



Dipartimento di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi Di Laurea

Estrazione di Relazioni con Supervisione a Distanza e Link Prediction

Laureando

Enrico Verdolotti

Matricola 283812

Relatore

Prof. Paolo Merialdo

Anno accademico 2019/2020

“Far better it is to dare mighty things, to win glorious triumphs, even though checkered by failure, than to take rank with those poor spirits who neither enjoy much nor suffer much, because they live in the gray twilight that knows neither victory nor defeat.”

– **Theodore Roosevelt, Strenuous Life**

Ringraziamenti

Ringrazio i miei genitori che hanno consentito tutto questo, mia sorella che anche se lontana è sempre stata vicina nei momenti peggiori ed i miei amici che a volte inconsapevoli mi hanno mostrato quello che non vedevo proprio davanti al mio naso. Ringrazio i miei amici e colleghi dell'università vecchi e nuovi con cui con cui ho condiviso parti diverse del percorso, specialmente Giulia per aver reso le mie preoccupazioni inutili ed aver creduto più di me nelle mie capacità. Un ringraziamento particolare al mio ex-relatore della tesi triennale, Franco Milicchio, per aver riaccesso quella passione necessaria per continuare il mio percorso di studi.

Ringrazio il mio relatore Paolo Merialdo che nonostante le difficoltà portate da questo particolare periodo di pandemia globale ha comunque seguito lo svolgimento della tesi ed un un doveroso ringraziamento anche al dottorando Andrea Rossi per essere stato sempre disponibile, rapido nelle risposte ed avermi incoraggiato quando era necessario. Infine ringrazio Matteo Cannaviccio per il materiale e le spiegazioni che mi ha fornito.

Prefazione

I grafi di conoscenza, o *Knowledge Graphs*, sono repository di conoscenza in cui entità, reali o astratte del mondo reale, sono rappresentate come nodi connessi da archi a formare triple, dette fatti, che incarnano i loro specifici collegamenti semantici. I grafi di conoscenza hanno acquisito rilievo nell'informatica grazie alla loro versatilità e scalabilità, e giocano un ruolo centrale nella trasformazione da World Wide Web a *Semantic Web* [6].

La costruzione di un Knowledge Graph ed il suo arricchimento sono spesso effettuati con tecniche di Information Extraction, che ottengono dati strutturati a partire da sorgenti non strutturate o semi-strutturate. Queste tecniche si concentrano spesso su fonti testuali, individuando le principali entità (Named Entity Recognition) ed associando alle formule che le collegano nel testo le corrispondenti relazioni semantiche nel grafo (Relation Extraction).

Ad esempio, dalla frase "*Barack got married with Michelle in 1992*" un sistema di Relation Extraction potrebbe individuare la formula lessicale (*phrase*) "got married with"; avendola già vista occorrere tra entità collegate nel grafo dalla relazione "spouse", il sistema estrarrà il nuovo fatto:
<Barack, spouse, Michelle> da aggiungere al grafo.

Malgrado la loro efficacia, i sistemi di Relation Extraction soffrono di alcuni problemi che ad oggi rimangono sostanzialmente irrisolti: tra questi, una sfida centrale è come gestire formule lessicali che non corrispondono a nessuna relazione del grafo. Sia data ad esempio la frase "*Barack met Hillary in 1989*": la formula lessicale "met" esprime un tipo di relazione che non viene gestita dal grafo di conoscenza. Non potendo saperlo, il sistema di Relation Extraction lo prenderà comunque in considerazione e lo assocerà alla relazione che più spesso, nel grafo, collega entità che nel testo vengono connesse da quella formula testuale; nell'esempio precedente potrebbe essere "spouse". In ultima analisi, quindi, saranno estratti fatti errati.

La mia tesi descrive un approccio innovativo per affrontare il problema, proponendo un nuovo framework *Deflector* che si avvale di modelli di Link Prediction per distinguere, in un sistema di Relation Extraction, le formule lessicali non informative da quelle informative.

Il task di Link Prediction consiste nella deduzione di nuovi archi a partire da quelli già a disposizione nel grafo. Noti ad esempio:

<Barack, born_in, Honolulu> e <Honolulu, located_in, USA>

è possibile inferire con ragionevole certezza <Barack, nationality, USA>. Negli ultimi anni la comunità di ricerca ha sviluppato numerosi modelli di Link Prediction che hanno dimostrato risultati promettenti nella stima della plausibilità di fatti non presenti nel grafo.

Il framework *Deflector* combina i due approcci per ottenere risultati che, in isolamento, nessuno dei due riuscirebbe a raggiungere. Da un lato infatti le tecniche di Relation Extraction possono estrarre da fonti testuali dei fatti che un modello di Link Prediction non sarebbe in grado di inferire; dall'altro, un approccio di Link Prediction può stimare la plausibilità dei fatti estratti e permette quindi di validare sui grandi numeri la bontà delle associazioni formule-relazioni del sistema di Relation Extraction.

Nel lavoro di tesi sono stati effettuati esperimenti su sistemi di Relation Extraction e di Link Prediction noti in letteratura, ottenendo risultati promettenti che attestano la bontà dell'approccio Deflector.

La tesi è strutturata come segue. Nel **Capitolo 1** viene presentato il contesto e gli obiettivi di ricerca in una breve introduzione; nel **Capitolo 2** si forniscono elementi di base sui Knowledge Graph come strutture dati in senso generale; nel **Capitolo 3** si descrivono le tecniche di Relation Extraction, con particolare riferimento a quelle basate su apprendimento automatico e su tecniche di supervisione a distanza; il **Capitolo 4** è dedicato all'ambito di ricerca di Link Prediction; il **Capitolo 5** è incentrato sul framework *Deflector*, descrivendone i concetti fondanti e fornendone aspetti progettuali ed implementativi; il **Capitolo 6** presenta gli esperimenti effettuati, descrivendo il setting sperimentale utilizzato e confrontando in modo critico diverse alternative architettoniche; il **Capitolo 7**, infine, riporta osservazioni conclusive ed esplora possibili sviluppi futuri.

Indice

1	Introduzione	9
2	Knowledge Graph	15
2.1	Cos'è un grafo?	15
2.2	Grafi di conoscenza	16
2.2.1	Breve storia dei Knowledge Graphs	19
2.2.2	Utilizzi dei Knowledge Graphs	20
2.2.3	Principali KG	20
2.2.4	Incompletezza	22
3	Relation Extraction	25
3.1	Estrazione di pattern	25
3.2	Estrazione di relazioni	26
3.3	Distant Supervision	28
3.3.1	Etichettatura automatica	29
3.3.2	Vantaggi e Svantaggi	29
3.4	Caso d'uso: Lector	31
4	Link Prediction	33
4.1	Deduzione automatica	33
4.2	Breve storia	34
4.3	Funzionamento Generale	35
4.3.1	<i>Funzione di score</i>	36
4.3.2	Addestramento	38
4.3.3	Valutazione	38
4.3.4	ComplEx	40

5 Deflector	42
5.1 Scartare i pattern errati	42
5.2 Architettura proposta	44
5.3 Scoprire nuovi pattern	45
5.4 Implementazione	45
5.4.1 Wrappers	46
5.4.2 Tracciamento pattern	49
5.4.3 Batch processing	49
5.4.4 Valutazione pattern	50
6 Risultati sperimentali	52
6.1 Dati di test	52
6.2 Test effettuati	54
7 Conclusioni e sviluppi futuri	58

Elenco delle figure

1.1	Etichettatura errata. In alto a sinistra, la tabella con ciò che si è estratto dal testo, cioè le entità separate dal pattern tra virgolette. In alto a destra, la tabella con la nostra base di conoscenza, cioè i fatti di cui siamo al corrente. In basso il risultato dell'etichettatura del pattern "met" che viene etichettato con "spouse" 5 volte su 7.	13
2.1	Alcune tipologie di grafi.	15
2.2	Famiglia Obama. Un semplice knowledge graph per rappresentare la famiglia Obama.	16
2.3	KB vs. KG. A sinistra una base di conoscenza, a destra il grafo di conoscenza equivalente. Fonte: [25].	18
2.4	Storia dei Knowledge Graph. Fonte: [25]	20
2.5	Famiglia Musk. In questo caso in rosso vi sono alcuni collegamenti mancanti che rappresentano l'incompletezza della conoscenza.	22
2.6	Un KG di grandi dimensioni Le dimensioni variabili dei nodi rispecchiano la cardinalità di archi entranti/uscenti (degree)	24
3.1	Un esempio di Named Entity Resolution. Ogni entità conosciuta viene individuata ed annotata, inclusi i pronomi che possono riferirsi ad entità specifiche.	26
3.2	Lector pipeline. Processo di etichettatura fatti di Lector[11]	31
3.3	Esempio del funzionamento di Lector. All'interno del riquadro tratteggiato è indicata la parte che non è stata implementata per questo progetto.	32
4.1	Inferire un link. Un semplice esempio concettuale di link prediction, la <i>nationality</i> di Barack Obama può essere inferita sapendo che Obama è nato ad Honolulu e che Honolulu si trova negli U.S.A.	33

4.2	Knowledge Graph Embeddings. Ad ogni nodo del grafo, associato ad una entità, viene associato un vettore numerico.	36
4.3	Tassonomia dei modelli di LP. [40] Le frecce tratteggiate indicano che il metodo puntato è costruito sul metodo da cui parte la freccia, o generalizzando o specializzando la definizione della funzione di scoring.	37
5.1	Lector idea. Le barre in grigio rappresentano tutti i pattern di coppie che non è stato possibile etichettare con una relazione, le barre colorate rappresentano i pattern etichettati con una certa relazione. Sono mostrati gli effetti di diversi sottocampionamenti e come questo influisca sull'apprendimento dei pattern.	43
5.2	Deflector idea Nel riquadro a sinistra, PX rappresenta il pattern, RX la relazione associata al pattern, e la lista di coppie sono le istanze della relazione RX estratte perché trovate separate dal pattern PX. Nel riquadro a destra, link prediction ha espresso una predizione per ogni coppia ed è stata verificata l'accuratezza del pattern in base a quante relazioni predette coincidono con quella associata.	44
5.3	Architettura di base Questo schema mostra i concetti principali del meta-modello proposto. La struttura più importante è pattern blacklist che conterrà i pattern considerati non buoni per estrarre relazioni.	45
5.4	Architettura V2 Lo schema mostra l'effetto delle modifiche al modello base per cercare di scoprire nuovi pattern.	46
6.1	Costruzione dei dataset Viene eseguito il processo inverso a Distant Supervision sui dati originali per ottenere i dati necessari ad addestrare Link Prediction.	53
6.2	Accuratezza. Ogni barra è relativa ad una relazione, viene mostrata l'accuratezza con cui un pattern errato viene effettivamente scartato. Questi risultati sono stati ottenuti con una soglia per blacklisting e discovery = 0.8	55
7.1	Google Knowledge Graph. La pagina di ricerca mostra lateralmente le informazioni su ciò che si è cercato e tutti i possibili collegamenti.	58
7.2	I knowledge graph per la lotta al COVID-19. Fonte: <i>The Graph Lounge</i> . https://theographlounge.com/knowledge-graphs-in-the-fight-against-covid-19	59

Capitolo 1

Introduzione

Con la continua evoluzione dell’informatica ed il suo divenire sempre più pervasiva nella vita quotidiana, la gestione delle vaste collezioni di dati utilizzate dai sistemi informativi è un tema sempre più critico. L’avvento dell’era dei Big Data, in cui l’informazione digitale viene generata e consumata con volumi, velocità e varietà senza precedenti, comporta infatti la costruzione di imponenti sorgenti di informazione che necessitano di aggiornamenti ed integrazioni per evitare che diventino obsoleti e, col passare del tempo, inutilizzabili.

Tali nuovi problemi sono stati e continuano tuttora ad essere affrontati dalla comunità scientifica con la formulazione nuove metodologie in ambiti di ricerca come Information Extraction [12] e Big Data Integration [14], e sfruttando le opportunità offerte dall’avvento di approcci innovativi come il Machine Learning e il Deep Learning [22].

Tra le svariate tipologie di sorgenti di informazione, i grafi di conoscenza (Knowledge Graphs) richiedono per loro stessa natura un continuo processo di arricchimento e manutenzione. I grafi di conoscenza rappresentano in formato strutturato conoscenza del mondo reale, e modellano tramite nodi e archi entità (come persone, luoghi, concetti) e le relazioni semantiche che li collegano. La maggior parte dei grafi di conoscenza quindi possono essere visti come liste di ”fatti” relazionali binari, come ad esempio (`<Barack>, <birthPlace>, <Honolulu>`): tale fatto esprime la relazione ”birthPlace” (ovvero, luogo di nascita) tra le entità Barack Obama ed Honolulu ed equivale quindi alla frase ”Barack Obama è nato ad Honolulu”. I grafi di conoscenza sono oggi impiegati in innumerevoli applicazioni come sistemi di *Question Answering*; sistemi di raccomandazione automatica;

data cleaning; e molto altro.

Un grafo di conoscenza è utile solo nella misura in cui lo è la conoscenza che esso contiene e rappresenta. Sfortunatamente, tutti i grafi di conoscenza soffrono del problema dell'incompletezza, poiché contengono solo una piccola parte di tutti i fatti che rappresentano il mondo reale. Tra le tecniche più efficaci ed utilizzate per far fronte a questo problema, degli approcci di Relation Extraction possono estrarre nuovi fatti a partire da testo in linguaggio naturale. I metodi di Relation Extraction, in altri termini, consentono di estrarre e formulare in modo strutturato il contenuto semantico delle frasi: ad esempio, il fatto che una certa persona sia impiegata presso una particolare organizzazione, o che un luogo geografico si trovi in una particolare regione. Un vantaggio fondamentale delle tecniche di Relation Extraction è dato dall'enorme disponibilità su web di testo non strutturato da cui estrarre relazioni (basti pensare a Wikipedia).

L'obiettivo principale di questo lavoro è capire se è possibile sfruttare un modello di Link Prediction (LP) per *migliorare* le prestazioni di un modello di Relation Extraction (RE). *Ma migliorare come?*

I sistemi di Relation Extraction generalmente operano creando associazioni tra le relazioni del grafo e le formule lessicali che, nel testo, collegano le varie entità (precedentemente isolate con tecniche di Named Entity Recognition). Quando in una frase si osservano due entità collegate da una certa formula lessicale, il sistema è in grado di risalire alla relazione che nel grafo equivale a quella formula lessicale, e quindi costruire un nuovo fatto. La maggior parte dei sistemi di Relation Extraction oggi sfruttano forme di apprendimento automatico, e possono essere classificati in base alla specifica metodologia di addestramento utilizzata. Si osservano quattro paradigmi di apprendimento principali: (i) approcci supervisionati classici; (ii) approcci non supervisionati; (iii) approcci di bootstrap supervision; (iv) approcci di distant supervision.

Negli approcci **supervisionati** classici, ciascuna frase all'interno del testo, prima deve essere etichettata a mano, con le entità e la relazione tra le entità presenti in essa. Ad esempio il corpora¹ NIST Automatic Content Extraction (ACE) RDC del 2003 e quello del 2004, includono più di 1.000 documenti in cui coppie di entità sono state etichettate con 5-7 tipi di relazioni e con 23-24 sotto-relazioni, totalizzando 16,771 istanze di relazioni. I sistemi

¹collezione di testi o documenti pre-etichettati

ACE successivamente possono estrarre una grande varietà di features, lessicali, sintattiche e semantiche ed utilizzare classificatori supervisionati per estrarre la relazione che si suppone essere espressa fra ogni coppia di entità data nell'insieme di test. Tuttavia, l'estrazione di relazioni supervisionata, soffre di un certo numero di problemi. I dati di addestramento etichettati sono costosi da produrre in termini di tempo e perciò disponibili in quantità limitata. Inoltre, siccome le relazioni sono etichettate su corpus particolari, cioè in genere documenti provenienti da un certo dominio (e.g. papers scientifici, testi giuridici, articoli di giornale), il classificatore risultante tende ad essere alterato, orientandosi verso tale dominio testuale.

Nell'approccio **non supervisionato**, invece, si segue un principio diametralmente opposto al precedente, che prevede di estrarre stringhe di testo comprese fra coppie di entità in grandi quantità di testo, per poi "clusterizzare" (raggruppare per similarità) e semplificare tali stringhe di parole per produrre delle corrispondenze stringa-relazione (Shinyama and Sekine, 2006; Banko et Al., 2007). Gli approcci non supervisionati consentono di utilizzare quantità di dati maggiori ed estrarre grandi quantità di relazioni, tuttavia potrebbe non essere facile trovare le corrispondenze fra tali relazioni e quelle utilizzate in una particolare *knowledge base*.

Nell'approccio comunemente noto come **bootstrap learning**, l'idea chiave è utilizzare un numero ristretto di istanze o pattern "generatori" (Brin, 1998; Riloff e Jones, 1999; Agichtein e Gravano, 2000; Ravichandran e Hovy, 2002; Etzioni et al., 2005; Pennacchiotti e Pantel, 2006; Bunescu e Mooney, 2007; Rozenfeld e Feldman, 2008). Questi pattern "generatori" vengono utilizzati per estrarre altri pattern simili da grandi corpus di documenti che sono poi usati per estrarre più istanze ed a loro volta ancora altri pattern con uno schema iterativo che si ripete. I pattern risultanti spesso soffrono di bassa precisione e variazioni semantiche.

