Node Classification e Link Prediction su Knowledge Graph tramite Graph Neural Network

di Macaluso Girolamo e Mistretta Marco

Università degli studi di Firenze

Laurea Magistrale in Intelligenza Artificiale

9 marzo 2022

Sommario

Il presente lavoro è stato svolto come esame e project work in Knowledge Engineering, presso l'Università degli Studi di Firenze, sotto la supervisione del Professore Bellini Pierfrancesco. L'obbiettivo era quello di implementare una Graph Neural Network al fine di effettuare Node Classification e Link Prediction su di un sottografo del Knowledge Graph **Km4City**. Implementato tale modello, lo step successivo sarebbe stato dunque quello di confrontare i risultati ottenuti con quelli di modelli preesistenti, riadattati per essere applicati a tale Ontologia.

Keywords
Node Classification, Link Prediction, GNN, Km4city

1 Introduzione

I Knowledge graphs sono rappresentazioni strutturate di fatti del mondo reale. Tuttavia, in genere contengono solo un piccolo sottoinsieme di tutti i fatti possibili. L'obbiettivo sarebbe quello di dedurre i fatti mancanti sulla base di quelli esistenti. Tuttavia, al momento non tutte le informazioni disponibili sono memorizzate nei Knowledge graphs e l'aggiunta manuale di nuove informazioni è costosa, il che crea la necessità di algoritmi in grado di dedurre automaticamente i fatti mancanti.
Node Classification e Link Prediction sono task ampiamente studiati nell'ambito del Graph Representation Learning. Node Classification è un'attività di apprendimento automatico applicata ad un grafo, consiste nell'addestrare un modello per apprendere a quale classe appartiene un nodo. Link Prediction invece addestra un modello per imparare, tra coppie di nodi in un grafo, dove dovrebbero esistere delle relazioni.

L'obbiettivo di questo paper è invece quello di proporre due soluzioni GNN-based per la risoluzione di entrambi i task ed applicare tale modello ad un sottografo del Knowledge Graph Km4city. I due modelli che andremo ad illustrare prendono il nome di RGCN (Relational Graph Convolutional Network) [1], per quanto riguarda la Node Classification, e DQ-GNN (Dual Quaternion GNN) di NoGe [2] per quanto riguarda l'operazione di Link Prediction. Ai fini di avere un termine di paragone qualitativo e quantitativo in termini di accuracy e tempo di esecuzione, abbiamo deciso successivamente di confrontare il primo modello con Decision Tree di MINDWALC [3] ed il secondo modello con TransE di Pykeen [4], nella risoluzione dei rispettivi problemi.

2 Cenni Teorici

2.1 Knowledge Graph

Nell'ambito della Knowledge Representation e del Knowledge Reasoning, il Knowledge Graph è una Knowledge Base che utilizza un modello di dati strutturato a grafo. I Knowledge Graph sono spesso usati per memorizzare descrizioni interconnesse di entità quali oggetti, eventi, situazioni o concetti astratti, codificando anche la semantica alla base della terminologia utilizzata.

2.1.1 Km4city

Km4City [5] è una Knowledge Base e una research line del laboratorio DISIT sviluppata principalmente prima dell'inizio dei progetti *Sii-Mobility*, *RESOLUTE*, *REPLICATE*, ma migliorata con essi. Km4City nasce nel 2013 con un nome generico: *smart city ontology*. Le principali informazioni e modelli in quel momento

erano focalizzati sugli Open Data del Comune di Firenze e sfruttavano *Lamma* per il meteo. I principali set di dati informativi sono stati: localizzazione digitale, attività culturali, scuole, attività commerciali, ecc. Successivamente, come detto sopra, con lo sviluppo di progetti correlati la Knowledge Base si è ampliata con essi.

