ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

APPRENDIMENTO NON SUPERVISIONATO DI RAPPRESENTAZIONI E LEGAMI DI SIMILARITÀ TRA EVENTI MENZIONATI NELLA LETTERATURA BIOMEDICA

Elaborato in
Programmazione di applicazioni data intensive

Relatore
Prof. Gianluca Moro

Co-relatori
Dott. Giacomo Frisoni

Presentata da
Eleonora Bertoni

Terza Sessione di Laurea
Anno Accademico 2020 – 2021

PAROLE CHIAVE

Event Embedding

Graph Representation Learning

Graph Similarity Learning

Graph Kernels

Biomedical Text Mining

Noi siamo come nani sulle spalle di giganti, così che possiamo vedere più cose di loro e più lontane, non certo per l'acume della vista o l'altezza del nostro corpo, ma perché siamo sollevati e portati in alto dalla statura dei giganti.

Bernardo di Chartres

Sommario

L'estrazione automatica degli eventi biomedici dalla letteratura scientifica ha catturato un forte interesse nel corso degli ultimi anni, dimostrandosi in grado di riconoscere interazioni complesse e semanticamente ricche espresse all'interno del testo. Purtroppo però, esistono davvero pochi lavori focalizzati sull'apprendimento di embedding o di metriche di similarità per i grafi evento. Questa lacuna lascia le relazioni biologiche scollegate, impedendo l'applicazione di tecniche di machine learning che potrebbero dare un importante contributo al progresso scientifico. Approfittando dei vantaggi delle recenti soluzioni di deep graph kernel e dei language model preaddestrati, proponiamo Deep Divergence Event Graph Kernels (DDEGK), un metodo non supervisionato e induttivo in grado di mappare gli eventi all'interno di uno spazio vettoriale, preservando le loro similarità semantiche e strutturali. Diversamente da molti altri sistemi, DDEGK lavora a livello di grafo e non richiede nè etichette e feature specifiche per un determinato task, nè corrispondenze note tra i nodi. A questo scopo, la nostra soluzione mette a confronto gli eventi con un piccolo gruppo di eventi prototipo, addestra delle reti di cross-graph attention per andare a individuare i legami di similarità tra le coppie di nodi (rafforzando l'interpretabilità), e impiega dei modelli basati su transformer per la codifica degli attributi continui. Sono stati fatti ampi esperimenti su dieci dataset biomedici. Mostriamo che le nostre rappresentazioni possono essere utilizzate in modo efficace in task quali la classificazione di grafi, clustering e visualizzazione e che, allo stesso tempo, sono in grado di semplificare il task di semantic textual similarity. Risultati empirici dimostrano che DDEGK supera significativamente gli altri modelli che attualmente detengono lo stato dell'arte.

Introduzione

I progressi sia nei metodi di ricerca che nei metodi computazionali agiscono come un catalizzatore per un numero sempre maggiore di pubblicazioni scientifiche, specialmente nel dominio biomedico [1]. In questo contesto identificare e tracciare le scoperte e le teorie disseminate all'interno dei paper biomedici risulta quindi un aspetto critico per accelerare il progresso in ambito medico. Purtroppo, stare al passo con tutti i più recenti articoli sta diventando una sfida sempre più difficile per gli esperti. Mentre la letteratura in ambito biologico cresce esponenzialmente sotto forma di documenti testuali, l'estrazione automatica di entità biomediche (proteine, geni, malattie, farmaci...) e delle relazioni semantiche presenti tra di esse sta divenendo sempre più oggetto di studio [2, 3]. I recenti sviluppi relativi a deep learning e natural language processing (NLP) hanno dato vita a metodi intelligenti che hanno permesso di ricavare la conoscenza strutturata, concisa e priva di ambiguità menzionata in testi lunghi e non strutturati. Quando parliamo di text mining in ambito biomedico sappiamo molto bene come entità con classe e nome e le relazioni tra coppie di esse non siano sufficienti per comprendere appieno interazioni che variano dal livello molecolare fino a quello dell'organismo [4, 5]. Di conseguenza, l'estrazione da testo di relazioni complesse (e quindi di eventi) che coinvolgono strutture innestate e molteplici partecipanti è diventato uno dei task per eccellenza tra quelli fortemente legati all'analisi semantica. Secondo le competizioni BioNLP-ST [6, 7, 8], gli eventi sono composti da un trigger (parte del testo che indica chiaramente il verificarsi dell'evento, sono parole come "interagisce", "regola"), un tipo (ad esempio "legame", "regolarizzazione") e un insieme di argomenti con un ruolo specifico che possono essere entità o eventi a loro volta. Seguendo le orme delle scoperte in ambito NLP, gli eventi sono alla base di numerose e utili applicazioni come la scoperta di conoscenza basata sulla letteratura [9, 10], la costruzione di reti biologiche [11], i task di pathway curation [12], diagnosis prediction [13] e question answering [14].

Come molti altri dati relazionali gli eventi possono essere convenientemente e in modo naturale modellati come grafi (diretti, aciclici e con etichette) [15, 3, 16]. Come si vede dalla Figura 1, nella conversione da evento a grafo abbiamo che i nodi sono dati dal trigger (nodo sorgente) e dalle entità mentre gli archi

sono i collegamenti tra un argomento e il rispettivo trigger con un'etichetta che ne indica il ruolo. Inoltre i grafi sono uno strumento di modellazione estremamente diffuso e con solide basi matematiche. È interessante notare che l'applicazione di reti neurali profonde su dati non euclidei strutturati in forma grafo sia un trend emergente, vi è infatti un successo incredibile verso questa tendenza a lavorare sempre di più sul combinare reti neurali e grafi [17, 18]. Un'efficace graph analytics fornisce ai ricercatori una più profonda comprensione dei dati e allo stesso tempo è di supporto a task in diversi domini quali bioinformatica, chemioinformatica, neuroscienze e sociologia. Tra i task citiamo protein folding [19], sistemi di recommendation [20], analisi di social network [21], studi molecolari [22] e progettazione di nuovi farmaci [23], tutti problemi complessi per cui c'è un forte interesse.