Nell'approccio di **distant supervision**, infine, si propone di generare un training set sfruttando (anche) una fonte di dati alternativa come appunto un grafo di conoscenza. Nel dominio di Relation Extraction l'intuizione di base è che le formule lessicali che occorrono sistematicamente tra entità che, nel grafo, sono connesse da uno stesso tipo di relazione, probabilmente esprimono proprio quella relazione; ad esempio la formula "got married with" occorrerà più frequentemente tra entità che, nel grafo, sono connesse dalla relazione "spouse". Avere a disposizione questo tipo di associazioni

tra formule lessicali e relazioni consente di generare in modo automatico un training set ampio a sufficienza da addestrarare un sistema di Relation Extraction: se una formula lessicale è correlata più fortemente ad una relazione che alle altre, nel training set saranno aggiunte molte istanze contenenti quella formula etichettate con quella relazione, e quindi il modello di Relation Extraction (qualunque esso sia), venendo addestrato su quel training set tenderà a mappare quella formula a quella relazione. In altri termini, questo principio sfrutta sui grandi numeri la conoscenza già presente nel grafo malgrado la sua incompletezza. L'uso di distant supervision ha recentemente acquisito grande popolarità in svariati ambiti di ricerca grazie alla sua maggiore robustezza ai problemi descritti per gli altri approcci sopracitati. In Relation Extraction, però, soffre a sua volta di un problema intrinseco.

È infatti possibile che non si abbia una corrispondenza 1:1 tra formule lessicali e relazioni nel grafo. Se una formula corrisponde semanticamente a nessuna relazione, ma correla con una o più relazioni, questo può portare ad associazioni (e esempi di training) errati, che indurranno all'errore anche il sistema di Relation Extraction. Questo può accadere ad esempio per formule lessicali molto generiche. Si consideri l'esempio in figura 1.1, riguardante la formula lessicale "met".

Le entità che nel testo sono collegate tramite "met" potrebbero mostrare correlazioni inattese con alcune relazioni; nello specifico, "met" correla con la relazione "spouse" molto più che con le altre, perché due persone sposate si sono certamente incontrate almeno una volta. Di conseguenza l'associazione "met"- "spouse" porterà a generare etichettature errate, e questo farà sì che, successivamente, il sistema di Relation Extraction addestrato su quel training set considererà come sposate delle persone che si sono in realtà soltanto incontrate. Questo esempio non è solo speculativo: al contrario, è stato osservato nella pratica in sistemi reali come il software di Relation Extraction: Lector [11].

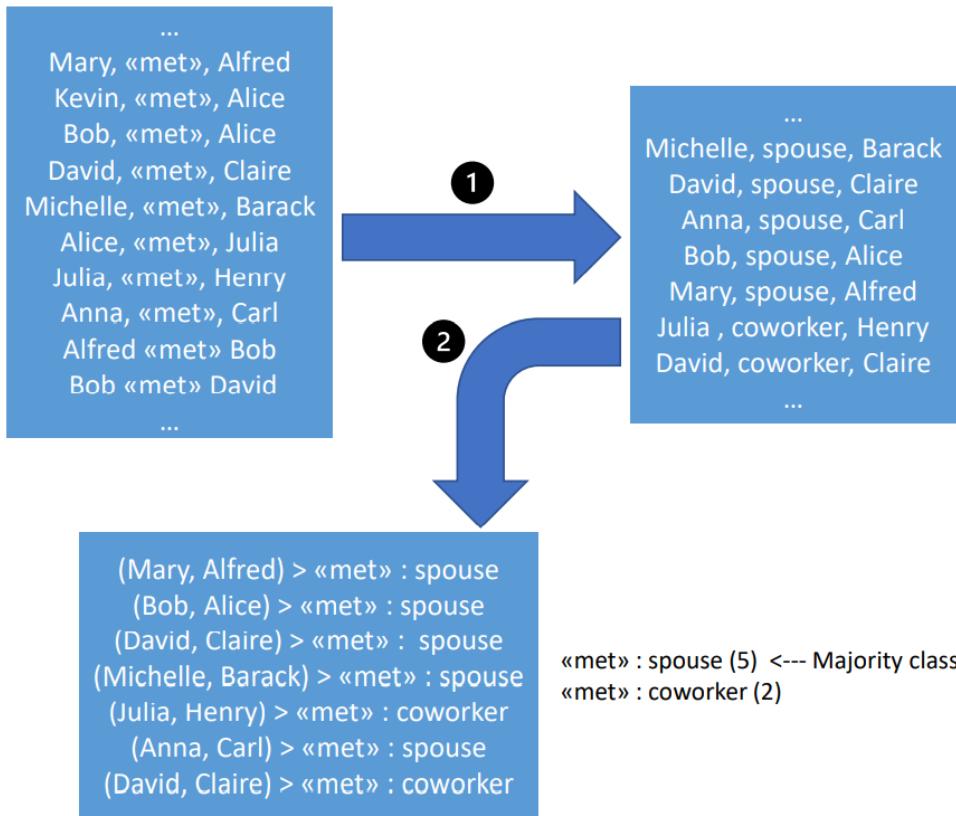


Figura 1.1: **Etichettatura errata.** In alto a sinistra, la tabella con ciò che si è estratto dal testo, cioè le entità separate dal pattern tra virgolette. In alto a destra, la tabella con la nostra base di conoscenza, cioè i fatti di cui siamo al corrente. In basso il risultato dell'etichettatura del pattern "met" che viene etichettato con "spouse" 5 volte su 7.

Questo è il problema centrale che la mia tesi si propone di affrontare. Il mio lavoro studia la possibilità di sfruttare modelli di Link Prediction (LP) per individuare le associazioni errate nei sistemi di Relation Extraction (RE) basati su distant supervision: questo comporterà in ultima analisi un miglioramento nelle prestazioni di tali sistemi.

I modello di Link Prediction sono in grado di inferire nuovi fatti in un grafo dall'interno, deducendoli a partire dalle informazioni già contenute all'interno del grafo. Una delle loro features principali è la capacità di valutare la plausibilità di nuovi fatti, permettendo quindi di effettuare una scrematura di fatti completamente inverosimili.

Nel nostro scenario un modello di Link Prediction può essere impiegato in due modi distinti:

- A priori: con questo approccio si usa il modello di Link Prediction per validare le associazioni tra formule lessicali e relazioni direttamente prima della generazione del training set. In questo modo il training set conterrà fatti generati esclusivamente da mapping "sicuri".
- A posteriori: con questo approccio i mapping errati vengono individuati in fase di estrazione, quindi dopo che il sistema di Relation Extraction è già stato addestrato. Si utilizza il modello di Link Prediction per validare i fatti estratti dal sistema di Relation Extraction: se i fatti estratti a partire da una specifica formula lessicale appaiono tendenzialmente non plausibili, è possibile scartare il pattern o richiedere una verifica manuale da un operatore umano.

Nel corso di questo progetto si è posta l'attenzione maggiormente sul secondo approccio, seppur non ignorando completamente il primo. Questo principalmente perché il primo richiederebbe che i pattern siano in una forma semplice come "met" usato nell'esempio e quindi raggruppabili perché identici; sistemi di RE più avanzati invece potrebbero catturare un maggior numero di features molto più dettagliate e quindi produrre pattern molto più variegati anche se etichettabili con la stessa relazione. Quindi per affrontare il problema in modo totalmente indipendente dal modello in questa tesi verrà posta l'attenzione sul secondo approccio.

Capitolo 2

Knowledge Graph

2.1 Cos'è un grafo?

Un grafo è una struttura largamente usata in diversi contesti. Molto semplice e flessibile, ne esistono innumerevoli varianti ma tutte condividono due componenti principali: *I nodi e gli archi*. Ancora più semplicemente, un nodo è di solito rappresentato con un pallino ed un arco è una linea che unisce due nodi. Nonostante la semplicità, o forse grazie ad essa, questa struttura è estremamente potente per una varietà di applicazioni. Nell'ambito della rappresentazione della conoscenza e del ragionamento, un grafo di conoscenza è un sistema per memorizzare dati che utilizza una struttura a grafo.

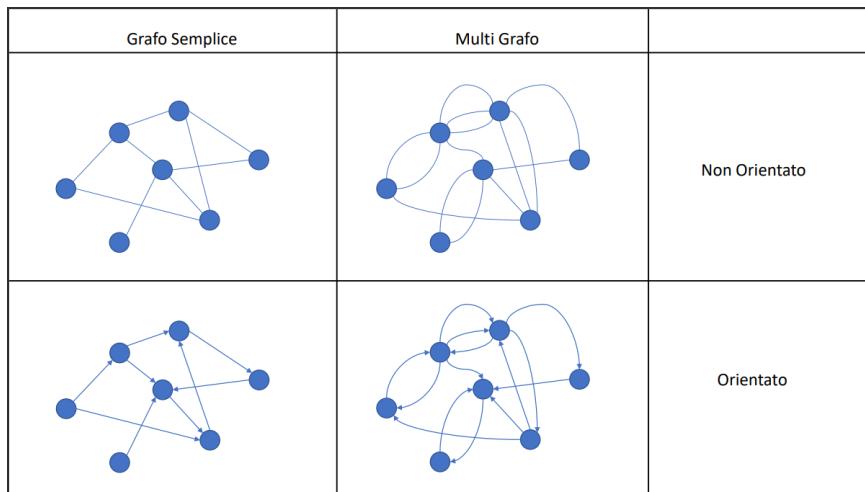


Figura 2.1: Alcune tipologie di grafi.

2.2 Grafi di conoscenza

I grafi di conoscenza o *knowledge graphs* (KGs) sono rappresentazioni strutturate dell'informazione del mondo reale. In un KG i *nodi* rappresentano delle entità reali come ad esempio persone, posti o concetti astratti. Le *etichette* sono i possibili tipi di relazioni che possono connettere le entità; e gli *archi* sono i fatti specifici che connettono due entità con una relazione. Molto spesso gli archi hanno una direzione per poter meglio rappresentare la semantica della relazione che descrivono. Se ad esempio (figura 2.2), per la relazione **spouse** non è importante la direzione della freccia, lo è ad esempio per **parent**.

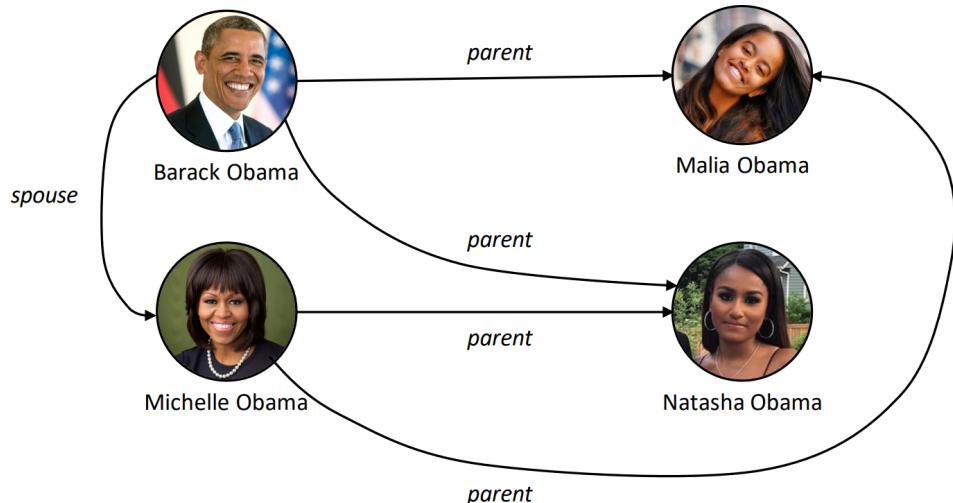


Figura 2.2: **Famiglia Obama.** Un semplice knowledge graph per rappresentare la famiglia Obama.

Più formalmente, un Knowledge Graph può essere visto come una tripla **(E, R, G)** dove:

- **E** è l'insieme di entità;
- **R** è l'insieme di relazioni;
- **G** è l'insieme di archi esistenti, usati per connettere le varie entità tra loro.

A partire da questa semplice definizione, come precedentemente accennato, esistono molte varianti, sia per caratteristiche strutturali che per tipolo-

gia di contenuto. Per esempio, alcuni KG supportano attributi letterali assegnabili alle entità e/o agli archi, altri KG sviluppano ontologie che usano per descrivere gerarchie e tipizzazione delle entità, altri ancora KG supportano persino iper-archi che collegano più di due nodi contemporaneamente.

Allo stesso modo, alcuni KG sono verticali su specifici argomenti, come WordNet che opera esclusivamente in ambito di grammatica e lessico; altri puntano a una rappresentazione trasversale di conoscenza dal mondo reale, come DBpedia o Wikidata.

Perciò esiste un panorama abbastanza ampio riguardo le possibili configurazioni, fortunatamente il progetto di questa tesi è stato svolto sulla tipologia più semplice di KG mostrata qui negli esempi e costituita da una semplice collezione di archi, con riferimento alla figura 2.1

Nel resto di questa sezione parleremo brevemente di come i Knowledge Graphs siano nati e stiano acquisendo sempre maggior rilevanza nell'informatica (sottosezione 2.2.1); delle applicazioni pratiche in cui sono oggi ampiamente impiegati (sottosezione 2.2.2); presenteremo una breve panoramica dei più importanti KG attuali (sottosezione 2.2.3); infine, descriveremo quali limitazioni rendano oggi i KG ancora oggetto di ricerca e sviluppo (2.2.4).

Come descritto da Shaoxiong Ji, Shirui Pan et al. nell'articolo [25], comprendere la conoscenza umana è uno dei campi di ricerca dell'intelligenza artificiale (AI). La rappresentazione della conoscenza e del ragionamento, ispirata alla capacità umana di risolvere i problemi, cerca di rappresentare la conoscenza per consentire a sistemi intelligenti di ottenere l'abilità di risolvere compiti complessi [34]. Recentemente, i grafi di conoscenza, hanno attirato grande attenzione da parte della ricerca sia accademica che industriale [13]-[23]. I grafi di conoscenza *knowledge graphs* (KGs) sono rappresentazioni strutturate dell'informazione del mondo reale. Un *knowledge graph* è una rappresentazione strutturata di fatti, composta da entità, relazioni e descrizioni semantiche. Le **entità** possono essere oggetti del mondo reale o concetti astratti, le **relazioni** rappresentano le relazioni fra entità e la descrizione semantica delle entità, e le loro relazioni contengono tipi e proprietà con un significato ben definito. Grazie alla loro capacità di modellare strutturalmente dati complessi in modo che siano leggibili da una macchina, i KGs sono oggi largamente impiegati in vari domini, che vanno dal *question answering*, *l'information retrieval* e nei sistemi di raccomandazione basati sui contenuti, inoltre sono di vitale importanza per qualsiasi

progetto relativo al *semantic web* [24] Il termine knowledge graph, è sinonimo di knowledge base (KB) (o **conoscenza di base** già introdotta nella parte iniziale di questa tesi), con una leggera differenza. Un knowledge graph può essere visto come un grafo quando si considera la sua struttura [45]. Quando invece comprende anche una semantica formale, può essere considerato knowledge base per l'interpretazione e l'inferenza sui fatti [8]. Un esempio di knowledge base e knowledge graph sono mostrati in figura 2.3

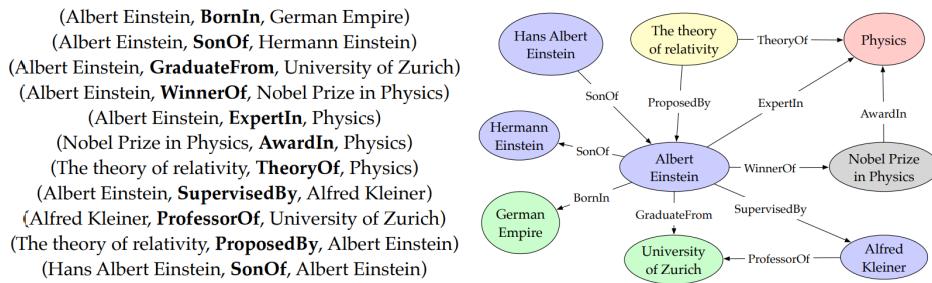


Figura 2.3: **KB vs. KG.** A sinistra una base di conoscenza, a destra il grafo di conoscenza equivalente. Fonte: [25].

Recenti scoperte nei vari campi di ricerca sui knowledge graph si sono concentrate sull'apprendimento automatico della rappresentazione della conoscenza meglio noto come **knowledge representation learning** (KRL) o anche come **knowledge graph embeddings** (KGE) cercando di mappare entità e relazioni su vettori a bassa dimensionalità, con l'intenzione di catturare i loro significati semantici. Altri tipi di ricerca specifici includono, il *knowledge graph completion* (KGC) con l'obiettivo di trovare collegamenti, e quindi fatti, mancanti in un KG; classificazione delle triple; riconoscimento delle entità nel testo ed **estrazione di relazioni**. I modelli in grado di sfruttare la conoscenza beneficiano dell'integrazione con informazione che sia eterogenea, cioè non bloccata su specifici domini, oltre che della conoscenza in lingue differenti. Per questo motivo, molte applicazioni del mondo reale come i *recommendation system* ed il *question answering*, sono state portate verso un crescente utilizzo grazie alle abilità acquisite paragonabili alla comprensione del senso comune ed al ragionamento. Alcuni prodotti del mondo reale che implementano tali tecnologie, per esempio: Microsoft's Satori ed il Google's Knowledge Graph [13], hanno mostrato una grande capacità di fornire servizi ancora più efficienti.

