2* che andranno a contenere rispettivamente i tipi dei soggetti e i tipi degli oggetti. Si va a inizializzare anche la lista che conterrà le triple del grafo di prossimità e un set per cogliere i predicati del Knowledge Graph.

Una volta fatto ciò si va a iterare sulle triple del grafo. Se il soggetto o l'oggetto risulta essere un blank node, si dichiara un tipo specifico *BlankNode* ideato per dichiarare nel grafo di prossimità la presenza del blank node. Questo tipo è stato ideato per consentire l'inserimento delle triple anche con blank node all'interno del grafo. Così facendo si cerca di non avere perdita di informazione e di consentire di trovare una soluzione alla gestione dei blank node che in AutoAlign non vengono menzionati.

Dopo aver controllato se l'entità è un blank node o meno, se non risulta un blank node, si va a fare un check per definire se l'entità è un URI o un letterale. Se è un letterale si inserisce semplicemente il tipo. Nel caso sia un URI per il Knowledge Graph con metodologia bottom-up abbiamo tre possibilità:

1. Un URI nel formato `https://github.com/PeppeRubini/EVA-KG/tree/main/ontology/ontology.owl#`: in questo caso si crea una query SPARQL per recuperare il tipo dell'entità all'interno del grafo RDF. Se la query non dà risultato si effettua una ricerca all'interno del file XML del progetto dove è stato creato il Knowledge Graph con metodologia bottom-up. Questo perchè questa tipologia di URI non ha *rdf:type* esplicito, di conseguenza per effettivamente ritrovare il tipo bisogna ricondursi al file XML dove viene dichiarato il tipo. Dal risultato ritrovato si estrae quindi il tipo;
2. Un URI nel formato `wikidata.org/entity/`: si crea una query per l'interrogazione dell'endpoint di wikidata attraverso SPARQLWrapper. La query ritrova *instance of*, *subclass of*, *part of* dell'entità. La ricerca avviene per tutte e tre le proprietà perchè ci sono entità wikidata in cui l'una o l'altra non sono definite. Dal risultato ottenuto si estraggono poi i risultati;
3. Un URI nel formato `hudoc.echr.coe.int`: in questo caso si inizializza un driver, in particolare Chrome, per effettuare l'estrazione del tipo della decisione dal sito hudoc tramite Chrome automatizzato. Dalla pagina si estrae non il titolo ma il tipo del documento (i tipi in particolare sono due decision,

judgement). Questa scelta è stata fatta sia perchè il tipo del documento effettivamente identifica il tipo della risorsa, sia perchè i titoli dei singoli casi delle pagine hudoc non rappresentano informazione necessaria per i tipi delle entità che comunque, dopo il ritrovamento dei sinonimi, verrebbero sostituite da altri tipi. Così facendo quindi ritroviamo effettivamente il vero tipo della risorsa e semplifichiamo il processo di ritrovamento dei sinonimi.

Una volta effettuato il ritrovamento dei tipi delle entità, si inseriscono i tipi dei soggetti e degli oggetti nei rispettivi set, si inserisce la nuova tripla nella forma $\langle \text{tipo_soggetto}, \text{predicato}, \text{tipo_oggetto} \rangle$ nel grafo di prossimità dei predicati e si inseriscono i predicati nel set.

Per quanto riguarda il ritrovamento dei tipi per il Knowledge Graph creato tramite LLM, il processo generale è lo stesso ma cambia l'URI da prendere in considerazione. In questo caso abbiamo un unico URI, `http://example.org/abuse-of-women#`. Si va quindi a effettuare una query SPARQL sul grafo RDF per avere restituito il tipo dell'entità. Se l'entità risulta essere di tipo *Class* o *ObjectProperty*, si reperisce la categoria dall'URI stessa. Se invece è di tipo *DatatypeProperty*, si effettua un'altra query SPARQL sul grafo per reperirne il range, ossia il letterale.

Una volta effettuati questi passaggi, si uniscono tutti i tipi del primo Knowledge Graph considerato in un unico file e così anche per i tipi del secondo Knowledge Graph. Alla fine di questa fase si avranno quindi come file di output per le fasi successive: i grafi di prossimità dei predicati di entrambi i KGs, due set per KG contenenti i tipi dei soggetti e degli oggetti, un set per i predicati e due file contenenti uno tutti i tipi del primo KG e l'altro tutti i tipi del secondo.

4.2.3 Creazione del prompt

Per poter proseguire nelle fasi successive, fondamentale è la creazione del prompt da passare poi all'LLM. Una giusta stesura del prompt consente di ottenere dall'LLM i risultati sperati nel modo sperato.