2.1.2 Apache Jena Fuseki

Apache Jena Fuseki è un server SPARQL. Può essere eseguito come servizio del sistema operativo, come applicazione Web Java (file WAR) e come server standalone. Fuseki è disponibile in due forme, un'unica *webapp* di sistema, combinata con un'interfaccia utente per l'amministratore e le query, e come *main*, un server adatto per essere eseguito come parte di una distribuzione più ampia, fornito di un proprio *Docker*. Entrambi i moduli utilizzano lo stesso motore di protocollo principale e lo stesso formato di file di configurazione. Fuseki fornisce i protocolli SPARQL 1.1 per le query e l'aggiornamento, nonché il SPARQL *Graph Store protocol*. Fuseki è strettamente integrato con *TDB* per fornire un robusto livello di archiviazione persistente transazionale e incorpora la query di testo Jena.

2.1.3 SPARQLWrapper

SPARQLWrapper è un semplice wrapper Python di servizi SPARQL che permette di eseguire query da remoto. Aiuta l'utente permettendo di creare un Classe di invocazione delle query e, facoltativamente, convertendo il risultato in un formato più gestibile.

2.1.4 Knowledge Graph Embeddings

Le prestazioni degli algoritmi di apprendimento automatico dipendono strettamente dalla rappresentazione dei dati. Il Representation Learning migliora le prestazioni del machine learning derivando l'insieme delle feature che servono per rappresentare i dati senza l'intervento umano. Vale a dire, l'obiettivo di un algoritmo Representation Learning è quello di dedurre automaticamente le caratteristiche latenti - o nascoste - dei dati. Queste caratteristiche sono rappresentate come vettori densi e di bassa dimensione, noti come embedding, che non vengono estratti direttamente come quantità nei dati, ma le cui variazioni influenzano ogni singola informazione che siamo in grado di osservare.
I Knowledge Graph Embedding (KGE) sono il risultato di specifici modelli di Representation Learning applicati ai KG. L'obiettivo dei modelli KGE è quelle di incorporare le informazioni contenute nel KG - entità e relazioni - in uno spazio vettoriale continuo e a bassa dimensione. I fattori latenti proiettati nei KGE hanno un ruolo essenziale nell'analisi e nell'estrazione di ulteriore soft-knowledge nei KG. Infatti, per ogni coppia di entità $\{s,\ o\}$ e per ogni relazione r, è possibile determinare se un'affermazione $(s,\ r,\ o)$ è vera secondo gli embedding appresi dalle tecniche KGE. Tradizionalmente, gli approcci KGE definiscono una funzione di punteggio per ogni istruzione KG $(s,\ r,\ o)$.
In generale, i KG includono solo affermazioni vere, mentre affermazioni inesistenti possono essere considerate mancanti o false (assunzione del mondo chiuso). Di conseguenza, la funzione di punteggio restituisce un punteggio più alto per le affermazioni esistenti e un punteggio più basso viceversa. Considerando questo tipo di funzione di punteggio, nella maggior parte dei casi, l'obiettivo dell'addestramento può essere formalizzato come un problema di classificazione binaria.

2.2 Graph Neural Network

Le tecniche di KGE [6] codificano le interazioni tra entità e relazioni attraverso modelli che non sono costruiti in modo nativo per la codifica delle strutture dei grafi. Tuttavia, è stata proposta una nuova famiglia di architetture neurali per affrontare questa limitazione. In questo contesto, le Graph Neural Network (GNN) [1] stanno diventando la struttura chiave per l'apprendimento della rappresentazione latente di dati strutturati a grafo. Il principio di funzionamento alla base delle GNN si basa sull'analogia tra Neural Network e grafi. In effetti, le architetture neurali sono formalizzate come grafi pesati e diretti, i cui nodi definiscono le unità computazionali - o neuroni artificiali - e gli archi sono le connessioni pesate tra queste unità. Tuttavia, la topologia delle NN tradizionali è strutturata a priori per uno scopo specifico. Ad esempio, le architetture feed-forward forniscono una sequenza di livelli completamente connessa e omogenea: ogni nodo del livello precedente è collegato da un bordo pesato a tutti i nodi del livello successivo. Al contrario, la struttura dei dati del grafo è eterogenea perché la topologia è guidata direttamente dagli archi definiti tra i nodi. I GNN si basano su architetture neurali progettate seguendo la topologia dei dati del grafo, in cui le connessioni pesate del NN corrispondono agli archi nella struttura del grafo. La classe di GNN più adottata è nota come Graph Convolutional Networks (GCNs). L'obbiettivo dei GCN