2.2.1 Breve storia dei Knowledge Graphs

Come ben riassunto da Shaoxiong Ji e Shirui Pan nel survey [25], la rappresentazione della conoscenza è stata per un lungo periodo storico oggetto di sviluppo nel campo della logica inferenziale e dell'intelligenza artificiale (AI). L'idea di rappresentare graficamente la conoscenza risale inizialmente al 1956 così come il concetto di rete semantica proposta da Ritchens [39], mentre la conoscenza della logica simbolica risale al 1958 con il General Problem Solver [34]. La knowledge base venne usata in principio con i sistemi basati sulla conoscenza per il ragionamento e la risoluzione di problemi. MYCIN [43] è uno dei più famosi *sistemi esperti* basato su regole, in grado di effettuare diagnosi mediche con una base di conoscenza di 600 regole. In seguito, la comunità per la rappresentazione della conoscenza umana assistette allo sviluppo dei linguaggi frame-based, rule-based ed a rappresentazione ibrida. All'incirca alla fine di questo periodo il progetto Cyc¹ iniziò a cercare di assemblare la conoscenza umana. Il formato *resource description framework* (RDF)² per la descrizione di entità e relazioni ed il *Web Ontology Language*³ sono stati a loro volta rilasciati e divenuti importanti standard per il *Semantic Web*⁴. In seguito molte knowledge bases ed ontologie sono state pubblicate, come WordNet, DBPedia, YAGO e Freebase. Nel 1988 Stokman e Vries [45] proposero un'idea moderna di come strutturare la conoscenza in un grafo. Tuttavia fu solo nel 2012 che il concetto di knowledge graph guadagnò grande popolarità a partire dal primo lancio integrato con il motore di ricerca di Google, in cui il framework di fusione della conoscenza chiamato Knowledge Vault fu proposto per costruire un knowledge graph su larga scala. Un breve riassunto delle tappe storiche dello sviluppo dei KGs è mostrato in figura 2.4

¹<https://www.cyc.com>

²<http://w3.org/TR/1999/REC-rdf-syntax-19990222>

³<http://w3.org/TR/owl-guide>

⁴<http://w3.org/standards/semanticweb>

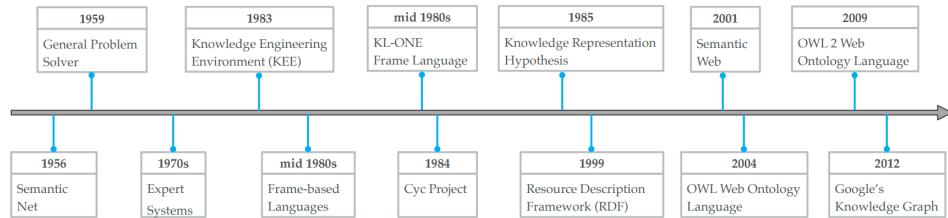


Figura 2.4: Storia dei Knowledge Graph. Fonte: [25]

2.2.2 Utilizzi dei Knowledge Graphs

I KG sono impiegati in numerosi ambiti dell'informatica nei quali è necessario aver accesso ad una conoscenza del mondo reale. Parte del loro successo è dato dal fatto che la struttura a grafo li rende contemporaneamente una rappresentazione sia naturale e intuitiva per l'uomo, che facilmente visitabile e gestibile tramite software come un comune Graph DB, al punto che esiste un particolare campo dell'algebra lineare che sfrutta le operazioni e le strutture elementari largamente usate in algebra lineare appunto, come matrici, vettori, prodotto fra matrici per implementare qualsiasi tipo di algoritmo su grafi, visita, modifica archi ecc. rendendo possibile un notevole incremento delle prestazioni computazionali [26].

Come riportato da Shaoxiong Ji e Shirui Pan in [25] una buona conoscenza opportunamente strutturata può essere utile per le possibili applicazioni di intelligenza artificiale. Tuttavia, come integrare tale conoscenza simbolica nel contesto delle applicazioni del mondo reale resta una sfida. Le applicazioni inerenti ai knowledge graphs comprendono due possibilità: 1) Applicazioni "intrinseche" al KG come Link Prediction di cui sarà discusso in un capitolo dedicato e la *named entity recognition*; e 2) Applicationi "estrinseche", che includono la relation extraction a cui è dedicato un'altro capitolo di questa tesi ed applicazioni, basate su conoscenza, meno in voga come il question answering ed i recommender systems.

2.2.3 Principali KG

Alcuni dei KG maggiormente conosciuti sono:

- **DBpedia:** DBpedia è il più popolare e prominente KG nel cloud LOD (Linked Open Data). Il progetto è stato avviato da ricercatori della Libera Università di Berlino e dell'Università di Lipsia, in collaborazione

con OpenLink Software. Dalla prima versione pubblica nel 2007, DBpedia viene aggiornato all’incirca una volta all’anno. DBpedia è creato da una informazione strutturata estratta automaticamente contenuta in Wikipedia, come da tabelle di infobox, informazioni sulla categorizzazione, coordinate geografiche e link esterni. A causa del suo ruolo come hub di LOD, DBpedia ne contiene molti collegamenti ad altri set di dati nel cloud LOD come Freebase, OpenCyc, UMBEL, GeoNames, Musicbrainz, CIA World Factbook, DBLP, Progetto Gutenberg, DBtune Jamendo, Euro stat, Uniprot e Bio2RDF. DBpedia è ampiamente utilizzato nella comunità di ricerca sul Web semantico, ma è rilevante anche in ambito commerciale: le aziende lo usano per organizzare i propri contenuti, come la BBC e il New York Times.

- **Freebase:** Freebase è un KG annunciato da Metaweb Technologies, Inc. nel 2007 ed è stato acquisito da Google Inc. il 16 luglio 2010. Contrariamente a DBpedia, Freebase aveva fornito un’interfaccia che ha consentito agli utenti finali di contribuire al KG modificando i dati strutturati. Oltre tutto dati forniti dagli utenti, dati integrati in Freebase da Wikipedia, NNDB, FMD e MusicBrainz. Freebase utilizza un modello di grafo proprietario per la memorizzazione di statements anche complessi. Il 16 dicembre 2014, il team di Freebase ha annunciato che Freebase avrebbe interrotto i suoi servizi il 30 giugno, 2015. Wikimedia Deutschland e Google prevedevano di integrare i dati di Freebase in Wikidata ed è stato in sviluppo uno strumento per questo fino ad agosto 2015. Successivamente è stato chiuso il sito web di Freebase.
- **OpenCyc:** Il progetto Cyc è iniziato nel 1984 come parte di Microelectronics e Computer Technology Corporation. Lo scopo di Cyc è quello di immagazzinare (in modo processabile da una macchina) milioni di fatti di senso comune come: “Ogni albero è una pianta”. Mentre il focus di Cyc nei primi decenni era sull’inferenza e il ragionamento, un lavoro più recente pone l’accento sull’interazione umana come la creazione di sistemi di risposta alle domande basati su Cyc. Poiché Cyc è proprietario, una versione più piccola del KG chiamata OpenCyc è stato rilasciato sotto licenza Apache open source. Nel luglio 2006, venne reso pubblico ResearchCyc per la community di ricerca, contenente

più fatti di OpenCyc.

- **Wikidata:** Wikidata è un progetto di Wikimedia Deutschland, iniziato il 30 ottobre 2012. Lo scopo del progetto è fornire dati che possano essere utilizzati da qualsiasi progetto Wikipedia, incluso Wikipedia. Wikidata non memorizza solo i fatti, ma anche le fonti corrispondenti, in modo che la validità dei fatti possa essere controllata. Etichette, alias e descrizioni delle entità in Wikidata sono fornite in più di 350 lingue. Wikidata è uno sforzo della community, ad esempio, gli utenti aggiungono e modificano in modo collaborativo le informazioni. Inoltre, lo schema viene mantenuto ed esteso sulla base di accordi comunitari. Wikidata è cresciuto anche grazie all'integrazione dei dati di Freebase.
- **YAGO:** YAGO (Yet Another Great Ontology) è stato sviluppato presso l'Istituto Max Planck di Informatica a Saarbrücken dal 2007. YAGO comprende informazioni estratte da Wikipedia (ad esempio, categorie, rettifiche, infobox), WordNet (ad esempio, synset, iponimia) e GeoNames. A partire dal 24 marzo 2015 è disponibile YAGO3.

2.2.4 Incompletezza

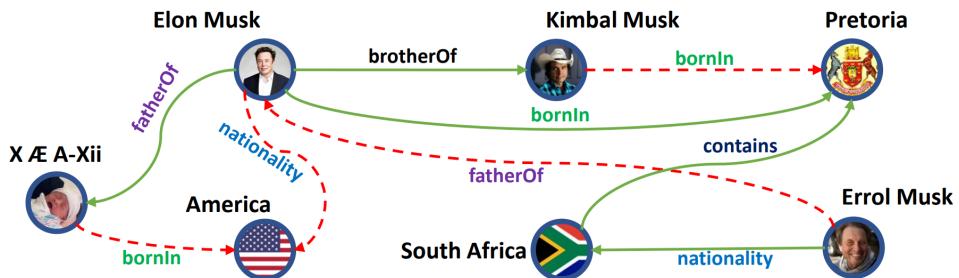


Figura 2.5: **Famiglia Musk.** In questo caso in rosso vi sono alcuni collegamenti mancanti che rappresentano l'incompletezza della conoscenza.

Il principale problema dei KG ad oggi consiste nel fatto che non sono completi, ovvero non tutti gli archi che rappresentano fatti del mondo reale sono presenti nel KG. Questa incompletezza è comprensibile considerando che queste strutture possono essere state costruite sia a mano con tecniche di crowdsourcing (gruppi di persone fisiche che introducono nuovi fatti) che

estraendo fatti con tecniche automatiche (come sarà descritto nel capitolo successivo) oppure anche con una combinazione delle due. Per questo motivo è inevitabile che si perdano molte informazioni. Questo problema è da sempre noto in letteratura, come viene riportato qui [49]:

"For example, 71% of the roughly 3 million people in Freebase have no known place of birth, 94% have no known parents, and 99% have no known ethnicity."

E non è un problema limitato a Freebase, ma vale per tutti i KG principali, ad esempio [41]:

"DBpedia, which is generated from Wikipedia's infoboxes, contains 4.6 million entities, but half of these entities contain less than 5 relationships"

Per questo sono nati diversi approcci di Knowledge Graph Completion o Knowledge Graph Augmentation, che puntano a suggerire quali possano essere i collegamenti mancanti, ovvero i fatti mancanti nel KG. Nelle prossime due sezioni saranno documentati due approcci principali al problema: 1) **Relation Extraction** nel capitolo 3, e **Link Prediction** (capitolo 4).

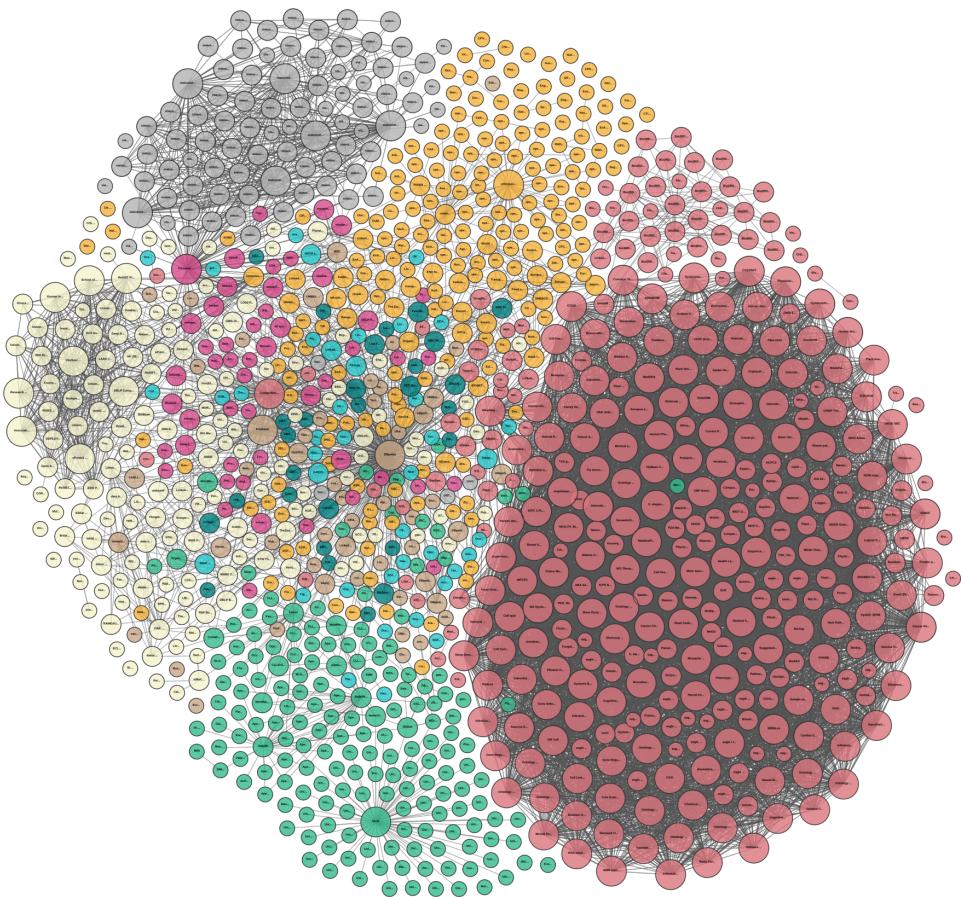


Figura 2.6: **Un KG di grandi dimensioni** Le dimensioni variabili dei nodi rispecchiano la cardinalità di archi entranti/uscenti (degree)

Capitolo 3

Relation Extraction

3.1 Estrazione di pattern

Il processo vero e proprio di estrazione di relazioni si colloca come parte finale in una pipeline più estesa che prevede, nelle prime fasi, di avere a che fare con testo scritto in linguaggio naturale. Questo tipo di pre-processamento, normalmente viene eseguito con tecniche che coinvolgono un intero campo di studi specifico che è il *Natural Language Processing* (NLP).

In questo progetto è stato possibile tralasciare quasi completamente questo aspetto in quanto si è lavorato con dati già pre-processati e si è scelto di concentrarsi sull'obiettivo finale. Nonostante questo in questa sezione sono brevemente descritti gli aspetti fondamentali che caratterizzano le fasi iniziali di qualsiasi sistema completo di relation extraction e che quindi hanno a che fare con un input costituito da nient'altro che testo scritto in linguaggio umano.

Il primo step necessario è comunemente noto come *Named Entity Resolution* (NER) e consiste nel trovare in modo automatico nel testo le entità del mondo reale come mostrato in figura 3.1.

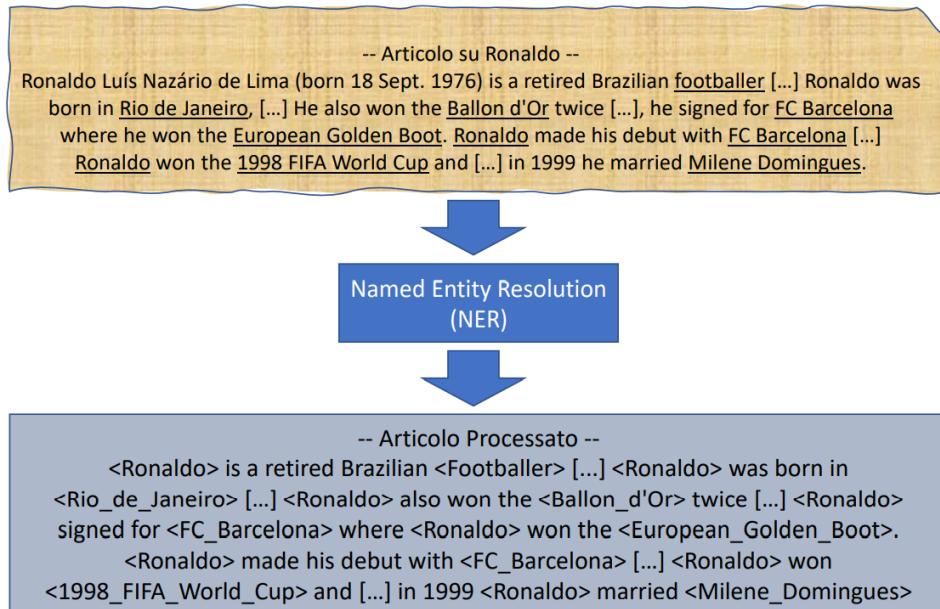


Figura 3.1: **Un esempio di Named Entity Resolution.** Ogni entità conosciuta viene individuata ed annotata, inclusi i pronomi che possono riferirsi ad entità specifiche.

Il secondo step prevede l'estrazione e la manipolazione del testo che riguarda le entità rilevate ed è parte integrante di un modello di relation extraction. Dal momento che il tipo di elaborazione e quindi le *features* testuali estratte possono variare da modello a modello.

3.2 Estrazione di relazioni

Estrarre una relazione consiste nel "tradurre" una frase testuale anche complessa in una etichetta associata al concetto semantico che tale frase esprime. Ad esempio date le seguenti triple:

```

<Ronaldo>, "made his debut with", <FC_Barcelona>
<Barack_Obama>, "met his future wife", <Michelle_Robinson>

```

estrarre relazioni significa essere in grado di dire:

```

"made his debut with" ----> team
"met his future wife" ----> spouse

```

Un compito abbastanza elementare per un essere umano ma non altrettanto per una macchina. In quest'ultimo caso, il task di estrarre relazioni può

essere considerato come un problema di classificazione multipla. Ovvero, l’obiettivo è predire, per ciascuna coppia di entità la relazione in cui queste partecipano, scegliendo la relazione giusta fra diverse relazioni e quindi classi possibili. Così come per qualsiasi task di classificazione, il classificatore necessita di dati di training in cui ciascuna coppia di entità e le relative features, siano etichettate con la relazione corrispondente corretta [44]. Le *features* potrebbero ad esempio essere le frasi indicate nell’esempio sopra fra virgolette o caratteristiche più complesse del contesto in cui vengono trovate le due entità.