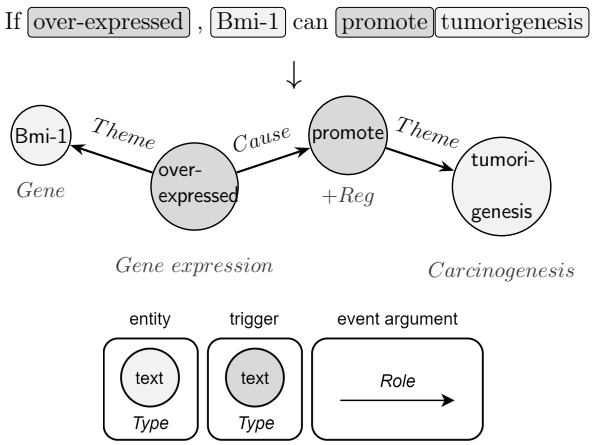


Figura 1: Un grafo evento biomedico estratto a partire da una frase. L'esempio mostra due eventi innestati: (i) un evento di `Positive regulation` (+Reg), legato al trigger "promote", che riguarda una "tumorigenesis" con argomento *Theme*; (ii) una `Gene expression` di "Bmi-1" che ha come argomento (i) *Cause*.

Mettendo da parte i task tradizionali che vengono solitamente affrontati grazie alla combinazione di reti neurali e grafi come link prediction, classificazione di nodi, archi o sottografi, uno dei problemi fondamentali è quello di fare retrieval di grafi simili da un database a fronte di una specifica query. Matching, clustering, ranking e retrieval di grafi evento espressi nella letteratura biomedica possono semplificare significativamente il lavoro degli scienzati aiutandoli soprattutto nello stare al passo con i più recenti sviluppi e nell'identificazione degli studi rilevanti. Permetteranno, per esempio, di cercare interazioni simili a una fornita in input oppure di selezionare eventi che coinvolgono specifiche entità biomediche o determinati tipi di partecipanti con uno specifico ruolo. Per un medico potrebbero essere utili non solo per cercare conferme sull'esistenza

di legami tra una specifica proteina e una malattia all'interno di un paper ma anche per scoprire interazioni tra la stessa malattia e proteine nuove, che non erano state prese in considerazione prima. Inoltre renderanno possibile l'aggregazione e la quantificazione di concetti tra loro fortemente legati andando a osservare i cluster di grafi. Nell'ambito medico questi cluster possono essere manifestazioni di un sintomo e aggregandoli ci si può trovare di fronte allo stesso concetto che può essere stato espresso in modo più dettagliato oppure con parole diverse. Grazie a questi raggruppamenti è anche possibile esprimere il numero di volte che, almeno in termini di ordine di grandezza, una determinata interazione è stata espressa e di conseguenza quanto è alto il riscontro all'interno della letteratura. Per risolvere questo task è necessario trovare un metodo che ci permetta di definire quanto due grafi siano simili. In passato si faceva affidamento su metriche basate su nozioni di distanza predefinite e su specifiche feature: ad esempio alcune tecniche determinavano quanto un grafo si discostasse da un altro contando il numero di operazioni di aggiunta/eliminazione di nodi/archi e modifiche alle label che servivano per passare da un grafo all'altro. Questi approcci presentavano però dei forti limiti perchè erano operazioni davvero costose da calcolare e, nella pratica, difficilmente generalizzabili in quanto comportavano che la definizione di similarità fosse data in anticipo e venisse quindi fatta una forte approssimazione [24, 25]. Per questi motivi, negli ultimi anni la metrica non viene più definita in anticipo ma essa diventa l'obiettivo di predizione [26]. Quando parliamo di Deep Graph Similarity Learning (GSL) significa che addestriamo una rete deep affinchè apprenda in automatico una metrica per misurare i punteggi di similarità tra coppie di grafi. L'idea chiave è quella di allenare un modello che mappi i grafi in input all'interno di uno spazio vettoriale target in modo tale che la distanza nello spazio target approssimi la distanza tra le rappresentazioni simboliche nello spazio di partenza (Figura 2). Notiamo come il Deep Graph Similarity Learning sia strettamente legato al Graph Representation Learning (GRL). Proiettando i grafi in spazi vettoriali continui a bassa dimensionalità preservandone le proprietà intrinseche, gli embedding forniscono uno strumento per memorizzare e accedere alla conoscenza relazionale in modo efficiente sia in termini di tempo che di spazio [27]. Essi possono inoltre favorire la risoluzione di molti problemi di machine learning sui grafi tramite framework standard adatti a lavorare con rappresentazioni vettoriali.
Nonostante il loro potenziale i contributi che GSL e GRL hanno dato all'interno della sfera degli eventi sono tuttora scarsi e limitati. Questo è dovuto al fatto che rappresentare eventi come grafi porta con sè anche dei problemi. Per prima cosa gli eventi sono oggetti intrinsecamente discreti e con una *struttura fortemente irregolare* che può cambiare significativamente da istanza a istanza. Questo ci porta ad affrontare una situazione molto più complicata rispetto a