è aggiornare la rappresentazione dei nodi da uno strato all'altro sulla base della struttura dei vicini.
L'idea chiave dei GNN è che la rappresentazione di un nodo nel grafo viene aggiornata con le caratteristiche dei suoi vicini, attraverso l'operazione di convoluzione calcolata con parametri (o pesi) condivisi attraverso la struttura locale. Questo calcolo viene eseguito in parallelo per tutti i nodi ad ogni aggiornamento della rete. Inoltre, impilando più livelli, è possibile acquisire e codificare le relazioni tra nodi su più hop nel grafo.

2.3 Node Classifcation

L'obbiettivo è quello di determinare le label dei sample (in questo caso rappresentati dai nodi del grafo) osservando le label dei loro vicini. Per semplicità definiamo qui *hop* come la profondità con la quali si andrà a ricercare informazioni nei vicini dei nodi sui quali vogliamo fare la classificazione. Di solito, problemi di questo tipo vengono addestrati in modo semi-supervisionato, con solo una parte del grafo etichettata

2.4 Link Prediction

I Knowledge graphs sono rappresentazioni strutturate di fatti del mondo reale. Tuttavia, in genere contengono solo un piccolo sottoinsieme di tutti i fatti possibili. Link prediction ha il compito di dedurre i fatti mancanti sulla base di quelli esistenti, l'algoritmo deve comprendere la relazione tra le entità nel Grafo e cercare anche di prevedere se esiste una connessione tra due entità. È essenziale nelle social network per dedurre interazioni sociali o suggerire possibili amici agli utenti. È stato anche utilizzato nei sistemi di raccomandazione e nella previsione delle associazioni criminali.

3 I Modelli

3.1 RGCN

Le Relational Graph Convolution Networks (RGCNs) sono framework di passaggio di messaggi per l'apprendimento di caratteristiche latenti dei grafi relazionali. Le RGCN sono state ampiamente adottate per combinare Knowledge Graph con applicazioni di machine learning. Per maggiori informazioni rimandiamo a [7]

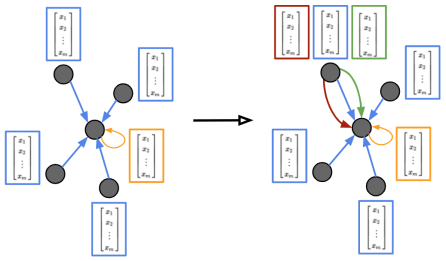

Figura 1: RGCN message passing

3.2 NoGE DQ-GNN

Il package NoGE utilizza un innovativo approccio *GNN-based* per fare link prediction su un Knowledge Graph: costruisce un modello *Node Co-occurrence based Graph Neural Networks*, cioè costruisce un grafo singolo considerando le entità e le relazioni come nodi individuali e quindi calcola i pesi per gli archi tra i nodi in base alla co-occorrenza di entità e relazioni. NoGE sfrutta una DQ-GNN (Dual Quaternion GNN) per aggiornare le rappresentazioni vettoriali dei nodi di entità-relazione. Risultati sperimentali completi mostrano che NoGE ottiene risultati all'avanguardia su tre nuovi e difficili set di dati di riferimento CoDEx.