Vale la pena soffermarsi sullo scenario generale della Relation Extraction come anche introdotto nell’articolo [4], esiste una grande quantità di testo non strutturato sul Web, contando anche siti web di notizie, blogs, comunicazioni via e-mail, documenti governativi, log di chat, e così via. Come si potrebbe aiutare un essere umano a comprendere e gestire tutti questi dati? Un’idea, abbastanza popolare è convertire tutto questo testo non strutturato in strutturato annotando l’informazione semantica.

Tuttavia il semplice volume e l’eterogeneità dei dati rendono l’annotazione dei dati da parte di esseri umani difficoltosa. Per questo motivo vorremmo invece avere un computer in grado di annotare tutti i dati con la struttura che ci interessa. Di norma, siamo interessati in relazioni fra entità come persone, organizzazioni, e luoghi. Alcuni esempi di relazioni possono descrivere l’affiliazione di una persona a qualcosa o il luogo fisico in cui si trova un’organizzazione.

Gli attuali *named entity recognizer* (NER) allo stato dell’arte, come ad esempio Bikel et al., 1999, e Finkel et al., 2005), possono etichettare automaticamente i dati con elevata accuratezza. Tuttavia l’intero processo di estrazione delle relazioni non è un task banale. Un computer deve sapere come riconoscere un frammento di testo avente una certa proprietà di interesse per effettuare un’annotazione corretta. Perciò estrarre relazioni semantiche fra entità in linguaggio naturale è un passo cruciale verso le possibili applicazioni della comprensione del linguaggio naturale.

Riprendendo quanto detto anche nell’introduzione, tipicamente i classifici survey sulla relation extraction, tendono a dividere gli approcci possibili in due rami principali: 1) Supervisionati e 2) Semi-supervisionati. Tuttavia entrambe le categorie mostrano limitazioni significative nel contesto dell’estrazione di relazioni. Gli approcci supervisionati ([10] [35] [52]) hanno

bisogno di dati di addestramento etichettati, che sono certamente costosi in termini di tempo da produrre. Questa limitazione inoltre, rende gli approcci supervisionati "difficili da estendere", nel senso che volendo rilevare nuovi tipi di relazioni saranno necessari nuovi dati di addestramento. Come se non bastasse, i classificatori supervisionati tendono ad adattarsi troppo al dominio dei testi utilizzati per l'addestramento per poi avere prestazioni sub-ottime quando testati su testi di altri domini.

Gli approcci semi-supervisionati ([2], [9], [16]) tipicamente fanno affidamento sul *bootstrap learning*. In pratica, prima usano un piccolo dataset per apprendere come estrarre relazioni aggiuntive, poi usano le relazioni estratte per l'addestramento, ripetendo questi passi iterativamente. Anche se gli approcci semi-supervisionati riducono significativamente lo sforzo di dover produrre manualmente i dati di addestramento, comunque richiedono un insieme iniziale di istanze etichettate per ogni relazione estratta. Perciò estendere questi approcci per poter estrarre nuove relazioni tipicamente necessita di uno sforzo umano.

Sono state anche proposte in passato tecniche non supervisionate per l'estrazione di relazioni [42] che successivamente si sono evolute nel sottocampo dell'Open Information Extraction ([15], [17], [50]). Sebbene i metodi non supervisionati non richiedano alcun tipo di dato di addestramento, tipicamente riportano risultati sub-ottimi che sono oltretutto difficili da interpretare e mappare su un dato insieme di relazioni, schema o ontologia [18].

Per via di tutti questi svantaggi, nel corso degli ultimi anni, è stato proposto un metodo di addestramento supplementare, chiamato Distant Supervision a cui è dedicata la sezione successiva.

3.3 Distant Supervision

[44] La Supervisione a Distanza o Distant Supervision combina i benefici degli approcci semi-supervisionati e non supervisionati per la relation extraction e sfrutta una knowledge base come sorgente di dati per l'addestramento.

La knowledge base che viene utilizzata tipicamente memorizza i dati in maniera semi-strutturata, utilizzando un serie di coppie chiave-valore per creare degli *statements* ed esprimere l'informazione come triple nella forma: (*soggetto*, *predicato*, *oggetto*), connettendo un dato *soggetto* con un

oggetto utilizzando un *predicato* (relazione), come visto negli esempi precedenti. Il Resource Description Framework (RDF1) è uno standard popolare per esprimere tali triple. Si noti che il soggetto e l'oggetto sono identificati (a piacere) con ID univoci nella knowledge base e che possono essere utilizzati in maniera intercambiabile, nel senso che l'oggetto di una data tripla potrebbe anche essere il soggetto di altre triple.

Un insieme di tali triple costituisce un *multi-grafo orientato* i cui nodi rappresentano i soggetti e gli oggetti delle triple e gli archi etichettati rappresentano i predicati che li connettono.

3.3.1 Etichettatura automatica

Nell'introduzione si è presentata l'idea alla base di Distant Supervision. Come più formalmente proposto da Smirnova in [44], data una collezione di documenti **C** ed una knowledge base **D**, Distant Supervision trova delle corrispondenze dalle relazioni in **D** alle frasi in **C**. Più specificamente, l'idea è prima collezionare tutte quelle frasi del corpus **C** che menzionano la coppia di entità (e_1, e_2) , tali che sia e_1 che e_2 esistano nella knowledge base **D**. In seguito se una frase menziona (e_1, e_2) ed esiste una tripla (e_1, r, e_2) , nella knowledge base, allora l'approccio distant supervision etichetta questa frase come un'istanza (anche detta menzione) della relazione r . Ad esempio la frase:

"South African entrepreneur **Elon Musk** is known for founding **Tesla Motors** and SpaceX."

Menziona la coppia (*Elon Musk*, *Tesla Motors*). Assumendo che nella knowledge base esista la tripla (*Elon Musk*, *created*, *Tesla Motors*) la frase testuale verrebbe etichettata con la relazione *created* e poi usata come dato di addestramento. Si noti che è puramente una scelta progettuale quella di decidere se e come processare il testo, ad esempio scegliendo di prelevare soltanto il testo compreso tra le due entità, in questo caso: "is known for founding" ---> *created*.

3.3.2 Vantaggi e Svantaggi

Alla luce di questi ulteriori dettagli possiamo analizzare nuovamente i problemi di distant supervision. Riproponendo la figura 1.1 vista nell'introduzione. Questo tipo di problema è già noto in letteratura, infatti viene

riportato anche in [44], gli approcci supervisionati sono specifici su un certo dominio verticale e spesso soffrono di overfitting (dal momento che i dati di addestramento sono tipicamente pochi e troppo specifici). Distant supervision invece, è applicabile a corpora grandi ed eterogenei visto che sfrutta una conoscenza su larga scala che definisce informazione semi-strutturata per una grande varietà di entità.

Sfortunatamente, i dati etichettati in maniera automatica spesso comportano *rumore*, cioè etichette mancanti (falsi negativi) ed etichette errate (falsi positivi). Da una parte, i falsi negativi sono perlopiù causati dall'incompletezza della knowledge base. Due entità possono anche essere in relazione fra loro nel mondo reale ma la loro relazione potrebbe mancare nella knowledge base, e quindi la loro menzione sarebbe erroneamente etichettata come esempio negativo da distant supervision. Dall'altra, due entità in relazione fra loro potrebbero apparire in una frase che tecnicamente non esprime la loro relazione. In tal caso si potrebbe generare un falso positivo usando distant supervision. Per mostrare ques'ultimo caso consideriamo i due esempi seguenti:

Elon Musk is the co-founder, CEO, and Product Architect at **Tesla Motors**.

Elon Musk says he is able to work up to 100 hours per week running **Tesla Motors**.

Entrambe gli esempi menzionano *Elon Musk* e *Tesla Motors* ma soltanto il primo esprime davvero la relazione *co-founderOf*. Comunque, data una knowledge base grande e curata, è molto più probabile che vengano generati falsi positivi che falsi negativi, ovvero le probabilità che due entità senza relazione nella knowledge base siano in relazione nel mondo reale sono tipicamente basse.

Possono essere utilizzate una grande varietà di tecniche per ridurre questo rumore che si genera via Distant Supervision, tuttavia la trattazione di questo argomento va oltre lo scopo di questa introduzione, sarà quindi presentato un semplice modello di relation extraction utilizzato nel corso del progetto, illustrando anche le metodologie applicate per affrontare il problema del rumore.

3.4 Caso d'uso: Lector

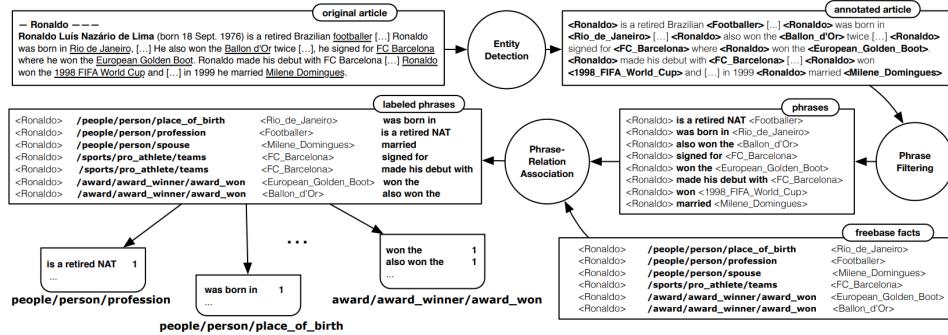


Figura 3.2: **Lector pipeline.** Processo di etichettatura fatti di Lector[11]

Lector [11] è un semplice modello di relation extraction che sfrutta Distant Supervision per etichettare le menzioni di determinate coppie di entità, è stato scelto per questo progetto 1) per la semplicità di realizzazione, 2) per il tipo di etichettatura che utilizza. In Lector infatti dalle frasi testuali così come sono state presentate negli esempi precedenti, viene estratta un vettore di feature composto da il testo compreso fra le due entità ed i tipi di DB-Pedia delle due entità. Questo comporta il vantaggio di poter direttamente visionare questo vettore di feature al quale ci riferiremo come *pattern*. 3) Inoltre la disponibilità di pattern già estratti in precedenza ha permesso di saltare tutta la parte di pre-processamento descritta nella prima sezione di questo capitolo.

Nella prima fase, un training set viene generato da Lector via Distant Supervision, in questo caso per cercare di risolvere il problema del rumore descritto precedentemente Lector etichetta tutte le frasi delle coppie di entità nella knowledge base come esempi negativi o "*unknown*" e le restanti con la relazione presente. Successivamente, i pattern vengono associati alle relazioni in modo puramente statistico, i.e. se un determinato pattern si ripete più volte ed il maggior numero di volte viene associato ad una determinata relazione, Lector conclude che quel pattern esprime quella relazione. Questa procedura è mostrata in figura 3.2

Successivamente le associazioni `pattern -> relazione` vengono usate per estrarre fatti dal testo. Di seguito viene riportata un esempio semplice del funzionamento di tutta la pipeline.

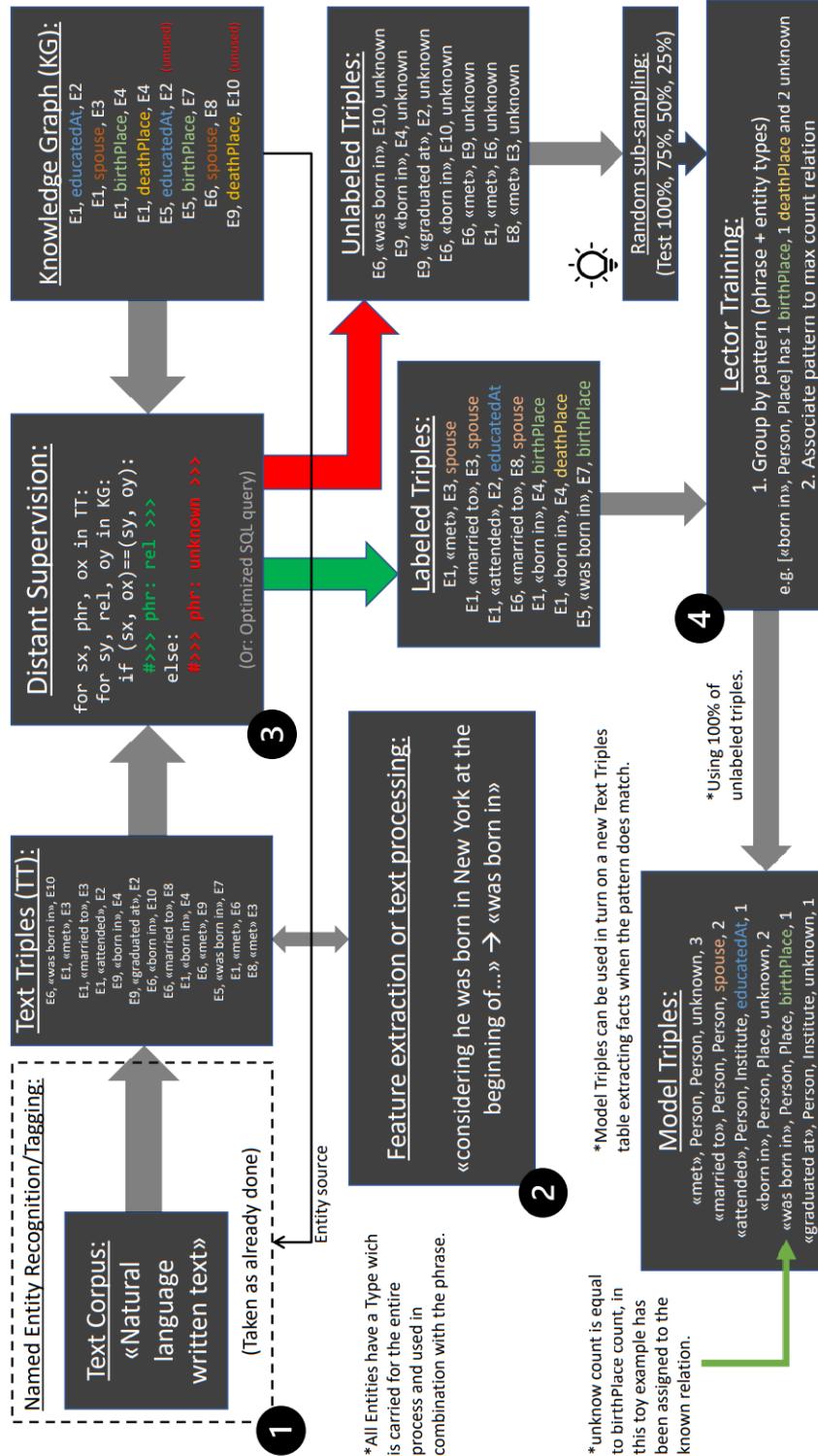


Figura 3.3: Esempio del funzionamento di Lector. All'interno del riquadro tratteggiato è indicata la parte che non è stata implementata per questo progetto.

Capitolo 4

Link Prediction

4.1 Deduzione automatica

A partire dai primi anni di sviluppo dell'intelligenza artificiale c'è sempre stato un notevole interesse nella possibilità di riuscire a fornire alle macchine la capacità di imitare i meccanismi del ragionamento umano.

Link Prediction (LP) è un particolare campo dell'IA che si occupa di dedurre o inferire collegamenti, appunto *links*, all'interno di strutture di dati organizzate a grafo, descritte nel capitolo 2 di questa tesi. Come area di ricerca, sempre più attiva nel tempo, ha recentemente beneficiato dell'improvviso e forte successo dell'*machine learning* e delle tecniche di *deep learning*. La stragrande maggioranza di modelli di link prediction utilizzano gli elementi costitutivi di un KG per apprendere delle rappresentazioni vettoriali a bassa dimensionalità chiamate *Knowledge Graph Embeddings*, per poi impiegarle nell'inferenza di nuovi fatti. Ispirati ad alcuni lavori iniziali come RESCAL [36] e TransE [7], nel breve arco di pochi anni i ricercatori hanno sviluppato dozzine di nuovi modelli basati su architetture molto diverse fra loro. Un aspetto comune che emerge nella maggior parte della

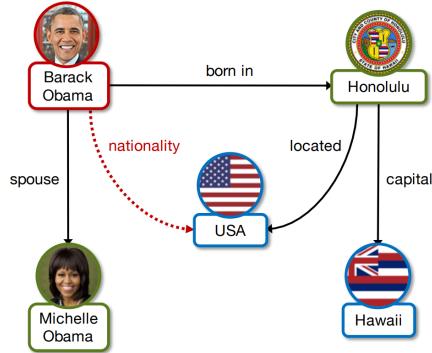


Figura 4.1: **Inferire un link.** Un semplice esempio concettuale di link prediction, la *nationality* di Barack Obama può essere inferita sapendo che Obama è nato ad Honolulu e che Honolulu si trova negli U.S.A.

documentazione in quest'area di ricerca, anche abbastanza problematico, è che vengono riportati dei risultati aggregati su un grande numero di fatti di test in cui poche entità sono sovrarappresentate. Come mostrato dai risultati, i sistemi di link prediction esibiscono buone performance su questi *benchmark* considerando però solo tali entità ed ignorando le altre. Inoltre, le limitazioni delle attuali *best-practice*¹ possono rendere di difficile comprensione come ciascun lavoro di ricerca si pone rispetto ad un altro e quindi riuscire ad individuare quale direzione di ricerca valga la pena seguire. In aggiunta, i punti di forza, le debolezze e le limitazioni delle tecniche correnti sono ancora sconosciute, ovvero, le circostanze che permettono a questi modelli di funzionare meglio o peggio sono state profondamente analizzate. In parole povere, ancora non sappiamo realmente cosa renda un fatto facile o difficile da inferire [40].