Per fare ciò si è partiti come base dal prompt ideato da AutoAlign stesso che si può vedere in Figura 4.4. A partire quindi dal prompt di AutoAlign si è creato il prompt in Figura 4.5 dove dopo Set 1 e Set 2 si inseriscono i tipi rispettivamente del Knowledge Graph con metodologia bottom-up e poi quello creato tramite LLM. Si

“Now you are an expert in linguistics and knowledge graphs. I will give you two sets of words, indicating the entity types from two knowledge graphs. You need to identify all the word pairs from the two sets that are synonyms. For example, if the first set has the word ‘people’ and the second set has the word ‘person’, you need to identify the two words being synonyms and return me the pair (people, person). Now the following are the two sets: Set 1: {people, music,...} Set 2: {person, thing,...} Please return all the pairs that are synonyms from the two sets regarding entity types. Do not output the pairs if they are exactly the same. Remember you only need to return the pairs, each pair in one line. Each pair contain two types, one from Set 1 and another from Set 2, in the format (type1, type2).”

Figura 4.4: Prompt di AutoAlign per il LLM [9]

Now you are an expert in linguistics and knowledge graphs. I will give you two sets of words, indicating the entity types from two knowledge graphs. You need to identify all the word pairs from the two sets that are synonyms. For example, if the first set has the word ‘people’ and the second set has the word ‘person’, you need to identify the two words being synonyms and return me the pair (people, person) or for example, if you have StrasbourgCaseLaw in the first set and LegalCase in the second (StrasbourgCaseLaw, LegalCase). Pay attention to details because for example decision and LegalJudgment aren't synonyms and don't consider literal types. Now the following are the two sets: Set 1: {list_to_str(typeset1)} Set 2: {list_to_str(typeset2)} Please return all the pairs that are synonyms from the two sets regarding entity types. Do not output the pairs if they are exactly the same. Remember you only need to return the pairs, each pair in one line. Each pair contain two types, one from Set 1 and another from Set 2, in the format (type1, type2). Please pay very attention that they have to be synonyms or have a strong semantic connection. For example (jurist, perpatator) are not synonyms. Remember the synonyms have to be from the two different sets not from the same, the pair must be (type1, type2) not (type1, typw1) neither (type2,type2). Only (type1, type2).

Figura 4.5: Prompt del progetto

può notare dai due prompt che la base è la stessa, ma nel prompt in Figura 4.5 per ottenere i risultati voluti è stato dovuto ripetere in modo chiaro, esplicito e, a tratti,

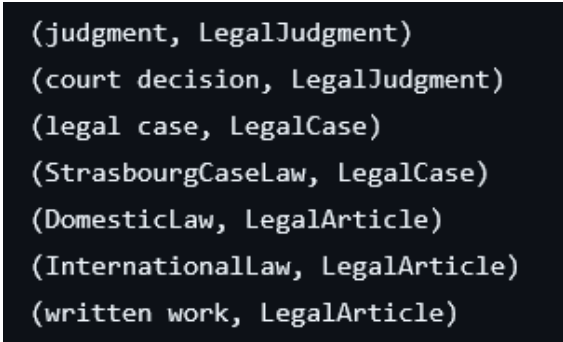
ripetuto, l'output desiderato, oltre ad aver dovuto inserire più esempi rispetto al prompt in Figura 4.4, probabilmente dovuto al dominio specifico da cui si stanno reperendo i sinonimi.

L'output di questa fase è quindi un prompt pronto per essere utilizzato con un LLM per il reperimento dei sinonimi.

4.2.4 Uso LLM

Una volta ottenuto il prompt, questa fase richiede l'utilizzo di LLM per trovare sinonimi tra i tipi delle entità. Il ritrovamento dei sinonimi dei tipi delle entità è fondamentale poiché gli stessi predicati di due Knowledge Graphs diversi connettono gli stessi tipi di entità, che è il concetto alla base di AutoAlign.

Per effettuare questo task è stato scelto come LLM, GPT senza l'utilizzo di API a causa dei costi del servizio. Si è quindi semplicemente dato a GPT il prompt creato, reperendo poi l'output che si può vedere in Figura 4.6. Il risultato ottenuto



```
(judgment, LegalJudgment)
(court decision, LegalJudgment)
(legal case, LegalCase)
(StrasbourgCaseLaw, LegalCase)
(DomesticLaw, LegalArticle)
(InternationalLaw, LegalArticle)
(written work, LegalArticle)
```

Figura 4.6: Risultato del ritrovamento dei sinonimi nel formato (type1, type2) dato da GPT

si trova nel formato corretto e specificato nel prompt e anche i sinonimi risultano essere coerenti e giusti rispetto all'ambito e ai tipi delle entità dati all'LLM.