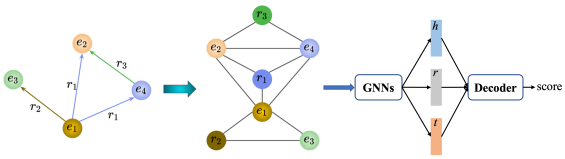

Figura 2: Node Co-occurrence GNN

3.3 MINDWALC Decision Tree

Il package MINDWALC implementa una tecnica che estrae cammini interpretabili da Knowledge Graph che sono molto informativi per un certo problema di classificazione. I cammini sono di un formato specifico per consentire la creazione di strutture di dati che si traducono in un mining molto efficiente. MINDWALC combina il mining dei cammini con diversi approcci per classificare i nodi all'interno di un grafo. In particolare Decision Tree MINDWALC estrae in modo efficiente un tipo specifico di cammini che massimizzano il guadagno di informazioni. I cammini hanno la forma seguente: inizia con una radice, seguita da 1 - 2 caratteri jolly (*) e quindi un'entità denominata. Un esempio potrebbe essere *root->*->*->*->Gent* che corrisponderebbe alla passeggiata *GillesVandewiele–>studyAt–>Ghent University–>LocatedIn–>Ghent.* Per questo, root è sostituito dall'istanza che stiamo classificando.

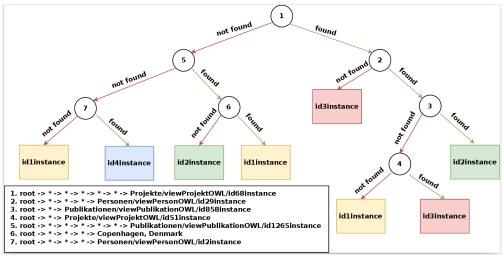

Figura 3: Decision Tree

3.4 Pykeen TuckER

Come confronto del metodo di link prediction basato su GNN (DQ-GNN) abbiamo scelto di utilizzare TuckER in una sua implementazione di Pykeen. TuckER è un modello lineare relativamente semplice ma potente basato sulla *Tucker Decomposition* del tensore binario delle triple, dalla quale si ottiene un tensore di dimensione inferiore e tre matrici (una per l'embedding dei subject, una per quello delle relazioni e una per quello degli object). Ciò permette di avere una valutazione di tutte le possibili triple tra le entità e le relazioni nel Knowledge Graph tramite l'applicazione di una semplice funzione di score.

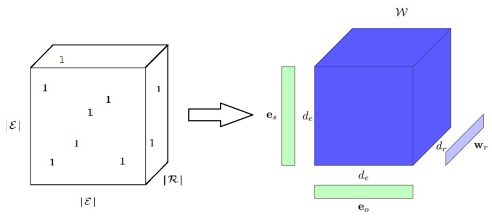

Figura 4: Tucker Decomposition

4 Implementazione Python

4.1 Preparazione dei Dataset

4.1.1 Download del Dataset

Innanzitutto dovevamo ottenere tutte le triple del nostro sottografo di interesse: "Eventi a Firenze". Dopo svariati tentativi, con differenti Sparql endpoint Query Editor, abbiamo deciso di cercare di automatizzare il processo di estrazione tramite una funzione in Python.

L'idea era quella di:

1. estrarre tutte le entità *Event*
2. scorrere tutte le entità trovate e per ciascuna di esse estrarre tutte le triple a cui esse partecipano
3. iterare per un numero pari alla profondità di *hop* desiderata

A livello implementativo abbiamo dunque settato un Endpoint Apache Jena Fuseki in local host, instanziato una classe SPARQLWrapper con edpoint quello di Fuseki sulla porta 3030, e automatizzato la scrittura delle query opportune con service l'effettivo Endpoint di km4city. In media con tale processo si eseguivano 13 query al secondo, per un totale di circa 700'000 triple estratte in 15 minuti. Le query ottenute con questo processo erano ovviamente ridondanti, in proporzione al grado di *hop* scelto, venivano dunque ovviamente rese uniche e revisionate.
Il sistema di interrogazione è stato realizzato in questo modo per via le limitazioni dell'endpoint, cioè il massimo numero dei risultati restituiti era pari a 40'000 alla volta.

4.1.2 Riformattazione dei dati

Le query estratte venivano dunque, salvate su un file TSV. Tale file è di fatto il punto di partenza, per ogni qual si voglia riformattazione del dataset, in formati compatibili con un genrico modello di node classifciation o link prediction dello stato dell'arte attuale. Siamo inffatti riusciti ad incapsulare in una funzione di "reformat" tutto il necessario per ottenere a real-time, da un qualsiasi file dataset.tsv, tutti i file necessari per fare training, testing.
In particolare per km4city:

- Per la node classification viene creato in automatico un file contenente tutte le triple tranne quelle che fanno riferimento alla categoria delle evento e un file, da suddividere in train e test, che contiene solo le triple che specificano la categoria di ciascun evento
- Per quanto riguarda invece link prediction invece l'insieme delle triple viene direttamente suddiviso in train e test.