lavorare solo con triplette. Il secondo motivo è che *non vi sono dataset che riportino score di similarità tra coppie di eventi* e ciò impedisce di sfruttare approcci supervisionati in quanto costruire questi dataset costerebbe troppo in termini di tempo e risorse. Inoltre, i lavori esistenti non riescono a utilizzare in modo efficace coppie di grafi non etichettati per l'apprendimento della metrica. La terza ragione è che i sistemi dovrebbero essere *interpretabili* nel processo di predizione di similarità tra grafi; in un dominio come quello biomedico è un requisito che non può essere ignorato: non ci si può accontentare solo di un punteggio di similarità, è fondamentale sapere esattamente perchè è stato ottenuto. Il quarto motivo è che spesso, quando lavoriamo con grafi etichettati [28, 29], sottili differenze nelle label possono portare due eventi a essere semanticamente coincidenti oppure no, mentre eventi con struttura differente possono ancora risultare simili. Detto questo, è chiaro che una soluzione di successo debba tenere conto *sia della struttura sia della semantica dei grafi.* Quinto, molti metodi si basano su una corrispondenza nota tra i nodi di due grafi ma questo non si può fare quando si lavora con gli eventi dove le etichette dei nodi sono porzioni di testo senza vincoli e sono ammessi più modi per riferirsi alla stessa entità o relazione (ad esempio non possiamo utilizzare *id condivisi*). Un altro problema, strettamente legato al GRL, è che la maggior parte dei lavori si concentra nel costruire embedding a livello dei nodi, degli archi oppure andando a individuare delle comunità di nodi all'interno di un unico e grande grafo considerato in input. Gli eventi estratti da un corpus, invece, corrispondono a numerosi grafi indipendenti e di piccole dimensioni che richiedono un embedding che prenda in considerazione *l'intero grafo.* Il settimo problema che prendiamo in considerazione è che il nostro obiettivo è quello di ottenere embedding *generali* ma accade spesso che gli embedding non siano l'obiettivo della rete ma piuttosto una rappresentazione intermedia che la rete fa per massimizzare l'accuratezza su un task per il quale viene addestrata. Questo porta a degli embedding che presentano un profondo bias e che quindi risultano diversi in base al task sul quale viene fatto l'addestramento. Infine, non vogliamo essere vincolati a un gruppo di grafi noto in anticipo preferendo quindi gli approcci di embedding *induttivi.*

All'interno di questa tesi si affronta il problema dell'applicazione di similarity e representation learning ai grafi evento nell'ambito biomedico. In particolare, si propone un modello non supervisionato[1] per calcolare, tramite deep graph kernel, rappresentazioni di grafi evento che siano in grado di risolvere i problemi elencati in precedenza. Il lavoro si basa su DDGK[30] che è in grado di ricavare una funzione kernel partendo dalle divergenze strutturali e semantiche

[1] Con il termine "non supervisionato", intendiamo quei modelli che si basano esclusivamente sulle informazioni rese disponibili dai grafi evento (struttura, feature di nodi e archi), senza utilizzare label specifiche per il task (come punteggi di similarità) oppure funzioni di loss.

presenti tra coppie di grafi senza il bisogno di feature specifiche, di etichette di similarità o di fare assunzioni sulla struttura o in base al dominio in cui lavoriamo. Inoltre include un meccanismo di cross-graph attention per allineare probabilisticamente le rappresentazioni dei nodi e questo aiuta a comprendere la natura dei punteggi di similarità perchè punta a trovare sottostrutture simili. Guidati dalle caratteristiche degli eventi nell'ambito NLP, forniamo un'estensione di DDGK, chiamata Deep Divergence Event Graph Kernels (DDEGK), in grado di catturare, grazie all'integrazione di language model preaddestrati, le congruenze semantiche tra etichette continue.

Si riassumono i maggiori contributi:

- **Analisi dettagliata della letteratura**. Si offre una panoramica del problema e vengono descritte le maggiori tipologie di soluzioni adottate e quelle con cui ci siamo andati a confrontare.
- **Deep Divergence Event Graph Kernel**. Un nuovo metodo per l'apprendimento della similarità tra grafi evento che si focalizza sulla costruzione di embedding generali per mezzo di deep graph kernels e che tiene conto sia della struttura che della semantica degli eventi.
- **Risultati degli esperimenti**. Gli esperimenti condotti per dimostrare l'efficacia di DDEGK applicato a casi di studio. Si mostra come la nostra soluzione sia in grado di individuare le similarità, sia a grana fine che più grossolane, presenti tra eventi nell'ambito biomedico. In particolare, gli embedding ottenuti da DDEGK, se utilizzati come feature ottengono risultati competitivi in diversi task quali sentence similarity, event classification e clustering. Per poter apprezzare meglio la soluzione ci confrontiamo con tecniche di embedding differenti andando a testare le prestazioni su nove dataset di eventi provenienti da differenti rami della biologia.

La tesi è così organizzata:

- nel Capitolo 1 viene fornita una panoramica sull'ambito di ricerca, vengono presentati i lavori correlati andando a esaminare le soluzioni adottate e quelle con cui ci siamo direttamente confrontati;
- nel Capitolo 2 sono introdotte le nozioni e i concetti necessari per comprendere il lavoro svolto. Si entra nel dettaglio della soluzione DDEGK partendo da DDGK per poi passare alla spiegazione delle modifiche introdotte;
- nel Capitolo 3 tramite esempi di codice viene mostrato il lavoro svolto e quanto appreso dal punto di vista dell'implementazione;

- nel Capitolo 4 sono elencati gli esperimenti effettuati e sono illustrati i risultati ottenuti dalla nostra soluzione confrontadoli con quelli ottenuti dai metodi della baseline;
- infine nel Capitolo 5 si traggono le conclusioni finali.