Oltre all'utilizzo di GPT, sono stati provati altri modi per ritrovare i sinonimi tra i tipi delle entità. In particolare sono stati fatti tre tentativi:

1. *google/flan-t5-small*²: modello di LLM compatto e leggero che è stato richiamato localmente. Meno potente di GPT, si è voluto provare per vedere i risultati che si sarebbero potuti ottenere. A causa del numero di token maggiore

²<https://huggingface.co/google/flan-t5-small>

a 512 (numero massimo di token del modello), si è effettuata una truncation, ossia un troncamento delle parole presenti nel prompt. Si carica quindi, tramite la libreria *transformers*³, *AutoTokenizer* che converte il prompt da testo in token numerici comprensibili dal modello, pre-addestrato in base al nostro modello. Poi si carica il modello pre-addestrato di tipo sequence-to-sequence perchè si vuole generare testo (i sinonimi) a partire da un prompt. Si crea poi una pipeline per la generazione text-to text per semplificare alcuni passaggi⁴ e per sfruttarla nella generazione della risposta. Il risultato ottenuto da questo modello si può trovare in Figura 4.7;



Consequence, Victim

Figura 4.7: Risultato di flan-t5-small

2. *google/flan-t5-large*⁵: fa parte della stessa famiglia di flan-t5-small solo riesce a gestire un maggior numero di token. Visti i risultati deludenti ottenuti con il modello small, grazie al maggior numero di token disponibili e a una maggiore potenza. Nonostante questo, come si può vedere nella Figura 4.8, i risultati non sono stati anche in questo caso quelli sperati;



hasConsequence, hasLegalJudgment, hasConsequence, hasLegalJudgment, hasContext

Figura 4.8: Risultato di flan-t5-large

3. *all-MiniLM-L6-v2*⁶: è un modello popolare di SentenceTransformer, noto per essere molto efficiente, leggero e veloce mantenendo buoni risultati con la similarità semantica. *SentenceTransformer* è una libreria di Python per l'uso di modelli di embedding pre-addestrati specializzati nel convertire parole/frasi in embeddings in uno spazio vettoriale dove frasi semanticamente simili sono posizionate vicine⁷. Quindi il terzo e ultimo modello visto è un embedding

³<https://huggingface.co/docs/transformers/index>

⁴https://huggingface.co/docs/transformers/main_classes/pipelines

⁵<https://huggingface.co/google/flan-t5-large>

⁶<https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

⁷<https://www.sbert.net/index.html>

che prende le parole dei file contenenti i tipi dei due KGs, carica il modello all-MiniLM-L6-v2 e si seleziona un threshold pari a 0,54 (si è visto che con threshold più grandi si tagliano molte sinonimie importanti mentre, con threshold più piccolo, si inseriscono troppe sinonimie non accurate). Si calcolano poi gli embedding per ciascuna parola nei sue set e si calcola la similarità del coseno. Come si può notare dalla Figura 4.9 si sono ottenuti risultati migliori rispetto ai due modelli utilizzati sopra ma comunque non buono come quello ottenuto con GPT.

```
(StrasbourgCaseLaw, LegalCase, 0.6087594628334045)
(jurist, LegalJudgment, 0.6194978952407837)
(jurist, LegalCase, 0.5663079023361206)
(legal case, hasLegalCase, 0.6000578999519348)
(legal case, LegalJudgment, 0.6185129284858704)
(legal case, LegalArticle, 0.5991063714027405)
(legal case, LegalCase, 0.8462175130844116)
(court decision, LegalCase, 0.5740938186645508)
(legal profession, LegalJudgment, 0.6281073689460754)
(legal profession, LegalArticle, 0.6417595744132996)
(legal profession, LegalCase, 0.6414142847061157)
```

Figura 4.9: Risultato di all-MiniLM-L6-v2

Per i motivi descritti sopra e per i risultati visti, per l'ultima fase si è scelto di usare i sinonimi trovati da GPT.

4.2.5 Allineamento dei tipi

Nell'ultima fase l'obiettivo è l'allineamento dei tipi delle entità sinonime. Visto che la terminologia usata dal Knowledge Graph creato tramite LLM risulta essere più generale e meno legata alle singole istanze del dominio preso in considerazione, si è scelto di sostituire i tipi del Knowledge Graph creato con metodologia bottom-up con i rispettivi sinonimi trovati.

Si è quindi preso il dizionario dei sinonimi e si sono sostituiti i tipi del primo Knowledge Graph. Dopo aver fatto la sostituzione i due grafi di prossimità dei

predicati che si erano creati sono stati uniti in un unico grafo di prossimità finale necessario per le future fasi successive.

Capitolo 5

Valutazione

Questa tesi apre le porte verso l'integrazione di Knowledge Graphs in ambito giuridico con focus sui casi di violenza sulle donne. Rappresenta un primo gradino verso l'effettiva integrazione dei KGs presi in considerazione o può portare alla considerazione di nuove metodologie e sistemi che consentano la risoluzione del problema.