4.2 Breve storia

Link prediction ha attirato l'attenzione di diverse community di ricerca che vanno dalla statistica alla network science fino al machine learning ed il data mining. In statistica, modelli per la generazione casuale di grafi come lo *stochastic block model*, propongono un approccio per generare nodi in un grafo casuale. Per quanto riguarda i social networks, Liben-Nowell e Kleinberg hanno proposto un modello di link prediction basato su differenti metriche di prossimità in un grafo [29]. Sono stati proposti diversi modelli statistici per link prediction dalla community del machine lerarning e del data mining, per esempio Popescul et al. hanno proposto un modello strutturato con regressione logistica che può utilizzare features relazionali [38]. Modelli probabilistici condizionati alla località, basati su attributi e features strutturali sono stati proposti da O'Madadhain et al [37]. Diversi modelli basati sul modello di grafo orientato per collective link prediction sono stati proposti da Getoor. Sono stati proposti anche altri approcci, basati su esplorazione casuale del grafo [5] e sulla fattorizzazione di matrici [31].

Tendenzialmente Link Prediction è stata eseguita attraverso approcci di Statistical Relational Learning tradizionali, come rule mining o path ranking algorithm.

¹Un modo standard di procedere suggerito dalla comunità

Dopo l'avvento del deep learning, ad oggi, l'alternativa più popolare invece è usare tecniche basate su embedding appresi con *machine learning* [51].

4.3 Funzionamento Generale

[40] Questa sezione fornisce uno schema del task di link prediction nel contesto dei knowledge graphs, introducendo i concetti chiave. Si definisce un KG come un multi-grafo etichettato orientato = $(\mathbf{E}, \mathbf{R}, \mathbf{G})$:

- \mathbf{E} : un insieme di nodi che rappresenta le *entità*;
- \mathbf{R} : un insieme di etichette rappresentanti le *relazioni*;
- $\mathbf{G} \subseteq \mathbf{E} \times \mathbf{R} \times \mathbf{E}$: un insieme de archi rappresentanti i *fatti* che connettono coppie di entità. Ciascun fatto è una tripla $\langle h, r, t \rangle$ in cui h è la *testa* (head), r è la *relazione* (relation), e t è la *coda* (tail) del fatto.

Link prediction ha l'obiettivo di sfruttare i fatti esistenti in un KG per dedurre quelli mancanti. Ciò può essere definito, nell'ambito della notazione esposta precedentemente come indovinare l'entità corretta che completa $\langle h, r, ? \rangle$ (predizione della coda o tail prediction) oppure $\langle ?, r, t \rangle$ (predizione della testa o head prediction). Per semplicità quando ci riferiamo globalmente a predizione di testa o di coda, chiamiamo entità *sorgente* l'entità che è nota prima della predizione, ed entità *obiettivo* quella che deve essere predetta.

Nel corso del tempo, numerosi approcci sono stati proposti per affrontare il task di link prediction. Alcuni metodi sono basati su features osservabili ed impiegano tecniche quali il Rule Mining [21] [20] [30] [3] od il Path Ranking Algorithm [27] [28] per identificare le triple mancanti nel grafo. Recentemente con l'avvento delle nuove tecniche di Machine Learning, i ricercatori hanno cercato di catturare le caratteristiche latenti dei grafi con rappresentazioni vettorizzate, o *embeddings*, delle loro componenti elementari.

In generale, gli *embeddings* sono vettori di valori numerici che possono essere usati per rappresentare qualsiasi tipo di elemento (e.g. a seconda del dominio: parole, persone, prodotti, ecc.). Gli *embeddings* vengono appresi automaticamente, basandosi su come gli elementi corrispondenti appaiono ed interagiscono fra loro nei datasets rappresentativi del mondo reale.

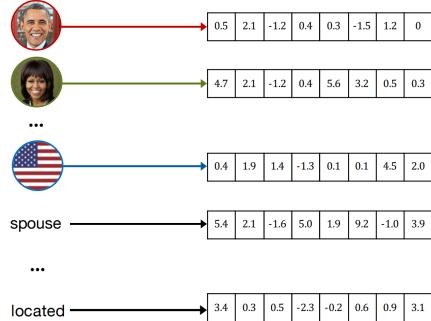


Figura 4.2: **Knowledge Graph Embeddings.** Ad ogni nodo del grafo, associato ad una entità, viene associato un vettore numerico.

Per esempio i *word embeddings* sono divenuti un modo standard di rappresentare le parole in un vocabolario, e generalmente vengono appresi utilizzando corpora testuali come input. Nel caso dei Knowledge Graphs, gli embeddings sono tipicamente usati per rappresentare entità e relazioni usando la struttura a grafo; i vettori risultanti, chiamati **Knowledge Graph Embeddings** (KGE), racchiudono la semantica del grafo originale e possono essere usati per identificare i nuovi collegamenti all'interno di esso, affrontando il problema posto da link prediction.

Successivamente si userà il *corsivo* per identificare gli elementi del knowledge graph (entità o relazioni), ed il **grassetto** per riferirsi ai corrispettivi embeddings.

4.3.1 Funzione di score

[40] La maggior parte dei modelli di link prediction basati su embeddings definiscono una *scoring function* ϕ per stimare la plausibilità di ogni fatto $\langle h, r, t \rangle$ usando i suoi embeddings:

$$\phi(\mathbf{h}, \mathbf{r}, \mathbf{t})$$

Di norma si assume che maggiore sia il valore che assume tale funzione ϕ , maggiore sarà la plausibilità del fatto. In parole poche è una funzione che prende gli embedding di *head*, *relation*, *tail* e calcola la plausibilità. Può prevedere o meno la presenza di pesi condivisi (ad esempio layer di reti neurali) da ottimizzare insieme agli embeddings. Uno dei primi modelli, pionieristico, è stato TransE che interpretava le relazioni come traslazioni in spazio latente; da lì poi cercando di superare i suoi limiti sono state sviluppate varie famiglie che usano gli approcci più disparati per calcolare lo score. Nello schema in figura 4.3 viene riassunta la tassonomia dei modelli di Link Prediction in ordine cronologico.

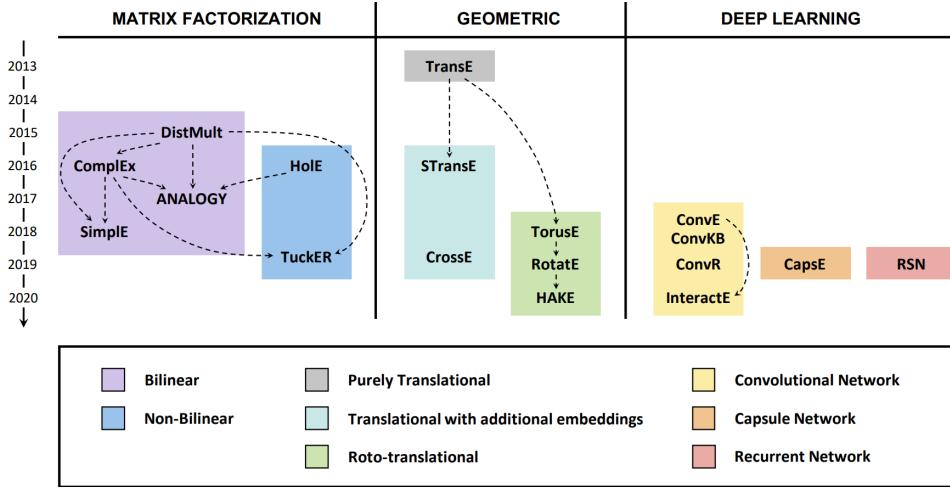


Figura 4.3: **Tassonomia dei modelli di LP.** [40] Le frecce tratteggiate indicano che il metodo puntato è costruito sul metodo da cui parte la freccia, o generalizzando o specializzando la definizione della funzione di scoring.

Uno dei primi modelli di Link Prediction basati su embeddings è stato TransE, sviluppato da Bordes et al. nel 2013 [7]. Ispirandosi alle proprietà traslazionali osservate sui word embeddings appresi da Word2Vec [32], TransE interpreta esplicitamente le relazioni come traslazioni nello spazio latente. A partire dall'embedding della head, la relazione dovrebbe applicare uno spostamento tale da condurre in prossimità dell'embedding dell'entità tail: $\phi(h, r, t) = |h + r - t|$. A partire da questa semplice formulazione, e cercando di superarne le limitazioni, col tempo sono state proposte decine di modelli che usano gli approcci più disparati nelle rispettive funzioni di score.

Recentemente il survey [40] ha cercato di interpretare e strutturare la ricerca condotta finora, organizzando questi modelli nella tassonomia riportata in figura 4.3. I modelli sono organizzati in tre famiglie principali: basati su fattorizzazione di matrici, su operazioni geometriche e su uso di layer di deep learning. Ciascuna delle famiglie può prevedere a sua volta delle sotto-famiglie: ad esempio, le operazioni geometriche più popolari nella letteratura di Link Prediction hanno una suddivisione netta tra traslazionali e rotazionali.

4.3.2 Addestramento

[40] I dataset impiegati nella ricerca su link prediction, sono tipicamente ottenuti sottocampionando knowledge graphs di reale utilizzo. Ciascun dataset può perciò essere visto come un piccolo knowledge graph con il suo set di entità **E**, relazioni **R** e fatti **G**. Per poter facilitare l’addestramento **G** viene a sua volta suddiviso in tre sottoinsiemi disgiunti: un training set **G_{train}**, un validation set **G_{valid}**, ed un test set **G_{test}**.

Durante l’addestramento, gli embeddings (ed eventuali pesi condivisi) sono generalmente inizializzati in maniera casuale e successivamente si cerca di massimizzare la plausibilità dei fatti e quindi la funzione di *score* sopracitata, mediante algoritmi per l’ottimizzazione come ad esempio back-propagation con gradient descent. Gli esempi positivi contenuti in **G_{train}** spesso vengono *corrotti* a caso per poter generare esempi negativi, nel corso del tempo sono stati proposti metodi sempre più efficaci per generare esempi negativi, ad esempio, campionandoli da una distribuzione di Bernoulli [48] o generarli con algoritmi avversari [46]. Oltre agli embeddings degli elementi del KG, i modelli possono anche usare gli stessi algoritmi di ottimizzazione per apprendere parametri addizionali (e.g. i pesi di un layer di una rete neurale). Tali parametri, se presenti, sono utilizzati nella funzione di scoring ϕ per calcolare gli embeddings veri e propri delle entità e delle relazioni. Dal momento che questi parametri addizionali non sono associati a nessun elemento del KG in particolare, spesso sono chiamati *parametri condivisi*.

Tutto il processo di ottimizzazione ha come obiettivo la massimizzazione della plausibilità di fatti positivi e contemporaneamente minimizzare quella dei fatti negativi. Questo spesso corrisponde ad utilizzare una **triplet loss** function che può essere di varia natura (e.g. Negative Log Likelihood, Binary Cross-Entropy, margin-based, pairwise ranking loss, ecc.). Se il sistema riesce a generalizzare darà plausibilità buone anche a fatti veri non visti durante la fase di addestramento, contenuti in **G_{train}**.

4.3.3 Valutazione

[40] La valutazione delle prestazioni di un modello di link prediction si ottiene eseguendo sia la predizione della *head* che della *tail* su tutte le triple di test in **G_{test}**, e calcolando per ogni predizione il punteggio ottenuto delle entità obiettivo, confrontandolo con tutte le altre. Utilizzando i punteggi che non sono altro che il risultato della funzione di *scoring* applicata a cia-

scuna tripla, è possibile ottenere una classifica dei fatti più plausibili. La posizione ottenuta da un certo fatto in tale classifica è detta *rank* del fatto predetto. Idealmente, l'entità obiettivo corretta dovrebbe riportare la più alta plausibilità.

I *ranks* possono essere calcolati secondo due scenari abbastanza differenti, chiamati *raw* e *filtered*. Difatti, una predizione potrebbe avere più risposte valide: per esempio, quando viene predetta la tail per *{Barack Obama, parent, Natasha Obama}*, un modello potrebbe associare un punteggio più alto a Malia Obama che a Natasha Obama. Più genericamente, se il fatto predetto è contenuto in \mathbf{G} (cioè si trova o in \mathbf{G}_{train} o in \mathbf{G}_{valid} o in \mathbf{G}_{test}), la predizione viene considerata corretta. A seconda di quando una predizione vada considerata corretta oppure no, sono stati ideate due modalità differenti:

- *Scenario Raw*: tutte le entità predette correttamente che superano in punteggio l'entità obiettivo, sono considerate errori, perciò contribuiscono al calcolo del *rank*. In altri termini, le risposte "valide" che superano in score l'entità obiettivo sono considerate errate.
- *Scenario Filtered*: tutte le entità predette correttamente che superano in punteggio l'entità obiettivo non vengono considerate come errori, perciò vengono saltate nel calcolo del *rank*. In altri termini, le risposte "valide" che superano in score l'entità obiettivo sono ignorate.

Inoltre per calcolare il *rank* è anche necessario definire una politica da applicare nel caso in cui più entità ottengano lo stesso score dell'entità obiettivo. In questo caso si parla di *pareggio* e può essere gestito in diversi modi con effetti differenti, ad esempio il *rank* dell'entità target potrebbe essere scelto a caso fra tutte le entità che sono alla pari.

I *ranks* Q ottenuti dall'previsioni del test set sono generalmente utilizzati per calcolare delle metriche globali standard. Le metriche più comunemente usate in link prediction sono: Il *Mean Rank* (MR) che è la media dei *ranks* ottenuti:

$$MR = \frac{1}{|Q|} \sum_{q \in Q} q \quad (4.1)$$

Sempre compresa fra 1 e $|\mathbf{E}|$, più è bassa e migliore è il risultato ed è molto sensibile agli outliers, perciò i ricercatori hanno iniziato ad evitare questa metrica, affidandosi invece al *Mean Reciprocal Rank (MRR)*:

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{q} \quad (4.2)$$

Che corrisponde alla media delle inverse dei *ranks* ottenuti, è sempre compreso tra 0 ed 1 e più alto è il valore, migliori saranno i risultati del modello. *Hits@K (H@K)* è il rateo di predizioni per le quali il *rank* è minore o uguale ad una certa soglia K :

$$H@K = \frac{|\{q \in Q : q \leq K\}|}{|Q|} \quad (4.3)$$

Valori comunemente usati per K sono, 1, 3, 5, 10. Maggiore è il valore $H@K$, migliore è il modello. In particolare, quando $K = 1$, viene misurato il rateo con cui nei fatti di test la target entity viene predetta correttamente al primo tentativo.

Queste metriche possono essere calcolate sia separatamente per sottinsiemi di predizioni, ad esempio se si vogliono considerare le predizioni di testa e di coda separatamente, oppure considerando tutte le predizioni di test insieme.

4.3.4 ComplEx

In questo progetto è stato usato il modello ComplEx [47], che si è dimostrato essere il più performante come anche riportato nel survey [40]. Come la gran parte degli altri modelli tenta di effettuare la link prediction attraverso la fattorizzazione latente che significa in pratica calcolare gli *embeddings* rappresentativi delle componenti del grafo, con una particolarità: utilizzare numeri complessi per ciascuna componente. La composizione di *embeddings* complessi può gestire una grande varietà di relazioni binarie, fra cui le relazioni simmetriche ed anti-simmetriche. Confrontato con modelli allo stato dell'arte come Neural Tensor Network ed Holographic Embeddings, questo approccio basato su *embeddings complessi* è senza dubbio più *semplice* dal momento che utilizza solo il prodotto scalare Hermitiano, la controparte complessa del prodotto scalare tra vettori reali. Questo approccio è scalabile su datasets di grandi dimensioni, mantenendo un costo lineare nel tempo e nello spazio, superando in prestazioni approcci alternativi nei benchmark

standard di link prediction. Come verrà esposto nel capitolo successivo, sono state effettuate alcune piccole modifiche per adattare il modello al contesto, pur mantenendo delle buone prestazioni: 1) Il problema di predizione non è più formulato come entità target ma come relation target: $\langle h, ?, t \rangle$, 2) La valutazione del *ranking* avviene in uno scenario *raw* e 3) viene applicata una policy personalizzata composta da normalizzazione + soglia. Questi aspetti saranno ripresi nel capitolo successivo.

Capitolo 5

Deflector

5.1 Scartare i pattern errati

Come descritto nei capitoli precedenti, in questo progetto viene esplorato un particolare approccio per risolvere il problema relativo a *patterns* (cioè feature vectors) erroneamente associati a relazioni fortemente correlate ad essi nella fase di generazione del training set.

In passato, questo problema è già stato affrontato in Lector basandosi sui patterns non etichettabili, cioè tutti quei patterns rilevati fra coppie di entità non presenti nella knowledge base. In tal caso, nella fase di Distant Supervision, questi pattern vengono etichettati con la relazione *unknown* ed utilizzati normalmente per associare ad un pattern una relazione. Dal momento che il numero di pattern estratti dal testo durante la *Named Entity Resolution* è molto elevato, vi sarà anche un gran numero di coppie non etichettate, da Lector chiamate *unlabeled triples*. Considerando tutte queste *unlabeled triples* nel conteggio, statisticamente si ottengono delle associazioni *pattern-relazione* molto precise tuttavia scartando alcuni pattern più complessi. In seguito, sottocampionando casualmente l'insieme di *unlabeled triples* con tagli differenti (eg. 25% 50%, 75% del totale) è stato possibile regolare la quantità di pattern scartati manualmente, adattandosi ai risultati ottenuti. In realtà questo tipo di approccio non risolve completamente il problema dal momento che gran parte delle *unlabeled triples* rappresentano informazione persa e spesso trovare un giusto *tradeoff* può dipendere dal domino testuale su cui si opera.