P.S.: Per raffinare la fase di tuning abbiamo implementato una funzione di split del dataset che ci permette di controllare size e seed di divisione.

4.2 Node Classification

Nel nostro caso l'obbiettivo della node classification era quello di predire la categoria (`http://www.disit.org/km4city/schema#eventCategory`) di un evento (`http://www.disit.org/km4city/schema#Event`). Le categorie presenti all'interno di *km4city* sono 26, tra cui citiamo: *Teatro*, *Sport*, *News*, ecc. Nel grafo

in input sono presenti tutte le informazioni riguardo agli eventi, in particolare quello che sembra essere più utile a capire il tipo di evento è la *location*.
L'input in questo caso sarà il sottografo degli eventi con tutti le relazioni a meno di tre hop di distanza senza, ovviamente la classificazione degli eventi, e il training set che contiene un insieme di eventi per i quali è stata assegnata la label.

4.2.1 R-GCN

É stato il primo modello che abbiamo implementato. Abbiamo deciso di incapsulare in una stessa classe di nome *Model*, sia l'RGCN-layer, che l'ottimizzatore (*Adam*), che il grafo (*km4city*), che viene utilizzato nel costruttore per inizializzare tale classe. Abbiamo deciso di utilizzare una nostra versione dell'RGCN-layer simil-DGL, così da rendere il caricamento dei dati all'interno della classe più semplice e generico. Simil-DGL così da permettere di rendere tale classe robusta ad eventuali variazioni di interfaccia e aggiornamenti della libreria (è come se fosse stato implementato un *Adapter* tra la GNN e il RGCN-layer). Abbiamo scelto DGL perché per la node classification usa dataset espressi nello stesso formato dei principali modelli dello stato dell'arte.

4.2.2 Decision Tree

L'implementazione usata è quella di MINDWALC che prende in input l'insieme delle triple senza quelle che definisco la relazione di EventCategory, il training e il test set che contengono rispettivamente 80 e 20 percento delle triple con oggetto eventi predicato EventCategory e oggetto la categoria. I parametri del modello sono *path_max_depth* e *min_samples_leaf*, il primo è impostato a 5 e rapprensenta la profondità massima dei decision tree creati; il secondo impostato a 1 rappresenta il numero minimo di foglie che deve avere un albero al livello più basso. MINDWALC può essere combinato con tre diversi approcci di classificazione, nella nostra trattazione abbiamo scelto di utilizzare la sua versione base: come *Decision Tree con cammini opportuni*.

5 Link Prediction

5.0.1 DQ-GNN

Come già accennato, non siamo stati in grado di implementare autonomamente un modello GNN-based per fare link prediction. Ci siamo dunque appoggiati alla libreria NoGe. NoGe costruisce un similarity graph, considerando relazioni ed entità, come nodi distinti di tale grafo. Su tale similiarity graph va poi a pesare gli archi (che forniscono un punteggio di score calcolato mediante il decoder QuatE [8] tramite una encoder QGNN. Viene di fatto implementato un modello di Autoenchoder, stato dell'arte attuale nell'utilizzo delle GNN per la link prediction [1]. L'implementazione in python si è ricondotta dunque a modificarne la sua interfaccia di instanzazione e di utilizzo ai fini di renderla compatibile con il resto del sistema e del dataset.) una parola

5.0.2 TuckER

L'implementazione utilizzata è quella di Pykeen, richiamata mediante il metodo *pipeline()* dando in input i parametri del modello, quali: il numero delle epoche, la dimensione dello spazio di embedding e il tipo di modello. Essendo tale modello molto sensibile alla *dimensione dello spazio di embedding*, abbiamo deciso, in fase di implementazione, di sperimentare varie opzioni. Dopo svariati test è risultato evidente che la scelta migliore, giusto compromesso tra performance e accuratezza, fosse quella di optare per 30. Si osserva che tale valore si colloca al di sotto di quello tipico considerato in contesti applicativi analoghi.