Il risultato di questa prima fase è stata la creazione di un grafo di prossimità dei predicati utilizzabile nelle fasi successive dell'integrazione con AutoAlign.

Per prima cosa si è presa visione quali tipi hanno subito l'allineamento e in che percentuale per capire a livello dei tipi che impatto quantitativo c'è effettivamente stato. Come evidenziato dalla Figura 5.1, i tipi totali ritrovati nei due KGs sono in totale 100, di questi solo il 7% ha subito l'allineamento. Nonostante sembri una piccola quantità, si può notare come ad essere state modificate sono stati i tipi di entità più importanti poiché strettamente collegati al mondo giuridico.

```
=== Risultato Analisi Riduzione Tipi ===  
Numero di tipi unici prima dell'allineamento: 100  
Numero di tipi unici dopo l'allineamento: 93  
Riduzione assoluta del numero di tipi: 7  
Riduzione percentuale: 7.00%  
Tipi unificati (sostituiti): ['Domesticlaw', 'Internationallaw', 'StrasbourgCaselaw', 'court decision', 'judgment', 'legal case', 'written work']  
=====
```

Figura 5.1: Risultato analisi riduzione dei tipi

Per dare una valutazione più visuale sul grafo di prossimità creato e sull'analisi in Figura 5.1, si utilizzano gli embeddings. Si ottiene così una valutazione

visiva e quantitativa di come l'allineamento dei tipi possa aver modificato la prossimità dei predicati. Il risultato sperato è che l'allineamento dei tipi abbia portato a un ricollocamento più logico e significativo, dimostrando l'effettiva utilità dell'allineamento.

Per effettuare la valutazione si considerano i grafi di prossimità prima e dopo l'allineamento dei tipi. Da entrambi i grafi si estraggono le triple e si creano delle frasi contestuali che servono all'embedding per catturare meglio il contesto semantico. Carica un modello pre-addestrato dalla libreria SentenceTransformers. In particolare si è utilizzato all-MiniLM-L6-v2 che è un modello piccolo e veloce, ma performante, già usato per scovare i sinonimi. Sono quindi stati generati gli embeddings relativi alle frasi contestuali estratte dalle triple, sia per quelle del grafo prima l'allineamento che per quelle del grafo dopo.

Dopodichè gli embeddings sono stati mediati per ciascun predicato, al fine di ottenere una rappresentazione compatta utile alla visualizzazione.

Si va così poi infine a realizzare il grafico che mette a confronto i predicati prima e dopo l'allineamento dei tipi. Il grafico è rappresentato in Figura 5.2 Se analiz-

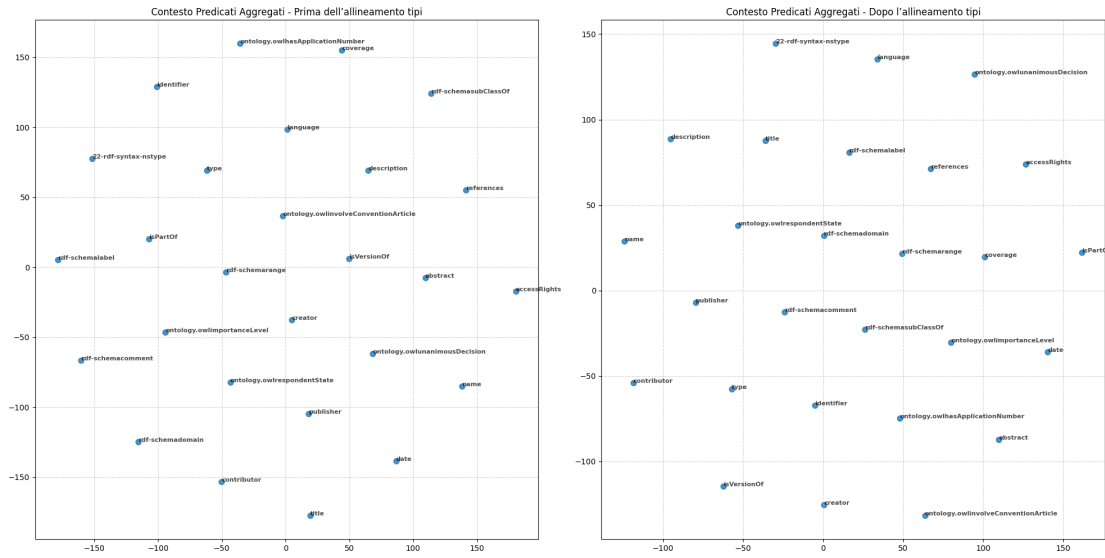


Figura 5.2: Grafico che confronta i predicati prima e dopo l'allineamento dei tipi

ziamo la distribuzione dei predicati prima dell'allineamento, a sinistra della figura Figura 5.2, possiamo notare come essi sembrano dispersi in gruppi, alcuni più vicini, ma senza una vera e propria correlazione. Se invece, osserviamo la distribuzione a destra possiamo già tranquillamente notare come tra i predicati si siano creati

dei cluster più compatti. E non solo, si nota anche una maggiore coerenza tra i predicati affini come il caso di *coverage*, *hasApplicationNumber*, *abstract*, *identifier*, che risultano essere più vicini nel plot.