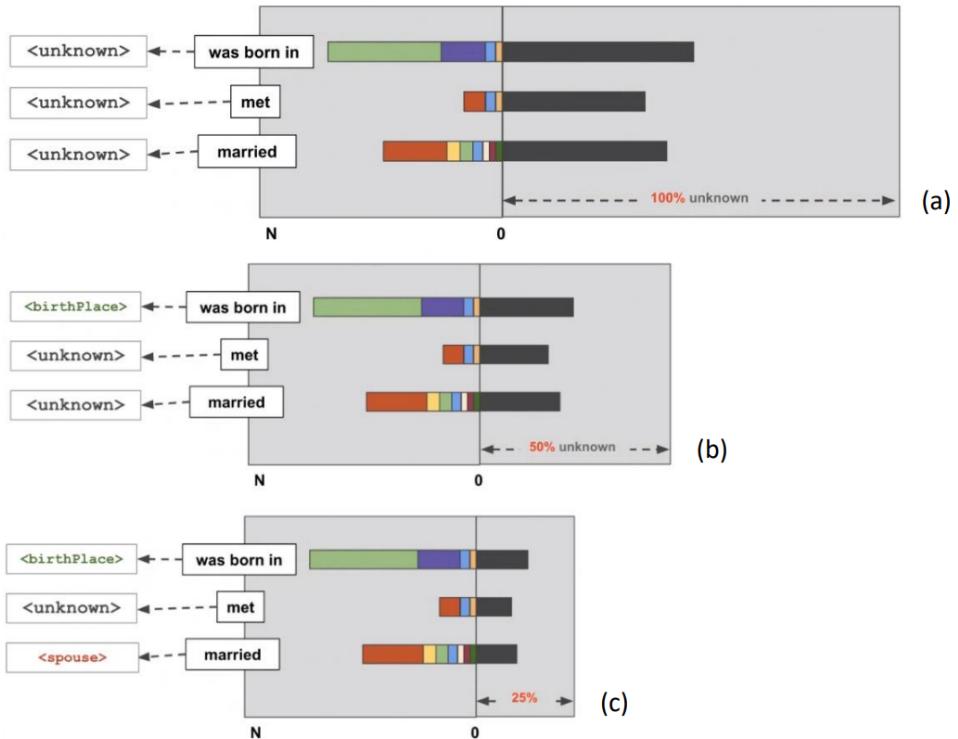


Figura 5.1: Lector idea. Le barre in grigio rappresentano tutti i pattern di coppie che non è stato possibile etichettare con una relazione, le barre colorate rappresentano i pattern etichettati con una certa relazione. Sono mostrati gli effetti di diversi sottocampionamenti e come questo influisca sull'apprendimento dei pattern.

L'approccio di Deflector, cerca di essere completamente indipendente dal modello di relation extraction impiegato. In breve, consiste nel tracciare le coppie di entità che vengono estratte da un determinato pattern, a sua volta associato ad una relazione e successivamente *verificare* il pattern sfruttando Link Prediction per valutare ogni coppia di entità ed aggregare i risultati, come schematizzato in figura 5.2.

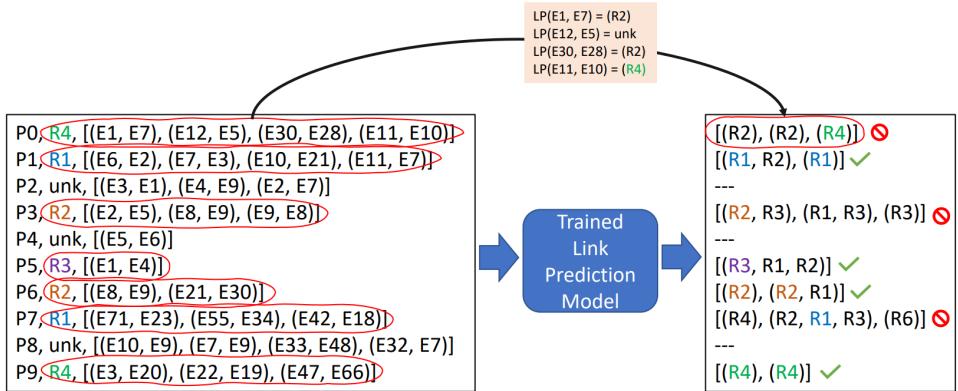


Figura 5.2: **Deflector idea** Nel riquadro a sinistra, PX rappresenta il pattern, RX la relazione associata al pattern, e la lista di coppie sono le istanze della relazione RX estratte perché trovate separate dal pattern PX. Nel riquadro a destra, link prediction ha espresso una predizione per ogni coppia ed è stata verificata l'accuratezza del pattern in base a quante relazioni predette coincidono con quella associata.

5.2 Architettura proposta

Con l'obiettivo di poter applicare l'idea precedentemente illustrata, a prescindere da quale sia il modello di Relation Extraction utilizzato, viene presentato **Deflector**, che sostanzialmente è un *meta-modello* di Relation Extraction, ovvero un modello composto da modelli che implementa la logica necessaria a filtrare i pattern errati e consente anche l'estrazione di fatti mediante due modalità di funzionamento. L'immagine 5.3 mostra l'architettura concettuale. Attraverso due wrapper, che è necessario definire secondo determinate specifiche, si possono fornire a Deflector due modelli pre-addestrati: uno di Relation Extraction ed uno di Link Prediction. Deflector è in grado di estrarre relazioni ereditando ogni caratteristica del modello di Relation Extaction con il quale viene inizializzato. La struttura dati **pattern blacklist** agisce da filtro: ogni volta che un pattern contribuisce all'estrazione di un fatto, se quel pattern è presente nella *blacklist* il fatto non viene estratto. Inizialmente questa struttura è vuota e pertanto i fatti che saranno estratti saranno gli stessi del modello di Relation Extraction utilizzato.

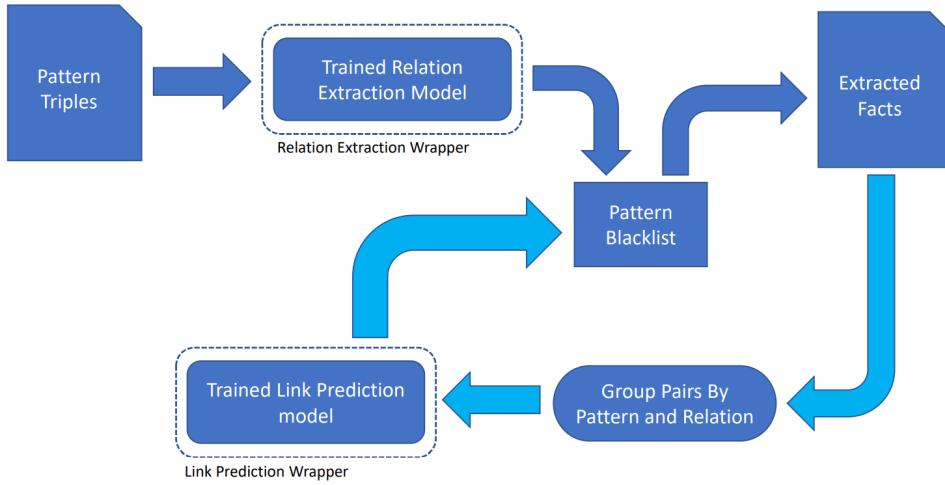


Figura 5.3: Architettura di base Questo schema mostra i concetti principali del metamodello proposto. La struttura più importante è pattern blacklist che conterrà i pattern considerati non buoni per estrarre relazioni.

5.3 Scoprire nuovi pattern

Scartare i pattern considerati errati, certamente può portare ad un forte incremento di *precision* ma abbastanza basso di *recall*. Per questo motivo si è tentato di rilevare eventuali nuovi pattern da quelli associati a relazioni sconosciute, cioè pattern che il modello originale di Relation Extraction non ha associato a nessuna relazione e quindi non può estrarre alcun fatto. Per quest'obiettivo, chiamato *Pattern Discovery*, si cerca di sfruttare lo stesso modello di Link Prediction per valutare tutti i pattern associati a relazioni *unknown*. Pertanto l'architettura precedente subisce una modifica che è presentata nella figura 5.4

5.4 Implementazione

Questa sezione descrive alcuni aspetti tecnici dello sviluppo di *Deflector* oltre ai problemi che sono stati affrontati. Saranno presenti anche delle porzioni del codice effettivo per mostrare in pratica quello che viene descritto. L'intero progetto è stato sviluppato in linguaggio Python¹ che consente una notevole velocità di scrittura del codice. L'intero progetto ha previsto anche

¹<https://www.python.org>

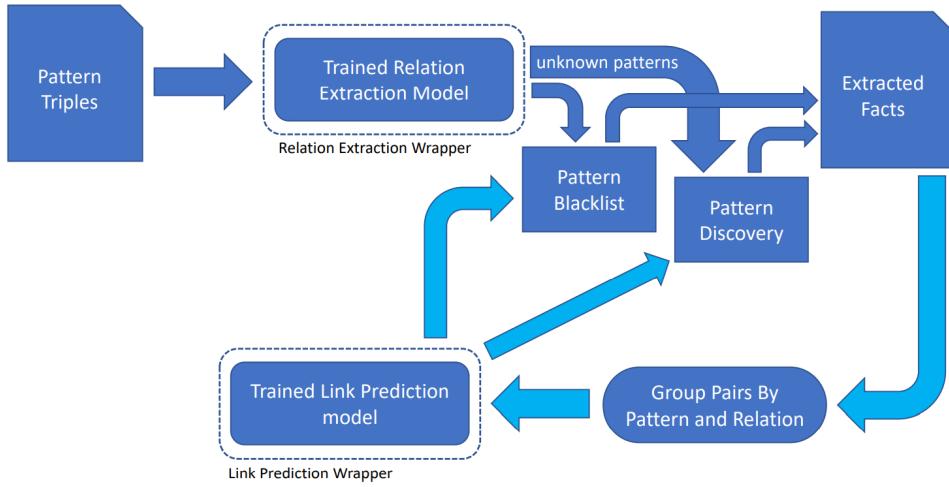


Figura 5.4: **Architettura V2** Lo schema mostra l'effetto delle modifiche al modello base per cercare di scoprire nuovi pattern.

l'utilizzo di librerie aggiuntive (Spacy²) per il *text processing* per esplorare in maniera rapida l'effetto dell'applicazione di *placeholders* nelle frasi dei pattern.

Gli elementi fondamentali di di Deflector sono:

- I wrappers dei modelli;
- Il tracciamento dei pattern in fase di estrazione delle relazioni;
- Il processamento in batch delle coppie di entità con link prediction (ComplEx);
- La valutazione di ciascun pattern in base alle coppie di entità prodotte.

5.4.1 Wrappers

Sono funzioni definite in Python che devono rispondere a specifici standard di input ed output. Il nome deriva dal fatto che "incartano" il modello originale permettendo a Deflector di interagire con degli output sempre nello stesso formato. Un esempio pratico è il wrapper scritto per ComplEx, per default questo modello espone una funzione `all_scores_relations` che prende come input una lista di triple di ID associati a relazioni ed entità del modello con cui è stato addestrato il modello. Viene riportato il codice

²<https://spacy.io>

dell’intero wrapper. Il compito di questa funzione è quello di ricevere una lista di coppie (pairs), cioè istanze e restituire una lista di `set()` ciascuno contenente le relazioni predette per ogni coppia. Vale la pena notare che ComplEx non gestisce istanze che non ”conosce” i.e. passare un’istanza di cui non esiste il rispettivo ID causerebbe un errore. Questo problema viene gestito nel wrapper offrendo il comportamento desiderato: anche passando istanze non note a ComplEx viene comunque restituita una lista di predizioni nel peggiore dei casi contenenti la relazione ”unknown”. Min Max scaler per normalizzare i pattern che applica la formula 5.1 su ciascuna riga della matrice di output contenente gli score per ogni relazione.

$$\mathbf{x}' = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \quad (5.1)$$

Quello che segue è il codice reale del wrapper nonostante l’apparente complessità, consente di delegare tutta la logica di formattazione di input ed output all’esterno di Deflector affidandosi alle capacità del programmatore.

```

# The ComplEx model wrapper
def complex_wrapper(list_of_pairs):

    # Structures to handle possible unknown entities
    pair2complex = dict()
    pair2unknown = dict()
    for pair in list_of_pairs:
        try:
            h_id = DATASET.get_id_for_entity_name(pair[0])
            r_id = 0 # Relation id will be ignored in predictions
            t_id = DATASET.get_id_for_entity_name(pair[1])
            pair2complex[pair] = (h_id, r_id, t_id)
        except:
            pair2unknown[pair] = set(['unknown'])

    # Build ComplEx input
    complex_input = list(pair2complex.values())

    # Compute all scores (output is a 3D tensor with dim (npairs, nrelations, 1))
    all_scores = COMPLEX.all_scores_relations(complex_input)[:, :, 0]

    # Normalize scores to range [0, 1]
    minmax_scale(all_scores, axis=1, copy=False)

    # Array 2d con le posizioni [[riga, colonna]] dei valori che superano la soglia
    best_rels = np.argwhere(all_scores > 0.9)

    # Build wrapper output
    complex_output = [set() for _ in complex_input]
    for row in best_rels:
        rel = DATASET.get_name_for_relation_id(row[1])
        # Exclude inverse relations
        if 'INVERSE' not in rel:
            complex_output[row[0]].add(rel)

    # Re-associate pairs to complex outputs
    pair2complex = dict(zip(pair2complex.keys(), complex_output))

    # Build wrapper output
    output = list()
    for pair in list_of_pairs:
        try:
            output.append(pair2complex[pair])
        except:
            output.append(pair2unknown[pair])

    # Add unknown prediction to empty sets
    for pred in output:
        if not pred:
            pred.add('unknown')

    return output

```

5.4.2 Tracciamento pattern

Supponiamo che il modello di Relation Extraction, già addestrato, abbia associato (erroneamente) al pattern "met" la relazione "spouse" ed estragga queste relazioni:

```
(Barack "met" Michelle) -> Barack spouse Michelle  
(Barack "met" Trump) -> Barack spouse Trump  
(Michelle "met" Trump) -> Michelle spouse Trump
```

Allora è possibile raggruppare le estrazioni in per pattern in questo modo:

```
{"met", spouse, [(Barack, Michelle), (Barack, Trump), (Michelle, Trump)]}
```

Proprio come in questo esempio, il wrapper del modello di Relation Extraction richiede in input lo stesso input del modello di RE ma produce come output il pattern + il fatto. Successivamente entra in gioco il modello di Link Prediction a cui viene "chiesto" di inferire per ciascuna di queste coppie quali siano i link maggiormente plausibili.

5.4.3 Batch processing

Dal momento che ciascuna coppia di entità estratta da il modello di Relation Extraction andrebbe valutata da Link Prediction, uno dei problemi principali affrontati è stato dover processare una grande quantità di coppie raggruppate per pattern comune, in gruppi relativamente piccoli. Quest'ultimi se processati singolarmente richiederebbero un tempo eccessivo di esecuzione principalmente dovuto al fatto che sono richiesti continui accessi ed operazioni in memoria centrale. Si consideri inoltre che, con riferimento alla figura 5.2, per ogni pattern una coppia di entità può comparire in più estrazioni creando una certa ridondanza. Per questi motivi si è scelto di procedere in questo modo:

- Creare un indice in cui le chiavi sono coppie di entità: (head, tail)
- Effettuare le predizioni con Link Prediction in batch associando ad ogni coppia di entità il rispettivo output di Link Prediction (wrapper)

In questo modo, non vengono predette più volte le stesse istanze e si può regolare a piacere la dimensione dei batch da passare a Link Prediction, ottenendo una struttura intermedia con tutte le predizioni pronte all'uso con accesso diretto.

5.4.4 Valutazione pattern

Di seguito viene mostrato il codice che valuta i pattern, per ciascun gruppo composto da (pattern, relazione, [lista di istanze]) viene presa una decisione. Nel caso in cui il modello abbia associato ad un pattern una relazione diversa da *unknown* viene valutata l'accuratezza. Nel caso in cui la relazione sia *unknown* si cerca di inferire una possibile relazione. I parametri che regolano questo processo sono quattro, due per ciascun metodo:

- Pattern Blacklisting:

- `bl_min_len`: quantità minima di predizioni *NON-unknown*;
- `bl_thresh` soglia sull'accuratezza minima per scartare il pattern corrente, compreso nell'intervallo [0, 1], aumentare questo valore consente di scartare più pattern riducendo l'accuracy.

- Pattern Discovery:

- `pd_min_len` Quantità minima di predizioni *NON-unknown*;
- `pd_thresh` Soglia sullo score minimo pattern corrente.