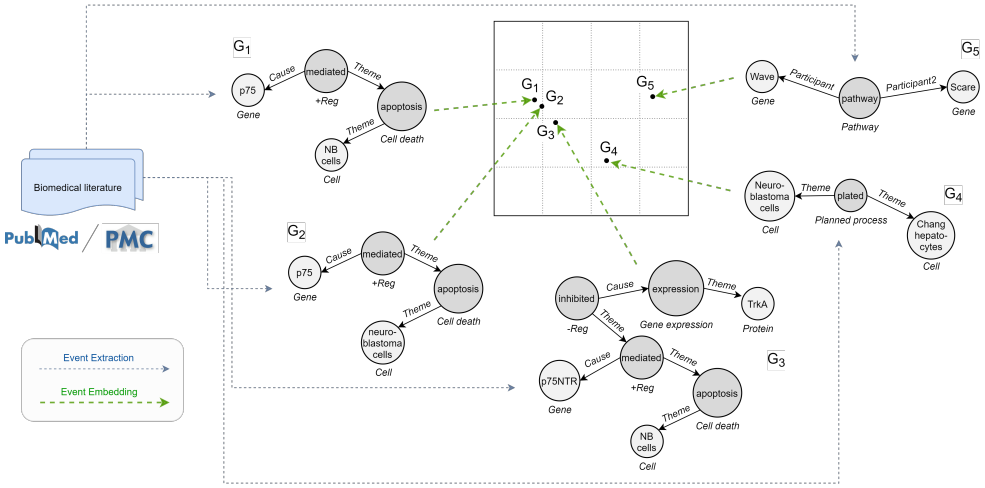


Figura 2: Esempio di embedding con conservazione della similarità tra i grafi evento, gli eventi campione sono presi dal dataset CG13. Ciascun evento è mappato in un vettore di embedding (indicato da un puntino nella rappresentazione in 2D). La funzione obiettivo dovrebbe considerare sia la struttura che la semantica. Si considerano alcuni casi interessanti: (i) grafi con la stessa struttura e, nel complesso, stessa semantica (G_1 e G_2); (ii) grafi che differiscono per la struttura ma simili per la semantica (G_2 e G_3); (iii) grafi con la stessa struttura ma semantica differente (G_4 e G_5).

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

In questo capitolo si va a posizionare accuratamente il nostro lavoro, delineandone l'ambito di applicazione, confrontandolo con metodi differenti e prendendo in considerazione gli studi su argomenti correlati. L'obiettivo è quello di fornire una panoramica sulla ricerca fatta fino ad oggi mettendo in evidenza quali siano i problemi da risolvere che hanno portato alla realizzazione di questo lavoro. Nella Figura 1.1, schematizziamo gli studi che ruotano attorno a DDEGK.

1.1 Individual Graph Embedding

Lavorare con dati relazionali richiede significative risorse computazionali, esperienza nel settore e feature dipendenti dal task considerato che permettano di integrare informazioni fondamentali all'interno di modelli di predizione. Fare embedding di grafi risolve efficacemente il problema degli elevati costi computazionali e di spazio propri della Graph Analytics, combinando sia i benefici degli approcci strutturali che di quelli statistici.

Al giorno d'oggi esistono tantissimi metodi per fare embedding di grafi che mettono a disposizione potenti strumenti per costruire spazi vettoriali di feature per i grafi e le loro componenti. I metodi di Graph similarity basati sugli embedding di grafi cercano di utilizzare le rappresentazioni apprese per predire punteggi di similarità. Di seguito, riassumiamo gli approcci dove gli embedding vengono appresi in modo indipendente tra grafi, in una fase che precede il calcolo della similarità. Divideremo l'analisi in base al tipo di embedding restituito come output (granularità), all'approccio metodologico e alla natura dei grafi di input.

Dal momento che la maggior parte dei modelli fa riferimento a specifici casi d'uso di grafi, faremo chiarezza sull'impostazione del problema. Per essere precisi, gli eventi sono grafi eterogenei e non pesati che contengono molte

informazioni ausiliarie. Nodi e archi (e quindi trigger/entità e ruoli coperti dagli argomenti) possono appartenere a diversi tipi, definiti all'interno di uno schema ontologico[1] che fungono da etichette categoriche. Inoltre, ciascun nodo ha un attributo testuale con un valore continuo che esprime lo span di testo che ne ha permesso l'estrazione. In modo più intuitivo possiamo dire che le etichette dei nodi/archi e gli attributi testuali forniscono maggiori dettagli sul grafo evento e questo potenzialmente può portare a embedding migliori: mettono in correlazione istanze vicine e supportano l'apprendimento induttivo permettendo una migliore generalizzazione verso esempi o domini mai visti prima.

1.1.1 Node Embedding

La maggior parte della ricerca su Graph Representation Learning si concentra su embedding a livello di nodo.

Matrix Factorization. I primi modelli utilizzati per fare embedding di grafi sono quelli basati sulla fattorizzazione di matrici. Essi rappresentano le proprietà di un grafo, quali le adiacenze o le somiglianze tra coppie di nodi, attraverso una grande matrice che provano ad approssimare mediante fattorizzazione di matrici a basso rango (ad esempio mediante singular value decomposition) andando quindi a scoprire rappresentazioni latenti dei nodi[31, 32, 33, 34, 35]. Anche se vi sono alcune varianti, in generale questi modelli corrispondono a un processo di riduzione della dimensionalità che punta a conservare le proprietà strutturali. Diversamente dal nostro metodo, sebbene siano non supervisionati e trasparenti dal punto di vista matematico, queste soluzioni spesso ignorano la semantica e conservano solo i vicini del primo ordine (similarità locale tra coppie di nodi connessi da archi) [36]. Inoltre, sia la costruzione di matrici sia la loro decomposizione basata sugli autovalori risultano costose in termini di tempo e spazio e ciò rende le operazioni di fattorizzazione di matrici inefficienti e non scalabili per grandi grafi [37] (ma non è questo il caso quando si lavora con gli eventi).