Possiamo quindi dire che, certamente l'allineamento dei tipi ha portato senza dubbio una maggiore coerenza semantica tra i tipi delle entità, soprattutto a quelle giuridiche. Anche il grafico in Figura 5.2 ha mostrato una buona redistribuzione dei predicati dopo l'allineamento, mostrando una maggiore compatezza.

Capitolo 6

Conclusioni e sviluppi futuri

Questa tesi nasce nell'ambito della giustizia predittiva, concentrandosi sull'integrazione dei Knowledge Graphs creati dalle sentenze della Corte europea dei diritti dell'uomo, con focus sui casi di violenza sulle donne. Essa mira ad essere un punto di partenza per l'integrazione dei Knowledge Graphs in ambito giuridico, gettando le basi per quello che diventerà un'effettiva integrazione di KGs o uno spunto per testare nuovi sistemi e metodologie che possano consentire il raggiungimento dello scopo.

L'ambito dell'integrazione di Knowledge Graphs risulta tuttora essere un ramo di ampia ricerca per cercare di trovare metodologie e sistemi che garantiscano buone prestazioni, automatizzando aspetti costosi e laboriosi dei tradizionali sistemi di integrazione. Da qui è partita l'idea dell'utilizzo di AutoAlign come sistema automatizzato ed efficace che potesse dare una soluzione ad un problema tuttora vivo nell'ambito dei Knowledge Graphs.

Si è partiti quindi da due Knowledge Graphs, costruiti dalle stesse sentenze della Corte europea, e si è arrivati alla costruzione del grafo di prossimità dei predicati, fondamentale per i passi successivi all'integrazione con AutoAlign. Si è quindi applicato una parte importante di un sistema altamente effettivo nell'integrazione dei KGs. Si è così gettato le basi per uno sviluppo futuro. Infatti questa tesi rappresenta solo il primo gradino di quello che è l'applicazione di AutoAlign nell'ambito giuridico e, in generale, di quello che è l'ambito dell'integrazione di KGs.

Come già accennato, essendo questa tesi solo un punto di partenza, vi possono essere diversi sviluppi futuri:

1. **Applicazione endpoint per reperire informazioni dal KG creato con metodologia bottom-up:** si potrebbe pensare di sostituire la query SPARQL al grafo locale con una query all'endpoint SPARQL cercando di reperire gli effettivi tipi delle varie entità;
2. **Gestione blank node:** si può pensare a integrare una metodologia che gestisca i blank node evitando di perdere informazione. Si è già inserito una categoria per determinare la presenza di un blank node, ma si può pensare a come sfruttarla per effettivamente proseguire con l'integrazione;
3. **API a LLM potenti:** si può pensare di ampliare il ritrovamento dei sinonimi tramite LLM pensando all'inserimento di richieste API a LLM potenti che possano effettivamente soddisfare a pieno la richiesta del prompt. Si sono provati LLM locali, embedding ma nessuno ha effettivamente avuto un risultato effettivo come GPT nella ricerca dei sinonimi. L'unico problema è la limitatezza dei token e quindi il servizio a pagamento che ne deriva per l'utilizzo delle API;
4. **Ampliare gli step successivi di AutoAlign:** si può pensare di ampliare questa tesi applicando gli step successivi di AutoAlign sfruttando il grafo di prossimità dei predicati qua creato;
5. **Approfondire e testare metodologie accennate:** si potrebbe pensare di ampliare metodologie accennate e analizzate in questa tesi per trovare metodi alternativi (per esempio approfondire l'allineamento delle ontologie per trovare metodi alternativi a quelli proposti), sfruttare sistemi già esistenti e la loro logica per traslarla al problema da affrontare (come per il sistema HiPrompt).