Segue un listato del *main loop* del codice sorgente che esegue gli step descritti. Si consideri che a questo punto Link Prediction ha già espresso, per qualsiasi coppia di entità presente, le relazioni più plausibili, e che queste sono state memorizzate nella struttura `pair2pred` che può essere usata con accesso diretto per ottenere le relazioni predette per ogni coppia.

```
print('\nDeflecting patterns...')

for pattern, track in tqdm(pattern2instances.items()):
    relation, pairs = track
    predictions = [pair2pred[pair] for pair in pairs if 'unknown' not in pair2pred[pair]]
    preds_len = len(predictions)
    if relation != 'unknown' and preds_len >= bl_min_len:
        matches = [relation in pred for pred in predictions]
        pattern_score = sum(matches)/len(matches)
        # Pattern blacklist
        if pattern_score < bl_thresh:
            self.pattern_black_list[pattern] = (relation, pattern_score)
    elif pd_enabled and preds_len >= pd_min_len:
        all_pred_size = [len(pred) for pred in predictions]
        rel2weight = dict()
        for n, preds in enumerate(predictions):
            for rel in preds:
                try:
                    rel2weight[rel] += 1/all_pred_size[n]
                except:
                    rel2weight[rel] = 1/all_pred_size[n]
```

```
max_rel1 = max(rel2weight, key=rel2weight.get)
max_wgh1 = rel2weight[max_rel1]
del(rel2weight[max_rel1])
try:
    max_rel2 = max(rel2weight, key=rel2weight.get)
    max_wgh2 = rel2weight[max_rel2]
except:
    max_wgh2 = 0

max_rel_score = (max_wgh1 - max_wgh2)/len(pairs)
# Pattern discovery
if max_rel_score >= pd_thresh:
    self.pattern_discovery[pattern] = (max_rel1, max_rel_score)
```

Capitolo 6

Risultati sperimentali

L’obiettivo principale di questo lavoro è stato cercare di capire se sia effettivamente possibile utilizzare link prediction per scartare pattern che sono cattivi predittori per una certa relazione. Pertanto nei risultati verrano mostrati i pattern che è stato possibile scartare per ispezione ed un grafico riassuntivo dell’accuratezza di Deflector. Per poter ottenere metriche più accurate sarebbe necessaria una *ground truth* adeguata che indichi quale sia un pattern da scartare e quale invece no. Test futuri più approfonditi consentiranno di valutare metriche più adeguate come precision e recall. La scelta di utilizzare Lector come modello di relation extraction ha portato con se vantaggi e svantaggi, se da un lato la semplicità di funzionamento ed il tipo di features utilizzate consente una valutazione diretta dei risultati rendendoli facilmente interpretabili, e non è stato necessario occuparsi della fase di NER in quanto già disponibili i dati pre-processati, dall’altro si sono dovuti compiere diversi sforzi per adattare appunto questi dati all’intero progetto. Vedremo una breve descrizione di questo nella prossima sezione

6.1 Dati di test

L’intero progetto ha fatto affidamento su gran parte del materiale prodotto da un progetto precedente: *Lector*¹. Quest’ultimo, fortemente legato a DBpedia e FreeBase, ha prodotto una gran quantità di dati che però necessitavano di alcune manipolazioni per poter essere utilizzati in questo progetto. Nello specifico si è fatto uso dell’insieme *labeled triples* contenente tutte le

¹<http://www.dia.uniroma3.it/db/lector/>

triple etichettate da Lector nella fase di generazione del training set via distant supervision come descritto nei capitoli precedenti. Dall'insieme *labeled triples* è stato possibile estrarre una *knowledge base* necessaria all'addestramento di ComplEx ed un insieme di pattern, con un procedimento inverso rispetto a Distant Supervision.

Come si intuisce, i due dataset ottenuti, *pattern* e *knowledge base*, se venissero usati così come sono, porterebbero al risultato iniziale. Inoltre, la knowledge base, che in pratica risulta essere un sottografo di DBpedia, risulta essere estratto casualmente e perciò con caratteristiche topologiche non adatte all'addestramento diretto di ComplEx. Pertanto, è stato necessario estrarre un altro sottografo dalla knowledge base di partenza, che soddisfacesse però determinati requisiti topologici.

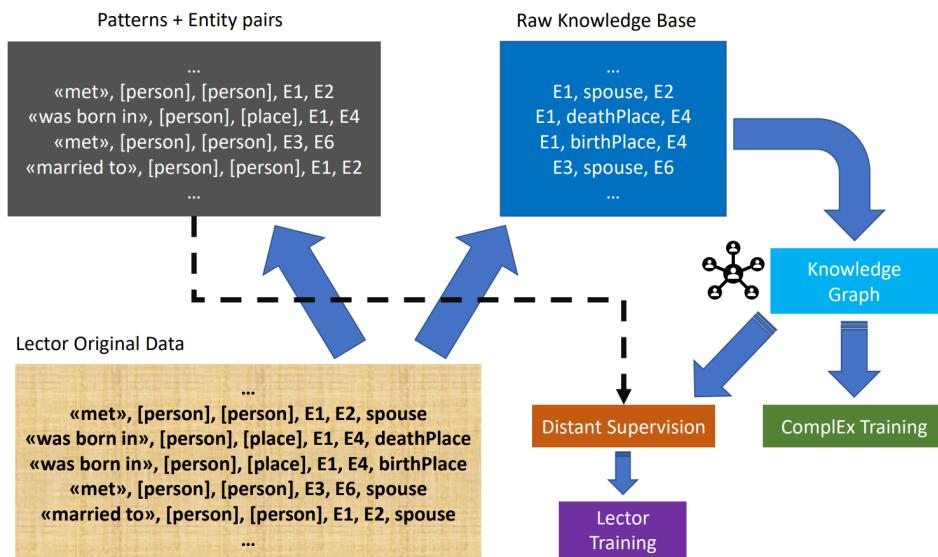


Figura 6.1: **Costruzione dei dataset** Viene eseguito il processo inverso a Distant Supervision sui dati originali per ottenere i dati necessari ad addestrare Link Prediction.

Quindi almeno due aspetti devono essere tenuti presenti:

- Un modello di Link Prediction richiede di essere addestrato su un Knowledge Graph con una certa topologia (i.e. non gli si può passare un training set di triple campionate a caso da una knowledge base) perché l'apprendimento sfrutta proprio le caratteristiche topologiche, come descritto nel capitolo dedicato. Questi è stato risolto con un campionamento casuale *pesato con il degree del nodo*, cioè sono stati

prelevati fatti fra entità con un numero di archi entranti/uscenti al di sopra di una certa soglia per evitare di ottenere un grafo composto da tanti piccoli sotto grafi disconnessi.

- Un modello di LP non può *"rispondere"* per quanto riguarda entità (nodi) mai visti durante l'addestramento. Questo problema viene gestito completamente dal *wrapper* descritto nel capitolo precedente.

6.2 Test effettuati

Con l'intenzione di ottenere subito un'idea del funzionamento del sistema proposto, proponiamo in questa sezione i risultati più significativi e di immediata comprensione. Prima di tutto è stato addestrato Lector con i pattern e la knowledge base già raffinata come mostrato in figura 6.1, ottenuti con il processo inverso sui dati grezzi. Contemporaneamente è stato addestrato ComplEx sulla stessa knowledge base di Lector per prepararlo all'inferenza. Successivamente, viene istanziato *Deflector* passandogli per parametro i wrappers relativi ai due modelli. Nel passo successivo, Deflector cerca di scartare i pattern non validi e scoprirne di nuovi con le modalità descritte precedentemente:

1. Deflector usa il modello di Relation Extraction per estrarre relazioni, avendo cura di *"ricordare"* quale pattern ha contribuito all'estrazione di un fatto. Questa estrazione può essere eseguita sullo stesso insieme di pattern utilizzati per Distant Supervision o su un sottoinsieme di tali pattern non utilizzato. Fondamentalmente vi sono due aspetti da considerare riguardo all'input di Deflector:
 - Deve contenere almeno un buon numero di entità note altrimenti Link Prediction (ComplEx) non saprebbe rispondere per qualsiasi coppia di entità contenente almeno una entità non presente nel Knowledge Graph di addestramento.
 - Questo passo **non** ha come obiettivo l'estrazione di relazioni ma serve a far ricostruire a Lector le associazioni *pattern-relazione* (o parte di esse) che il modello di Relation Extraction induce post-addestramento. Questo perché ragioniamo in un contesto indipendente dal modello, cioè al posto di Lector potrebbe esserci un modello a regressione lineare semplice o una rete neurale.

2. Deflector associa ad ogni pattern la lista di coppie di entità che sono state estratte con tale pattern.
3. Deflector usa link prediction per valutare ciascuna coppia di entità una sola volta, processando il tutto in batch e memorizza per ciascuna coppia le relazioni più plausibili.
4. In base ai parametri di soglia, un pattern associato ad una relazione diversa da "unknown" viene valutato e nel caso scartato (*pattern blacklisting*). In base ad altri parametri di soglia un pattern associato ad una relazione "unknown" viene valutato e nel caso ri-associato (*pattern discovery*). I dettagli di come questo avvenga sono stati affrontati nel capitolo precedente.

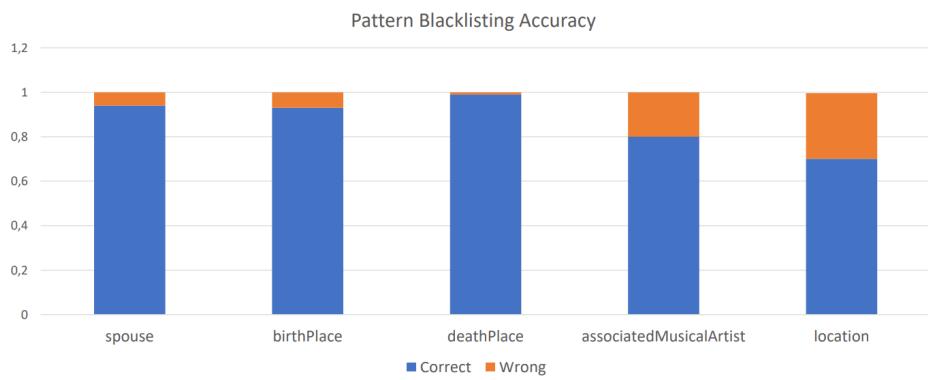


Figura 6.2: Accuratezza. Ogni barra è relativa ad una relazione, viene mostrata l'accuratezza con cui un pattern errato viene effettivamente scartato. Questi risultati sono stati ottenuti con una soglia per blacklisting e discovery = 0.8

In figura 6.2 si mostrano i risultati ottenuti in termini di accuratezza normalizzati nel range [0, 1] per alcune relazioni, è interessante notare come l'accuratezza possa variare a seconda del tipo di relazione a cui viene applicato il metodo proposto, a prescindere dal fatto che sia una relazione *simmetrica* (spouse) o *anti-simmetrica* (birthPlace), ciò lascia intuire che sia riconducibile a caratteristiche sintattiche dei pattern coinvolti da tale relazione.

Si tenga presente comunque che i pattern totali che vengono correttamente scartati restano un numero piuttosto piccolo in confronto a tutti quelli identificati dal modello, cioè potrebbe poi manifestarsi in una bassa recall.

Tuttavia, questo non è necessariamente un aspetto negativo in quanto non è noto a priori quali pattern siano da scartare oppure no, ed il modello di Relation Extraction utilizzato (in questo caso Lector) potrebbe aver correttamente associato un gran numero di pattern, vista anche la quantità di pattern a disposizione. A tal proposito presentiamo due output facilmente interpretabili.

Pattern Discovery: Questi pattern erano stati precedentemente assegnati ad una relazione "unknown" e sono stati ri-assegnati ad una relazione del Knowledge Graph. Nel listato seguente, il formato di ogni riga è:

```
(pattern) ---> (nuova relazione assegnata, score ottenuto via Link Prediction)
```

Anche se sono pochissimi i nuovi pattern identificati, alcuni sono corretti, mentre in certi casi si nota come comunque esista un nesso semantico tra relazione ri-assegnata e pattern:

```
('emigrated to', '[EthnicGroup]', '[Country]') ---> (populationPlace, 0.83)
('emigrated to the', '[EthnicGroup]', '[Country]') ---> (populationPlace, 0.93)
('before', '[MilitaryUnit]', '[MilitaryConflict]') ---> (battle, 0.85)
('at the', '[OfficeHolder]', '[PoliticalParty]') ---> (party, 0.81)
('that', '[MilitaryConflict]', '[OfficeHolder]') ---> (commander, 0.9)
('immigrants to', '[EthnicGroup]', '[Country]') ---> (populationPlace, 0.82)
('ian', '[Country]', '[EthnicGroup]') ---> (ethnicGroup, 1.0)
('is the official language of', '[Language]', '[Country]') ---> (spokenIn, 0.87)
('minority in', '[Language]', '[Country]') ---> (spokenIn, 0.8)
```

In ultima analisi si vuole sottolineare che questa modalità di funzionamento non era inizialmente prevista e rappresenta un esperimento guidato dalla curiosità, pertanto non è stata approfondita nel corso del progetto pur esibendo un comportamento interessante nel suo piccolo.

Pattern Blacklisting: L'obiettivo dell'intero progetto è sempre stato cercare di capire fino a che punto e sotto quali ipotesi potessero essere scartati i pattern erroneamente mappati utilizzando Link Prediction. Il listato seguente mostra una parte dell'output osservabile per la sola relazione **spouse**, il formato di ogni riga è:

```
(pattern) ----> (relazione originalmente associata, score ottenuto via Link Prediction)
```

Si possono notare alcuni errori, segnati con il carattere **x** commessi dal sistema che però nel complesso non intaccano troppo l'accuracy di come anche riassunto in figura 6.2. Mentre invece l'asterisco ***** segnala la presenza dell'ormai ben noto pattern "met" utilizzato negli esempi:

('and ORDINAL lady', '[OfficeHolder]', '[OfficeHolder]') ---> (spouse, 0.76)
 ('death', '[Person]', '[Person]') ---> (spouse, 0.5)
 ('and directed by', '[Person]', '[Person]') ---> (spouse, 0.67)
 ('produced by', '[Person]', '[Person]') ---> (spouse, 0.6)
 ('co starred with', '[Person]', '[Person]') ---> (spouse, 0.73)
 * ('met', '[FictionalCharacter]', '[FictionalCharacter]') ---> (spouse, 0.75)
 ('and of', '[Royalty]', '[Royalty]') ---> (spouse, 0.52)
 ('with the', '[Royalty]', '[Royalty]') ---> (spouse, 0.57)
 ('but', '[Person]', '[Person]') ---> (spouse, 0.22)
 ('accuses', '[FictionalCharacter]', '[FictionalCharacter]') ---> (spouse, 0.43)
 ('as PERSON', '[Person]', '[Person]') ---> (spouse, 0.78)
 ('brings', '[FictionalCharacter]', '[FictionalCharacter]') ---> (spouse, 0.5)
 ('to', '[Monarch]', '[Royalty]') ---> (spouse, 0.5)
 ('and his', '[Royalty]', '[Royalty]') ---> (spouse, 0.62)
 ('urged', '[Royalty]', '[Royalty]') ---> (spouse, 0.62)
 ('and PERSON', '[Royalty]', '[Royalty]') ---> (spouse, 0.38)
 x ('ORDINAL wife', '[OfficeHolder]', '[OfficeHolder]') ---> (spouse, 0.6)
 ('in DATE when', '[Royalty]', '[Royalty]') ---> (spouse, 0.6)
 ('whom', '[Person]', '[MusicalArtist]') ---> (spouse, 0.44)
 ('saved', '[FictionalCharacter]', '[FictionalCharacter]') ---> (spouse, 0.6)
 ('took', '[Royalty]', '[Royalty]') ---> (spouse, 0.57)
 ('announced that', '[Person]', '[Person]') ---> (spouse, 0.1)
 ('by', '[Monarch]', '[Royalty]') ---> (spouse, 0.6)
 ('coronation', '[Royalty]', '[Royalty]') ---> (spouse, 0.33)
 * ('met', '[OfficeHolder]', '[OfficeHolder]') ---> (spouse, 0.66)
 ('confronts', '[FictionalCharacter]', '[FictionalCharacter]') ---> (spouse, 0.48)
 x ('and his wife', '[President]', '[OfficeHolder]') ---> (spouse, 0.75)
 ('during the reign of', '[Royalty]', '[Royalty]') ---> (spouse, 0.5)
 ('in which', '[Person]', '[Person]') ---> (spouse, 0.6)
 ('and producer', '[Person]', '[Person]') ---> (spouse, 0.6)
 ('suggests to', '[FictionalCharacter]', '[FictionalCharacter]') ---> (spouse, 0.56)
 ('from', '[Royalty]', '[Royalty]') ---> (spouse, 0.67)
 ('ORDINAL lady', '[OfficeHolder]', '[OfficeHolder]') ---> (spouse, 0.64)
 ('whom', '[Royalty]', '[Royalty]') ---> (spouse, 0.4)
 ('appeared with', '[Person]', '[Person]') ---> (spouse, 0.5)
 ('with whom', '[Artist]', '[Artist]') ---> (spouse, 0.43)
 ('where', '[Royalty]', '[Royalty]') ---> (spouse, 0.5)
 ('and actress', '[MusicalArtist]', '[Person]') ---> (spouse, 0.62)
 ('for', '[Royalty]', '[Royalty]') ---> (spouse, 0.44)
 ('and actor', '[Person]', '[Person]') ---> (spouse, 0.53)
 ('on DATE and', '[Royalty]', '[Royalty]') ---> (spouse, 0.4)
 ('to', '[President]', '[OfficeHolder]') ---> (spouse, 0.33)
 ('was', '[Royalty]', '[Royalty]') ---> (spouse, 0.36)
 ('and sister in law', '[Person]', '[Person]') ---> (spouse, 0.67)
 ('to the', '[Royalty]', '[Royalty]') ---> (spouse, 0.69)
 ('and the late', '[Person]', '[Person]') ---> (spouse, 0.58)
 ('lets', '[FictionalCharacter]', '[FictionalCharacter]') ---> (spouse, 0.67)
 ('for', '[Person]', '[Person]') ---> (spouse, 0.5)
 ('and when', '[Royalty]', '[Royalty]') ---> (spouse, 0.6)
 ('accompanied by', '[Royalty]', '[Royalty]') ---> (spouse, 0.5)
 ('arranged for', '[Royalty]', '[Royalty]') ---> (spouse, 0.6)
 ('starred with', '[Person]', '[Person]') ---> (spouse, 0.67)