Deep Learning. Il Deep Learning ha mostrato risultati promettenti tra i diversi metodi di embedding grazie all'identificazione automatica di rappresentazioni adatte ottenute considerando complesse strutture a grafo come problemi di ottimizzazione. Questo modo di lavorare punta alla realizzazione di funzioni di costo che catturino le dipendenze e le similarità intra-grafo riuscendo, allo stesso tempo, a mantenere risultati di alta qualità nei task presi in considerazione e a costruire embedding di grafi rispettando vincoli di efficienza. Dividiamo i contributi di Deep Learning in due categorie a seconda che l'input considerato

[1] Il termine "evento" viene utilizzato per indicare interazioni complesse che sono il risultato di un task di event extraction a dominio chiuso dove i tipi target sono predefiniti

sia costituito da una serie di cammini casuali (e quindi sequenze di nodi) o grafi nella loro interezza.
Deep Learning con Random Walks. Gli embedding sequence-based sono costruiti basandosi sull'idea fondamentale di approssimare linearmente i grafi. Deep Walk [36] rappresenta il primo tentativo di generalizzazione dei modelli skip-gram di NLP con dati strutturati in forma grafo e che va quindi a tracciare le analogie tra linguaggio e grafi. In questa ottica, ciascun percorso ottenuto a partire da un grafo corrisponde a una frase e un nodo corrisponde a una parola. Entrando più nel dettaglio, DeepWalk effettua diversi cammini casuali a partire da ciascun nodo e addestra delle reti neurali massimizzando la probabilità di predire il contesto dei nodi (struttura del vicinato locale) influenzato dall'embedding del nodo stesso e codificando le ricorrenze presenti all'interno di piccole sotto-finestre. Molti studi ottimizzano, modificano o estendono questa idea, alcuni importanti esempi sono LINE [38], node2vec [39], GenVector [40], metapath2vec [41], HARP [42], diff2vec [43], WYS [44], e GEMSEC [45]. I metodi basati su DeepWalk sono non supervisionati e sono in grado di conservare i vicini fino al secondo ordine. Sono in grado di catturare relazioni anche a lunga distanza considerando vicini quei nodi aventi simili contesti/vicinati anche se non connessi direttamente. Fare sampling permette di esplorare e semplificare il grafo perchè comporta una mancanza di consapevolezza sulle informazioni a livello globale ma permette di mantenere la computazione trattabile. Quindi è adatto per singole grandi reti contenenti milioni di nodi e miliardi di archi [38], uno scenario in completa contrapposizione con il nostro. Quel che è peggio è che, nonostante vi siano alcune eccezioni[40, 46], molti di questi algoritmi lavorano solamente con la struttura.
Deep Learning senza Random Walks. Questi metodi applicano modelli di Deep learning direttamente a interi grafi. Un approccio comune è quello di utilizzare un *autoencoder*, solitamente ha una rete per l'encoding e per il decoding per modellare le non linearità. Se diamo in input una matrice di adiacenza, l'encoder è in grado di aggregare l'informazione locale a livello di nodo producendo rappresentazioni di grafi compresse, e il decoder minimizza una reconstruction loss ottenuta a partire dagli embedding dei nodi (assicurandosi, come obiettivo non supervisionato, che vengano mantenuti i vicini). Per esempio, GAE e VGAE [47] ricostruiscono la struttura del grafo considerando il prodotto scalare tra gli embedding dei nodi calcolati da una Graph Neural Network (GNN). Le GNN sono proprio un'altra grande famiglia di metodi all'interno di questa classe. Esse puntano all'apprendimento di funzioni differenziabili su grafi che possono avere struttura arbitraria, allo scopo di risolvere task supervisionati (o semi-supervisionati) che generalmente richiedono grandi volumi di dati etichettati per ottenere rappresentazioni significative. Mentre le tecniche più recenti prima mappano i nodi in rappresentazioni vettoriali latenti per poi passarli

come input a un'altra rete neurale, le tecniche per fare embedding di grafi supervisionato combinano le due fasi. Quindi, sulla base delle informazioni contenute nei nodi, le GNN apprendono degli embedding per uno specifico scopo (ad esempio per predire delle proprietà molecolari o la tossicità di un composto medico) e questi non possono essere utilizzati o trasferiti per compiti o problemi differenti. Questi modelli, per come sono progettati, non subiscono variazioni a fronte di permutazioni degli elementi del grafo e calcolano le rappresentazioni dei nodi mediante un processo di propagazione che aggrega iterativamente le informazioni locali. Uno dei vantaggi principali è quello di essere generalizzabili perchè le GNN sono in grado di catturare funzioni applicabili a qualsiasi grafo che sia del tipo supportato e quindi non sono limitate solo a quelli utilizzati come input durante l'addestramento. Tre lavori molto popolari sono graph convolutional networks (GCN) [48], graph attention networks (GAT) [49], e GraphSage [50]. Le GCN sostituiscono lo scambio di messaggi con convoluzioni di grafi. Le GAT usano dei masked layer di self-attention per bilanciare l'impatto che hanno i vicini sull'aggregazione. GraphSage migliora le GCN andando a campionare solo un numero fisso di nodi pesati scelti tra i vicini a diversi livelli di profondità e quindi integrando i cammini casuali. A differenza di questi metodi, DDEGK apprende delle rappresentazioni di eventi non supervisionate e indipendenti dal task per cui verranno utilizzate.
Gli embedding di nodi da soli possono essere utilizzati per calcolare similarità inter-grafo. Per esempio in Nikolentzos et al. [51], i nodi di due grafi vengono proiettati nello spazio euclideo usando gli autovalori delle matrici di adiacenza, rappresentando ciascun grafo come bag-of-vectors. La similarità viene quindi misurata calcolando una corrispondenza sulla base dell' EMD (Earth Mover's Distance) [52] tra due gruppi di embedding. In modo più originale, node2vec-PCA [53] porta gli embedding dei nodi all'interno di uno spazio a dimensioni ridotte e utilizza le similarità latenti che esistono tra nodi per codificare i grafi come colonne di un istogramma 2D; l'immagine artificiale che viene così costruita può quindi essere passata ad una classica architettura 2D CNN per fare un task supervisionato. In ogni caso, nonostante il grande sviluppo nelle tecniche di embedding di nodi, un problema assai noto riguarda la trascuratezza della struttura a livello globale e dei pattern presenti a più alto livello che risultano davvero necessari per poter fare dei confronti. Concludendo, come normalmente richiesto dai piccoli grafi, come proteine e molecole [54], noi puntiamo ad ottenere una singola rappresentazione per ogni grafo/evento biomedico e non per ciascun nodo.