Bibliografia

- [1] Reihaneh Amini, Sanaz Saki Norouzi, Pascal Hitzler, and Reza Amini. *Towards Complex Ontology Alignment using Large Language Models*. KGSWC 2024, 2024.
- [2] Grigoris Antoniou and Frank Harmelen. *Web Ontology Language: OWL*. Handbook on Ontologies, 2003.
- [3] Paolo Atzeni, Stefano Ceri, Piero Fraternali, Stefano Paraboschi, and Riccardo Torlone. *Basi di Dati*. McGraw-Hill, Italia, 2018.
- [4] Michael K. Bergman. *Advantages and myths of RDF*. Structured Dynamics LLC, 2009.
- [5] Dave Bergmann and Cole Stryker. Che cos'è un meccanismo di attenzione?, 2024. <https://www.ibm.com/it-it/think/topics/attention-mechanism>.
- [6] Tim Berners-Lee. Linked data - design issues, 2009. <https://www.w3.org/DesignIssues/LinkedData>.
- [7] Tim Berners-Lee, James Hendler, and Ora Lassila. *The Semantic Web*. Scientific American, 2001.
- [8] Taweh Beysolow II. *What Is Natural Language Processing?* Apress, Berkeley, CA, 2018.
- [9] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. *Translating Embeddings for Modeling Multi-relational Data*. Curran Associates, Inc., 2013.
- [10] Claudia d'Amato, Giuseppe Rubini, Francesco Didio, Francioso Donato, Fatima Zahra Amara, and Nicola Fanizzi. Prejust4women knowledge graph. https://lod-cloud.net/dataset/PREJUST4WOMAN_PROJECT.
- [11] Francesco Didio and Donato Francioso. Prejust4womans. <https://github.com/Fra3005/PreJust4Womans>.

- [12] Nikolaos Fanourakis, Vasilis Efthymiou, Dimitris Kotzinos, and Vassilis Christophides. *Knowledge graph embedding methods for entity alignment: experimental review*. Data Mining and Knowledge Discovery, 2023.
- [13] T. Matthias Frank and Stephan Zander. *The Linked Data Wiki: Leveraging Organizational Knowledge Bases with Linked Open Data*. Communications in Computer and Information Science, 2019.
- [14] Aidan Hogan. *Linked data & the semantic web standards*. Chapman and Hall/CRC, 2014.
- [15] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. *Knowledge Graphs*. Springer, 2021.
- [16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. *Survey of Hallucination in Natural Language Generation*. CoRR, 2022.
- [17] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 2025.
- [18] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, and K
- [19] Jiaying Lu, Jiaming Shen, Bo Xiong, Wenjing Ma, Steffen Staab, and Carl Yang. *HiPrompt: Few-Shot Biomedical Knowledge Fusion via Hierarchy-Oriented Prompting*. ACM, 2023.
- [20] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. *Large Language Models: A Survey*. arXiv, 2025.
- [21] S. S. Norouzi, M. S. Mahdavinejad, and P. Hitzler. *Conversational ontology alignment with ChatGPT*. ISWC 2023, 2023.
- [22] Archit Parnami and Minwoo Lee. *Learning from Few Examples: A Summary of Approaches to Few-Shot Learning*. 2022.
- [23] David L. Poole and Alan K. Mackworth. *Artificial Intelligence. Foundations of Computational Agents*. Cambridge University Press, Cambridge, 2023.

- [24] Giuseppe Rubini. Eva-kg. <https://github.com/PeppeRubini/EVA-KG>.
- [25] Cogan Shimizu and Pascal Hitzler. *Accelerating knowledge graph and ontology engineering with large language models*. Journal of Web Semantics, 2025.
- [26] Cogan Shimizu, Pascal Hitzler, and Karl Hammar. *Modular Ontology Modeling*. Semantic Web, 2023.
- [27] Meredith Syed and Vrunda Gadesha. What is zero-shot prompting?, 2025. <https://www.ibm.com/think/topics/zero-shot-prompting>.
- [28] Gytė Tamašauskaitė and Paul Groth. *Defining a Knowledge Graph Development Process Through a Systematic Review*. ACM Trans. Softw. Eng. Methodol. 32, 2023.
- [29] W3C. Owl web ontology language overview, 2004. <https://www.w3.org/TR/owl-features/>.
- [30] W3C. Owl web ontology language reference, 2004. <https://www.w3.org/TR/owl-ref/>.
- [31] W3C. Rdf primer, 2004. <https://www.w3.org/TR/rdf-primer/>.
- [32] W3C. Sparql query language for rdf, 2008. <https://www.w3.org/TR/rdf-sparql-query/>.
- [33] W3C. Owl, 2012. <https://www.w3.org/OWL/>.
- [34] W3C. Sparql 1.1 entailment regimes, 2013. <https://www.w3.org/TR/sparql11-entailment/>.
- [35] W3C. Rdf, 2014. <https://www.w3.org/RDF/>.
- [36] W3C. Linkeddata, 2023. <https://www.w3.org/wiki/LinkedData>.
- [37] W3C. Rdf 1.2 concepts and abstracts syntax, 2025. <https://www.w3.org/TR/rdf12-concepts>.
- [38] W3C. Rdf 1.2 schema, 2025. <https://www.w3.org/TR/rdf12-schema/>.
- [39] W3C. Rdf 1.2 turtle, 2025. <https://www.w3.org/TR/rdf12-turtle>.
- [40] Dawid Wiśniewski, Jędrzej Potoniec, Agnieszka Ławrynowicz, and C. Maria Keet. *Analysis of Ontology Competency Questions and their formalizations in SPARQL-OWL*. 2018.
- [41] Carl Yang, Ran Xu, Linhao Luo, and Shirui Pan. *Knowledge Graph and Large Language Model Co-learning via Structure-oriented Retrieval Augmented Generation*. IEEE Data Eng. Bull., 2024.