6 Analisi e Confronto

Implementati i modelli, esguito il tuning degli iperparametri, abbiamo deciso di confrontarli.
I termini di paragone presi in considerazione sono stati rispettivamente:

- Per la node classification: *accuracy* sulla predizione e tempo di esecuzione
- Per la link prediction: *loss* (*binary cross entropy*) e tempo di esecuzione

Si osserva che la scelta di utilizzo della *loss* nel contesto della link prediction è stata forzata dal contesto in cui essa viene calcolata. Nessun KG si può definire "completo nella sua interezza", è logico pensare infatti che relazioni o entità siano assenti. In fase di predizione di aggiunta di relazioni, nello stato dell'arte attuale si è per questo motivo costretti ad fare utilizzo della funzione di loss, misura della gravità di errori di predizione prodotti. Nella node classification, d'altro canto lo stato dell'arte predilige l'impiego della accuracy per via dei dataset di benchmark più utilizzati in tali contesto [9].

Notare inoltre, che su consiglio del *Professore Pierfrancesco Bellini*, è stata implementata una funzione *predict()* che permette di calcolare le predizioni su un insieme di entità di test desiderato, e restituisce per ciascuna di essa sia la EventCategory reale che quella predetta.

6.1 Tabelle

Algoritmo	Accuracy	Tempo
RGCN	72.7	43min
DecisionTree	77.9	7min

Tabella 1: Node Classification su km4city

Algoritmo	Loss	Tempo
DQ-GNN	1.13e-03	2h 6min
TuckER	5.44e-05	2h 44min

Tabella 2: Link Prediction su km4city

Come risulta evidente *Decision Tree* ha ottenuto performance migliori e tempi di esecuzione inferiori per quanto concerne il problema della node classification. Mentre, per quanto concerne il task di link prediction, *TuckER* ha ottenuto performance migliori ma tempi di esecuzioni leggermente più lunghi in media. Si nota inoltre come il task di link prediction richieda in linea generale, almeno in riferimento ai 4 modelli considerati, un tempo di esecuzione più lungo.

Conclusioni

In conclusione ci riteniamo davvero soddisfatti del lavoro svolto. Dati i risultati ottenuti, ci sentiamo da affermare che l'impiego delle GNN in questo settore è per il momento eccessivamente complessa o poco performante, tuttavia la ricerca è in continuo sviluppo ed è plausibile che in futuro tutto ciò possa cambiare. É stato interessante documentarsi su alcuni degli innumerevoli modelli presenti nella stato dell'arte attuale.

Riferimenti bibliografici

[1] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[2] Dai Quoc Nguyen, Vinh Tong, Dinh Phung, and Dat Quoc Nguyen. Node co-occurrence based graph neural networks for knowledge graph link prediction. In *Proceedings of WSDM 2022 (Demonstrations)*, 2022.

[3] Gilles Vandewiele, Bram Steenwinckel, Femke Ongenae, and Filip De Turck. Inducing a decision tree with discriminative paths to classify entities in a knowledge graph. In *SEPDA2019, the 4th International Workshop on Semantics-Powered Data Mining and Analytics*, pages 1–6, 2019.

[4] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021.

[5] DISIT. Km4city: Smart city model and tools, 2015.

[6] Giuseppe Futia. Improving knowledge graph embeddings with graph neural networks, 2020.

[7] Thiviyan Thanapalasingam, Lucas van Berkel, Peter Bloem, and Paul Groth. Relational graph convolutional networks: A closer look, 2021.

[8] Dai Quoc Nguyen, Vinh Tong, Dinh Phung, and Dat Quoc Nguyen. Node co-occurrence based graph neural networks for knowledge graph link prediction. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 1589–1592, 2022.

[9] State of the art dataset for node classification.