1.1.2 Whole-graph Embedding

Molti lavori si sono concentrati sull'apprendimento supervisionato a livello di nodi, archi e grafi e non supervisionato a livello di nodi (ad esempio clustering a livello di nodi). Al contrario, la ricerca sull'*apprendimento non supervisionato a livello di grafo* fino ad adesso ha ricevuto poca attenzione, nonostante il vasto range di applicazioni pratiche come trovare corrispondenze tra grafi oppure ottenere punteggi di similarità in specifici domini quali biologia e NLP [55]. Fare embedding di un intero grafo richiede di scegliere tra forza espressiva e efficienza andando a introdurre una qualche nozione di pooling per poter aggregare le informazioni provenienti dai livelli sottostanti all'interno di un singolo vettore. Come discusso precedentemente, generalizzare questa idea di pooling per grafi arbitrari è una questione per niente banale a causa della mancanza di regolarità nella struttura dei grafi; questa è quindi un'importante area di ricerca.
Apprendimento supervisionato. Le soluzioni prevalenti prevedono (i) semplicemente di sommare o calcolare la media di tutti gli embedding dei nodi in un layer finale [56, 57, 58], (ii) di aggiungere dei layer per fare pooling di grafi parametrici (alcuni esempi sono dati da DiffPool [59], SortPool [60], TopKPool [61], SAGPool [62]) oppure memory layer (per esempio MemGNN [63]) all'interno delle GNN supervisionate. In modo alternativo, Patchy-San [64] propone un approccio che utilizza operazioni di convoluzione su dati strutturati in forma grafo e che supporta attributi sia discreti che continui su nodi e archi.
Apprendimento non supervisionato. Mettendo da parte le GNN e le tecniche supervisionate, che risultano chiaramente differenti dalla nostra, ci concentriamo ora su alcuni metodi non supervisionati che sono stati proposti [65, 66, 67]. Il modo più intuitivo di generare l'embedding di un grafo è quello di calcolare gli embedding dei nodi per poi applicare una tecnica di flat pooling (aggregazione tramite calcolo di media, somma o massimo) oppure hierarchical pooling. Un'ulteriore possibilità è quella di fare una somma pesata guidata dal grado dei nodi oppure dall'importanza del nodo all'interno del grafo, ricavabile tramite context-aware attention [68]. Un'altra possibilità è quella di introdurre un supernodo virtuale [69]. Tra le soluzioni più avanzate troviamo sub2vec [65] e graph2vec [66]. Sub2vec impara la rappresentazione di qualsiasi sottografo campionando sottostrutture lineari con cammini casuali di lunghezza fissa; tuttavia presenta degli svantaggi in termini di stabilità e perdita di informazioni. Lavorando con sottostrutture non lineari, graph2vec preserva meglio le uguaglianze strutturali e ha raggiunto lo stato dell'arte su diversi dataset. Come node2vec è analogo a word2vec, graph2vec è analogo a doc2vec. Infatti, partendo dall'osservazione che i grafi possono essere visti come insiemi di sottografi allo stesso modo in cui i documenti possono essere pensati come insiemi di parole, gli autori adattano

doc2vec da un contesto di NLP e allenano un modello skip-gram per predire sottografi esistenti all'interno del grafo di input. Però graph2vec è trasduttivo e ciò significa che produce un embedding solo per le istanze conosciute a tempo di addestramento.
Rappresentazioni Statistiche. Un altro ramo di lavori [70, 71, 72, 73] rappresenta i grafi come vettori di feature scelte manualmente, utilizzandole per successivi confronti inter-grafo. Questi lavori puntano all'aggregazione di caratteristiche locali, proprietà statistiche o topologiche (come il grado di un nodo e i suoi vicini) e sono del tutto ignari dal punto di vista globale, pertanto necessitano di mappature nodo-nodo conosciute. In contrasto, DDEGK non progetta esplicitamente le feature e non fa alcun tipo di assunzione sulla loro importanza all'interno del task di applicazione.
Whole-graph embedding rappresenta una soluzione efficiente e diretta per il calcolo di similarità tra grafi, combinando velocità e scalabilità. Presenta comunque delle lacune: la vicinanza grafo-grafo viene ignorata e non vi sono interazioni tra le feature da un grafo all'altro. Di conseguenza, questi modelli potrebbero non risultare adatti per la predizione di similarità tra grafi se messi a confronto con metodi che integrano GRL con GSL. Per risolvere questo problema, DDEGK apprende l'embedding dell'intero grafo confrontando i grafi tra di loro.