Capitolo 7

Conclusioni e sviluppi futuri

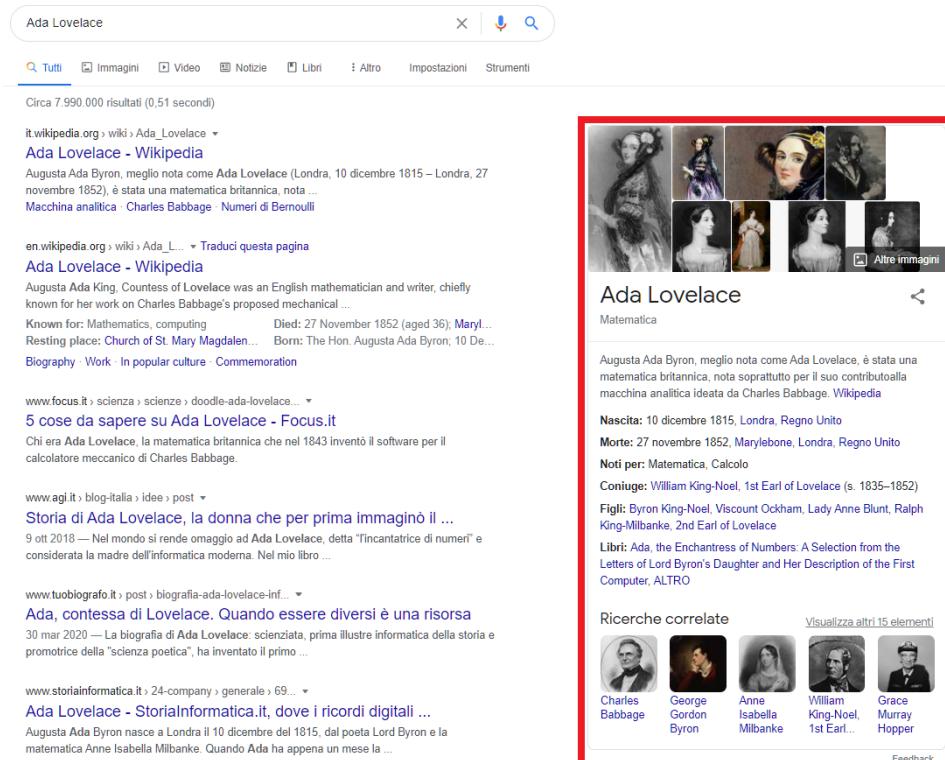


Figura 7.1: **Google Knowledge Graph.** La pagina di ricerca mostra lateralmente le informazioni su ciò che si è cercato e tutti i possibili collegamenti.

Al giorno d'oggi i knowledge graph rappresentano la spina dorsale di molti servizi offerti al pubblico, dai motori di ricerca ai social network e spesso l'utente finale ne ignora completamente le meccaniche interne, dopotutto

Non è necessario sapere come funziona un motore a combustione per poter guidare un'automobile.

Perseguire l'obiettivo di rendere queste strutture ancora più complete e corrette, è il passo necessario da compiere nell'era del *Semantic Web*: rappresentare le informazioni non più come stringhe di testo ma come entità del mondo reale.

Un altro esempio dell'utilizzo dei Knowledge Graph, su cui vale la pena spendere qualche riga, è legato al periodo storico che stiamo vivendo.

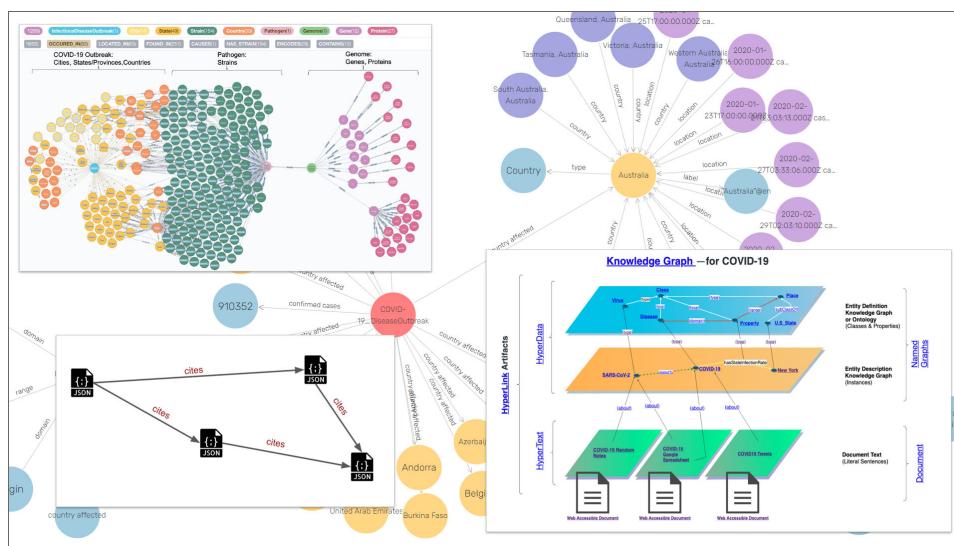


Figura 7.2: I knowledge graph per la lotta al COVID-19.

Fonte: *The Graph Lounge*.

<https://theographlounge.com/knowledge-graphs-in-the-fight-against-covid-19>

In questo particolare periodo, in cui il mondo intero ha avuto a che fare con il diffondersi di un'epidemia, la comunità scientifica è stata messa alla prova per cercare di ottenere risultati in tempi brevi. Per i ricercatori e coloro che hanno dovuto preparare delle strategie di risposta per combattere e contenere il diffondersi del virus COVID-19, cercare di dare un senso all'enorme quantità di dati raccolta da una moltitudine di sorgenti diverse ha richiesto un altrettanto enorme sforzo. Fortunatamente, dare un senso a dati disordinati provenienti da sorgenti multiple è una delle cose in cui i Knowledge Graph riescono meglio. Inoltre, rendono possibile derivare nuova conoscenza da quella contenuta al loro interno connettendo in maniera intelligente le informazioni.

Per quanto riguarda **Deflector**, alla luce dei primi risultati sperimentali ottenuti, è ragionevole pensare che valga la pena indagare ulteriormente sul metodo proposto, eventualmente modificandolo e mettendolo alla prova in tutti i suoi aspetti. Al di là dei pochi parametri che possono regolare il funzionamento di Deflector è bene notare che trattandosi di un *metamodello* cioè un modello composto da altri modelli, le combinazioni possibili sono davvero elevate.

Qui di seguito sono elencati alcuni punti da sviluppare raggruppati per categorie:

- **Relation Extraction.** Lector è senza alcun dubbio un modello che comporta diversi vantaggi: la semplicità dei concetti su cui si basa facilitano la comprensione del funzionamento di questo tipo di modelli e rendono accessibile la re-implementazione a scopo di test. Il poter visualizzare direttamente i pattern consente un'esperienza diretta del funzionamento effettivo di tutto il modello, senza doversi basare su metriche particolari che potrebbero essere soggette ad errori di calcolo. Tuttavia è naturale pensare di applicare l'idea alla base di Deflector anche usando modelli di Relation Extraction alternativi, potendo beneficiare di benchmark approvati dalla comunità di ricerca.
 - **Link Prediction.** In questo progetto si è impiegata un'implementazione già pronta di ComplEx, principalmente perché ha mostrato i risultati migliori rispetto alla concorrenza [40]. Tuttavia conoscendo la variabilità dei risultati di tali sistemi, legata al particolare knowledge graph di addestramento, si potrebbe esplorare altre soluzioni magari individuandone una ancora più adatta al contesto ed alle caratteristiche del knowledge graph.
- Un altro aspetto che dovrebbe essere approfondito riguarda l'impatto delle limitazioni di Link Prediction sulla conoscenza esterna al grafo, cioè che si può ragionare solo in termini di coppie che il modello conosce ed ha "visto" durante l'addestramento.
- **Deflector.** Al di là della semplice attività di *parameter searching* che prevede di eseguire numerosi test, con diverse combinazioni di parametri, il modello proposto offre senza dubbio un vasto banco di prova in due principali direzioni:

- *Pattern Discovery.* Attualmente il sistema per la scoperta dei pattern è basato su una formula interna che potrebbe essere oggetto di ottimizzazioni future. Inoltre non è stato affatto esplorata la possibilità di impiegare i **word embeddings** per individuare nuovi pattern, soluzione forse un po’ troppo vincolata al tipo di pattern utilizzato ma certamente interessante.
- *Pattern Blacklisting.* Occorre stabilire con certezza e misurare precisamente il vantaggio ottenibile da questo sistema mediante metriche largamente impiegate come *precision* e *recall*, per fare questo è necessaria una *ground truth* costruita ad hoc sull’insieme di pattern appresi dal modello. Alternativamente un’idea che vale la pena valutare è quella di utilizzare un sistema di generazione automatica di pattern volutamente errati da ”iniettare” nei pattern usati per l’addestramento del modello di Relation Extraction. Questo potrebbe essere fatto considerando le proprietà di correlazione e, scegliendo casualmente dei pattern reali, generare in per ciascuno di essi un pattern corrotto, imitando così la correlazione. Sapendo quali pattern sono stati generati e quali no, sarebbe poi semplice calcolare le metriche sopracitate.

In ultima analisi è possibile effettuare due tipi di valutazioni: 1) misurare la capacità di scartare i pattern e 2) misurare la capacità di estrarre relazioni, prima e dopo che sono stati scartati i pattern. Mentre nel primo caso è necessario costruire una *ground truth* valida potrebbe non esserlo nel secondo perché è una modalità classica di valutazione per un modello di Relation Extraction e quindi ottenibile via dataset comunemente disponibili (a patto di cambiare Lector con qualcos’altro).

- **Distant Supervision.** Nel corso di questa trattazione è stato detto, in relazione al problema del rumore causato dall’assunzione troppo lasca di Distant Supervision, che vi sono due strade percorribili con l’intenzione di sfruttare Link Prediction per risolvere tale problema. L’approccio di Deflector è del tutto *a posteriori* rispetto all’addestramento del modello di Relation Extraction, mentre un approccio *a priori* sarebbe più in linea con la gran parte degli studi fatti in questo settore. Consisterebbe, in pratica, nel lavorare sul dataset generato

via Distant Supervision direttamente, con l'intento di scartare i pattern errati prima ancora che possano essere dati in pasto al modello di Relation Extraction, riuscendo nell'impresa, si otterrebbero certamente risultati tendenti a quelli che si sono osservati con l'etichettatura manuale via crowdsourcing.

Bibliografia

- [1] Google knowledge graph wikipedia article.
- [2] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *In Proceedings of the 5th ACM International Conference on Digital Libraries*, pages 85–94, 2000.
- [3] Naser Ahmadi, Viet-Phi Huynh, Vamsi Meduri, Stefano Ortona, and Paolo Papotti. Mining expressive rules in knowledge graphs. *J. Data and Information Quality*, 12(2), May 2020.
- [4] Nguyen Bach and Sameer Badaskar. A review of relation extraction. 05 2011.
- [5] L. Backstrom and J. Leskovec. Supervised random walks: Predicting and recommending links in social networks, 2010.
- [6] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.
- [7] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *NIPS*, pages 2787–2795, 2013.
- [8] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI’11, page 301–306. AAAI Press, 2011.
- [9] Sergey Brin. Extracting patterns and relations from the world wide web. In *Selected Papers from the International Workshop on The World Wide Web and Databases*, WebDB ’98, page 172–183, Berlin, Heidelberg, 1998. Springer-Verlag.

- [10] Razvan Bunescu and Raymond Mooney. Subsequence kernels for relation extraction. 01 2005.
- [11] Matteo Cannaviccio, Denilson Barbosa, and Paolo Merialdo. Accurate fact harvesting from natural language text in wikipedia with lector. In *Proceedings of the 19th International Workshop on Web and Databases*, WebDB '16, New York, NY, USA, 2016. Association for Computing Machinery.
- [12] Jim Cowie and Wendy Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, 1996.
- [13] Xin Luna Dong, Evgeniy Gabrilovich, Jeremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 601–610, 2014. Evgeniy Gabrilovich Wilko Horn Ni Lao Kevin Murphy Thomas Strohmann Shaohua Sun Wei Zhang Jeremy Heitz.
- [14] Xin Luna Dong and Divesh Srivastava. Big data integration. In *2013 IEEE 29th international conference on data engineering (ICDE)*, pages 1245–1248. IEEE, 2013.
- [15] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. *Commun. ACM*, 51(12):68–74, December 2008.
- [16] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *ARTIFICIAL INTELLIGENCE*, 165:91–134, 2005.
- [17] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam Mausam. Open information extraction: The second generation. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, page 3–10. AAAI Press, 2011.

- [18] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.
- [19] Michael Färber, Basil Ell, Carsten Menne, and Achim Rettinger. A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web Journal*, July, pages 1–26, 2015.
- [20] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with amie. *The VLDB Journal*, 24(6):707–730, December 2015.
- [21] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW ’13, page 413–422, New York, NY, USA, 2013. Association for Computing Machinery.
- [22] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [23] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gérard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs, 2021.
- [24] Eduard Hovy, Roberto Navigli, and Simone Paolo Ponzetto. Collaboratively built semi-structured content and artificial intelligence: The story so far. *Artificial Intelligence*, 194:2–27, 2013. Artificial Intelligence, Wikipedia and Semi-Structured Resources.
- [25] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition and applications, 2021.
- [26] Jeremy Kepner, Henning Meyerhenke, Scott McMillan, Carl Yang, John D. Owens, Marcin Zalewski, Timothy Mattson, Jose Moreira,

Peter Aaltonen, David Bader, and et al. Mathematical foundations of the graphblas. *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep 2016.

- [27] Ni Lao, William W. Cohen, L Balcázar, Francesco Bonchi, Aristides Gionis, Michèle Sebag, N. Lao, and W. W. Cohen. Relational retrieval using a combination of path-constrained random walks. In *Machine Learning*, 2010.
- [28] Ni Lao, Tom Mitchell, and William W Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 529–539. Association for Computational Linguistics, 2011.
- [29] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58(7):1019–1031, May 2007.
- [30] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-grained evaluation of rule- and embedding-based systems for knowledge graph completion. In Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web – ISWC 2018*, pages 3–20, Cham, 2018. Springer International Publishing.
- [31] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. ECML PKDD’11, page 437–452, Berlin, Heidelberg, 2011. Springer-Verlag.
- [32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.
- [33] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. Distant supervision for relation extraction without labeled data. volume 2, pages 1003–1011, 01 2009.
- [34] Allen Newell, John C. Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959.

- [35] Thien Huu Nguyen and Ralph Grishman. Relation extraction: Perspective from convolutional neural networks. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 39–48, Denver, Colorado, June 2015. Association for Computational Linguistics.
- [36] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. ICML’11, page 809–816, Madison, WI, USA, 2011. Omnipress.
- [37] Joshua O’Madadhain, Jon Hutchins, and Padhraic Smyth. Prediction and ranking algorithms for event-based network data. 7(2):23–30, December 2005.
- [38] Alexandrin Popescul, Rin Popescul, and Lyle H. Ungar. Statistical relational learning for link prediction, 2003.
- [39] R. H. Richens. Preprogramming for mechanical translation. *Mech. Transl. Comput. Linguistics*, 3:20–25, 1956.
- [40] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction. *ACM Transactions on Knowledge Discovery from Data*, 15(2):1–49, Mar 2021.
- [41] Baoxu Shi and Tim Weninger. Open-world knowledge graph completion. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.
- [42] Yusuke Shinyama and Satoshi Sekine. Preemptive information extraction using unrestricted relation discovery. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 304–311, New York City, USA, June 2006. Association for Computational Linguistics.
- [43] Edward Shortliffe. *Computer-based medical consultations: MYCIN*, volume 2. Elsevier, 2012.
- [44] Alisa Smirnova and Philippe Cudré-Mauroux. Relation extraction using distant supervision: A survey. *ACM Comput. Surv.*, 51(5), November 2018.

- [45] Frans N. Stokman and Pieter H. de Vries. Structuring knowledge in a graph. In *Conference of the Dutch Psychonomic Society on Human-Computer Interaction: Psychonomic Aspects*, page 186–206, Berlin, Heidelberg, 1988. Springer-Verlag.
- [46] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space, 2019.
- [47] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML’16, page 2071–2080. JMLR.org, 2016.
- [48] Zhen Wang, J. Zhang, Jianlin Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.
- [49] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW ’14, page 515–526, New York, NY, USA, 2014. Association for Computing Machinery.
- [50] Fei Wu and Daniel S. Weld. Open information extraction using Wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 118–127, Uppsala, Sweden, July 2010. Association for Computational Linguistics.
- [51] Han Xiao, Minlie Huang, and Xiaoyan Zhu. From one point to a manifold: Knowledge graph embedding for precise link prediction. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI’16, page 1315–1321. AAAI Press, 2016.
- [52] Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. *J. Mach. Learn. Res.*, 3(null):1083–1106, March 2003.