- [42] Rui Zhang, Yixin Su, Bayu Distiawan Trisedya, Xiaoyan Zhao, Min Yang, Hong Cheng, and Jianzhong Qi. Autoalign. <https://github.com/ruizhang-ai/AutoAlign>.
- [43] Rui Zhang, Yixin Su, Bayu Distiawan Trisedya, Xiaoyan Zhao, Min Yang, Hong Cheng, and Jianzhong Qi. *AutoAlign: Fully Automatic and Effective Knowledge Graph Alignment enabled by Large Language Models*. 2023.

Ringraziamenti

Ringrazio innanzitutto la mia relatrice e i miei correlatori per la costante presenza e l'aiuto fornitomi durante la stesura della tesi. È stata un'opportunità di crescita e di approfondimento molto preziosa.

Ringrazio poi la mia famiglia, in particolare, mia madre Maria e mio padre Giuseppe per il loro continuo sostegno durante il percorso di studi e non solo. Li ringrazio per avermi cresciuto e sostenuto nelle mie scelte.

Ringrazio i miei fratelli Paolino e Giacomino per tutti i bei momenti passati insieme, per la loro costante presenza e affetto che ha contraddistinto la mia infanzia e adolescenza. Li ringrazio per tutti i consigli richiesti e non, per il loro continuo incoraggiamento. Nonostante la distanza, ogni volta che ci sentiamo e vediamo è come se non fosse passato un solo giorno lontani, ricordandomi di quando da piccola piangevo ogni volta che uno di voi andava in gita per qualche giorno. Vi ringrazio perché so che, nonostante il tempo passi e la distanza possa aumentare o diminuire, io posso sempre contare su di voi e che l'affetto che ci lega non si piega a ragioni di tempo o spazio. Spero che possiate sempre trovare la felicità dovunque siate e avere a fianco a voi persone che vi vogliano genuinamente bene e che vi sappiano valorizzare per le bellissime persone che siete.

Ringrazio mio zio Giuseppe, mia zia Gianna e mio cugino Andrea, pilastri importanti della mia vita che sono sempre stati presenti pronti a sostenere e consigliare. Ringrazio i miei nonni, Pippi e Tina, di cui conservo ricordi, momenti e insegnamenti preziosi. In particolare, di nonna Tina ricordo la profonda dolcezza nello sguardo, nelle parole e nei gesti; un'anima pura, sempre gentile e disponibile ad aiutare il prossimo anche a discapito di sé stessa. Di nonno Pippi ricordo l'educazione, l'impegno e il rigore con cui faceva le cose che lo contraddistinguevano, l'amore per i film western e l'affetto nei suoi occhi quando ci vedeva.

Ringrazio la mia migliore amica, Gaia, una ragazza forte ma un po' sbadata che mi ha sempre sostenuta e incoraggiata sopportando le mie lamentele per gli esami e le mie ansie. Ma non solo, la ringrazio per tutti questi lunghi anni insieme, nonostante il tempo passi e si cambi inevitabilmente, restiamo sempre anime affini. Il tempo ci ha fatto ritrovare nel momento giusto e ci ha unito più che mai. Spero che possa sempre trovare la sua strada verso la felicità, che sia nelle lezioni di chitarra o nelle passeggiate a San Giovanni, spero si ricordi che la felicità è spesso nelle piccole cose che ci fanno sorridere dopo una lunga giornata di lavoro, nelle persone sempre presenti pronte anche solo a condividere una chiacchierata pur di stare con noi. Spero che la vita le dia tutte le soddisfazioni possibili, senza rimpianti ma solo con belle esperienze da ricordare.