1.1.3 Knowledge Graph Embedding

Negli ultimi anni, la ricerca relativa agli embedding di reti si è concentrata molto su grafi speciali come ad esempio i knowledge graph. I knowledge graph sono astrazioni simboliche usate per codificare una base di conoscenza o una raccolta di dichiarazioni, a cui spesso facciamo riferimento usando la parola "fatti", essi hanno la forma di triplette formate da *soggetto-predicato-oggetto* connessi tra loro. Ciò che rende speciali i knowledge graph, ampiamente adottati nei task NLP, è che prendono in considerazione sia nodi che archi etichettati (e quindi numerose relazioni binarie differenti). Dal momento che adattare i metodi descritti nelle sezioni 1.1.1 e 1.1.2 a questo contesto non è semplice, l'embedding di knowledge graph è un problema che è stato ampiamente affrontato da una comunità che va a distaccarsi da quella descritta fino a questo momento. I metodi più utilizzati al momento creano, in modo coerente al grafo, un vettore per ciascuna entità e relazione (creano un embedding ibrido), attraverso un processo di apprendimento focalizzato sulla capacità di distinguere le triplette esatte da quelle negative (corrotte [74]). In particolare, i modelli traslazionali utilizzano la struttura della tripletta solamente per imparare le traslazioni *testa* $\rightarrow$ *coda* (alcuni esempi sono TransE [75], TransH [76], RotatE [77], HAKE [78]). Invece, i modelli bilinerari adottano un approccio differente

andando a rappresentare le relazioni come matrici all'interno di uno spazio vettoriale (alcuni esempi sono RESCAL [79], DistMult [80], HolE [81], ComplEx [82]). Infine abbiamo le reti neurali come le neural tensor network (NTN) [83], RDF2Vec [84], ConvE [85], e R-GCNs [86]. Alcuni ricercatori si spingono oltre, andando a integrare informazioni aggiuntive come descrizioni testuali, per arricchire dal punto di vista semantico le rappresentazioni dei knowledge graph [87, 88, 89]. Mentre ci sono tantissimi lavori relativi all'embedding di knowledge graph, ovvero su strutture relazionali binarie, non si può dire lo stesso sugli embedding di relazioni più complesse [90], come gli eventi. Ovviamente, un approccio sensato per fare embedding di relazioni n-arie potrebbe essere quello di suddividerle in strutture di incidenza binarie (insiemi di coppie di archi). D'altro canto, gli eventi esprimono singoli concetti complessi, e vogliamo sottolineare che singole triplette al loro interno potrebbero generare fatti incompleti o addirittura sbagliati. Inoltre, gli eventi collegano in modo diretto coppie trigger-entità o trigger-trigger ma non entità-entità. Di conseguenza, a differenza di questi lavori, la nostra soluzione affronta il problema del calcolo degli embedding di fatti relazionali, n-ari e indipendenti piuttosto che di archi contenuti all'interno di un unico e grande grafo sorgente.

1.2 Calcolo di Graph Similarity con Cross-Graph Feature Interaction

Questi metodi prendono una coppia di grafi come input e calcolano un punteggio di similarità tra di essi. A differenza dei metodi descritti nella 1.1 l'abbinamento viene fatto congiuntamente sulla coppia, piuttosto che andando a mappare ciascun grafo in un vettore in modo indipendente. Di conseguenza, questi modelli sono potenzialmente più forti di quelli standard al costo di qualche computazione aggiuntiva. Vogliamo inoltre sottolineare che, in numerosi casi, è più facile specificare una ragionevole funzione di dissimilarità o similarità tra istanze piuttosto che andare a costruire una rappresentazione delle feature per l'intero input strutturato. [91].

1.2.1 Metodi di Direct Comparison

Agli inizi, sono state proposte delle metriche per misurare la similarità tra grafi, queste si basavano su nozioni di distanza concrete e definite a priori, come Graph Edit Distance (GED) [92] oppure la dimensione del più grande sottografo in comune [93]. Sfortunatamente il calcolo di queste metriche risulta NP-completo nel caso generale [24], e questo implica costi in termini di tempo esponenziali che possono diventare insostenibili anche solo per istanze aventi

più di 16 nodi[25]. Data la significativa difficoltà nel calcolare in modo esatto la distanza tra grafi, strategie di potatura [24, 94] e metodi euristici [95, 96, 97] vengono utilizzati regolarmente per ridurre la complessità computazionale a un grado gestibile. Inoltre, GED considera tutte le operazioni di modifica allo stesso modo, senza valutare in che misura esse vanno a modificare la topologia e la semantica del grafo, e rimane quindi un metodo non adatto per confrontare grafi di dimensioni diverse. A differenza di questi approcci, il nostro metodo evita volutamente di introdurre queste metriche predefinite.