Ringrazio una delle mie amiche più importanti, Raggio di Sole, ritrovate per caso in casa insieme ma una delle coincidenze più belle della tua vita, sicuramente. Ricordo quando ti diedi questo soprannome perché non sapevo ancora il tuo nome ma mi sembravi solare proprio come un raggio di sole. Abbiamo legato fin da subito, un sorriso, un "dove appoggio le pentole" ed eravamo già sul tavolo della cucina con te che mi mettevi lo smalto dicendomi "Sara non ti muovere. Sii delicata". Lo sguardo di quando succedeva qualcosa e facevi ricadere lo sguardo fulminante su di me (questo tutt'ora). La tua voce soave che tutt'ora allietta i miei riposini pomeridiani o le cene in cucina. Ma, a parte gli scherzi, sei indubbiamente una delle persone più importanti della mia vita. Siamo poli completamente opposti eppure ci ritroviamo perfettamente. Litighiamo come una coppia di 70enni, ci punzecchiamo, ci diamo fastidio eppure non cambierei neanche un singolo momento... a parte quando non mi parli perché ti dico di no al dolce serale (spoiler: si va a prendere comunque). Conserverò per sempre le merende passate insieme, le compere, le colazioni da Andy il giovedì mattina prima del mercato, la tua faccia di sfiducia quando cucino un dolce ma poi, sorprendentemente, ti piace oppure come ti si illumina il volto quando ti dico di sì a qualcosa o, ancor meglio, quando non ti lascio completare la frase che so già cosa mi stai per chiedere. Sono sicura che la nostra amicizia, indipendentemente dal fatto che condividiamo un tetto o meno, indipendentemente dalle circostanze, rimarrà sempre forte e ben salda. Anche perché nessuno ti fa una torta pasticciotto meglio della mia.

Ringrazio Angela, anche lei una delle persone conosciute per caso condividendo

un tetto ma diventata subito importante. Una persona sempre sorridente, solare, con pessime battute e molesti rumori da pilates la mattina presto. La tua spontaneità e genuinità attraggono le persone, come anche il tuo costante pensiero a rendere felici i tuoi amici che ti contraddistinguono. Non dimenticherò mai gli aperitivi homemade sul tuo balcone, le cenette improvvisate, le serate karaoke in camera tua, le vacanze in Calabria e soprattutto non dimenticherò mai la splendida persona che ho avuto la fortuna di incontrare e di avere come amica. Spero che tu possa sempre trovare la tua strada, senza fretta, senza pretese. Che sia adesso o tra un anno, l'importante è cercare di essere felici nel tumulto della vita ricordandoci che la cosa importante sono le esperienze che si fanno, le persone che si incontrano. Ansie e paure alla fine scivolano via e quel che resta sono i bei momenti passati insieme.

Ringrazio Annalisa, prima persona incontrata qua a Bari. La persona più energica che conosca. Ci siamo avvicinate con il nostro tempo ma poi non ci siamo mai allontanate. Mi hai insegnato tanto e te ne sarò infinitamente grata. In questi anni ti ho vista raggiungere grandi traguardi, ti ho vista piangere, ti ho consolata, ti ho vista triste e poi felice ma sei sempre rimasta la persona più forte che io conosca. Meriti veramente tanto, meriti tutta la felicità che la vita può offrire e sono immensamente felice quando ti vedo sorridere, energica e solare. Spero tu possa rimanere sempre così, forte e inarrestabile.

Ringrazio Ludo, un'amicizia nata inaspettatamente ma una delle più belle e genuine avute qua a Bari. Sei un'anima buona ed emotiva, pronto sempre a farti in quattro per tutti. Spero tu possa continuare ad avere realizzazioni personali e che tu possa sempre essere circondato da persone che ti vogliono bene genuinamente e che ti fanno stare bene.

Ringrazio il mio gatto Spyros. Compagno di tutta la mia adolescenza. Sempre alla ricerca di cibo e della sua tranquillità. Mai troppo affettuoso, ma sempre nel mio cuore. Hai vissuto una vita piena. Sei stato amato e accudito. Avrai sempre un pezzo importante nel mio cuore e nella mia memoria.

Infine, ringrazio tutti gli amici qui non menzionati con cui passo bei momenti di spensieratezza e condivisione. Avete tutti un posto speciale.

A tutte le persone qui menzionate e non, auguro di fare le proprie esperienze senza rimorsi, di viverli bene quei momenti di vera felicità, di non farsi prendere

da ansie e paure e di cercare, quindi, la calma per godersi al meglio la vita. Non importa che siate lenti o veloci, eccellenti o non, ciò che conta è avere ricordi da raccontare e persone con cui condividere questi momenti importanti e unici. Cercate sempre di fare ciò che vi rende felici senza la paura del giudizio degli altri, senza l'ansia delle aspettative, la competizione per l'eccellenza o le paranoie per il futuro. Se si continua sul proprio cammino, ogni cosa verrà da sé, ricordandoci sempre che ognuno ha i suoi tempi, i suoi obiettivi, le proprie paure e le proprie ansie. Bisogna sempre avere il coraggio per andare avanti.