1.2.2 Graph Similarity learning basato su GNN

Questa famiglia di tecniche per fare graph similarity learning usa le GNN per apprendere contemporaneamente sia le rappresentazioni che i punteggi di similarità tra grafi. Presa una coppia di grafi in input $\langle G_i, G_j, y_{ij} \rangle$, dove y_{ij} indica il punteggio di similarità della coppia $\langle G_i, G_j \rangle$, queste tecniche prima impiegano delle GNN con diversi layer per calcolare gli embedding di G_i e G_j in cui ciascun grafo può influenzare gli altri grazie a un qualche meccanismo di condivisione dei pesi o di interazioni cross-graph. La predizione del punteggio di similarità tra le rappresentazioni vettoriali o matriciali dei due grafi restituite dalle GNN viene calcolata utilizzando un layer o più fully-connected layer per il calcolo del prodotto scalare. I punteggi di similarità calcolati per tutte le coppie di grafi e per le etichette sono poi messi a confronto con una funzione di loss per gli aggiornamenti del gradiente.
GNN-CNN. Le reti miste GNN-CNN (come ad esempio GSimCNN [98], SimGNN [68]) utilizzano le GNN per l'apprendimento delle rappresentazioni dei grafi, queste vengono poi sfruttate dalle CNN per predire i punteggi di similarità, spostando il problema sulla classificazione o regressione.
Siamese GNN. I modelli Siamese GNN (come S-GCN [99], HS-GCN [100], MatchGNet [101], UGRAPHEMB [102]) sono formati da due reti GNN gemelle con parametri condivisi, applicate in modo indipendente a due grafi di input per produrre rappresentazioni del grafo che verranno fuse da una piccola rete per poter fare predizione di punteggi di similarità. Infine il punteggio calcolato viene utilizzato all'interno di una funzione di loss per allenare l'intero modello.
Graph Matching Networks. I lavori all'interno di questa categoria (ad esempio GMN[28]) adattano le GNN siamesi introducendo dei meccanismi di corrispondenza e interazioni cross-graph durante il processo di GRL condotto dalle due GNN.
Molti punteggi di distanza tra coppie di grafi devono essere etichettati in anticipo per poter fare il training di tutti questi metodi. Invece, nel nostro lavoro non vengono utilizzate etichette e questa scelta è fortemente influenzata dal fatto che non vi siano dataset per eventi biomedici e eventi in generali.

1.2.3 Graph Kernels

I Graph Kernel valutano la similarità (valore del kernel) tra coppie di grafi scomponendoli ricorsivamente in sottostrutture atomiche su cui si va a definire una funzione di similarità (funzione kernel). Tradizionalmente i graph kernel usavano feature scelte manualmente sulla base della teoria dei grafi come i cammini casuali o i cammini minimi. Per esempio i kernel basati sui cammini minimi[103] mettono a confronto le lunghezze di tutti i cammnini minimi tra i nodi di due grafi. I Graphlet kernel [104] contano il numero di sottografi con 3,4 nodi (motivi). I Weisfeiler-Lehman kernel [105] propongono di aggregare le informazioni discrete o continue dei nodi attraverso un metodo analogo al color refinement utilizzato per capire se due grafi sono isomorfi. I Multi-scale Laplacian kernel [106] vanno a confrontare i grafi su scale differenti individuando relazioni topologiche tra nodi e associazioni di sottografi. Bisogna tenere conto, però, che i kernel definiti in modo esplicito sono limitati da problemi come il fatto che le rappresentazioni abbiano alta dimensionalità, siano sparse e non regolari e questo porta a una scarsa generalizzazione [107]. I modelli basati su deep graph kernel sono emersi solo di recente andando a sostituire feature selezionate manualmente con quelle apprese in modo automatico dai dati grazie alle reti neurali deep. Secondo un recente studio su GSL [26] DDGK [30] è l'unico contributo ad essere in grado di gestire grafi eterogenei e con attributi, che supporta interazioni cross-graph e che ha già avuto delle applicazioni in bioinformatica e in chemioinformatica. Il nostro lavoro si colloca all'interno di questo gruppo ma si concentra nell'ottenere embedding di grafi.

1.3 Event Embedding

Gli eventi sono la chiave per svelare le conoscenze fondamentali dal mondo reale e catturare i processi biologici descritti all'interno di testo libero, non strutturato. Purtroppo, i numerosi contributi sul fronte dell'event extraction [108] sono contrastati dall'esiguo numero di lavori sull'apprendimento di rappresentazioni di eventi di cui c'è sempre più bisogno. Fare embedding di eventi risulta essere un'efficiente soluzione per rappresentare eventi discreti e sparsi come vettori densi in uno spazio continuo, riflettendo le similarità presenti tra di essi e rendendo possibili numerose applicazioni. Il problema fondamentale che unisce quasi tutti i metodi esistenti al momento per fare embedding di eventi è che considerano solo le relazioni binarie [109, 110, 111, 112, 113]. Questa semplificazione ritrae un caso davvero poco realistico, decisamente lontano dai complessi concetti che sono l'obiettivo di alcune famose competizioni quali BioNLP-STs e ACE2005. Dato il ruolo fondamentale della semantica all'interno degli eventi, molti di questi metodi [109, 110, 111] vedono gli eventi come tuple

soggetto-predicato-oggetto e impiegano le neural tensors network tipicamente usate per l'embedding dei knowledge graph. Per mezzo delle neural tensors network gli autori sono in grado di apprendere le relazioni tra un predicato e il suo soggetto/oggetto, generando embedding di nodi e archi grazie all'utilizzo di tuple corrotte per l'addestramento e andando, eventualmente, a integrare conoscenze da fonti esterne [110, 112]. Karumba [113] indaga sull'apprendimento supervisionato di eventi grazie a strutture gerarchiche all'interno di uno spazio iperbolico. DeepEventMine [3], la soluzione che al momento è allo stato dell'arte per l'estrazione di eventi biomedici end-to-end, sfrutta le rappresentazioni di eventi ottenute come concatenazione degli embedding calcolati da un modello BERT, e ignora quindi la struttura. Degno di nota, Gui et al. [114] non scompone l'interazione tra tutte le entità partecipanti in differenti coppie di relazioni sparse e tra loro indipendenti ma impiega una struttura di iperarchi per evitare che ci sia perdita di informazione. Per lo stesso principio, noi codifichiamo più interazioni come un tutt'uno. Il nostro lavoro rappresenta il primo metodo che permette di fare embedding di eventi, a livello di grafo basato su deep graph kernel.

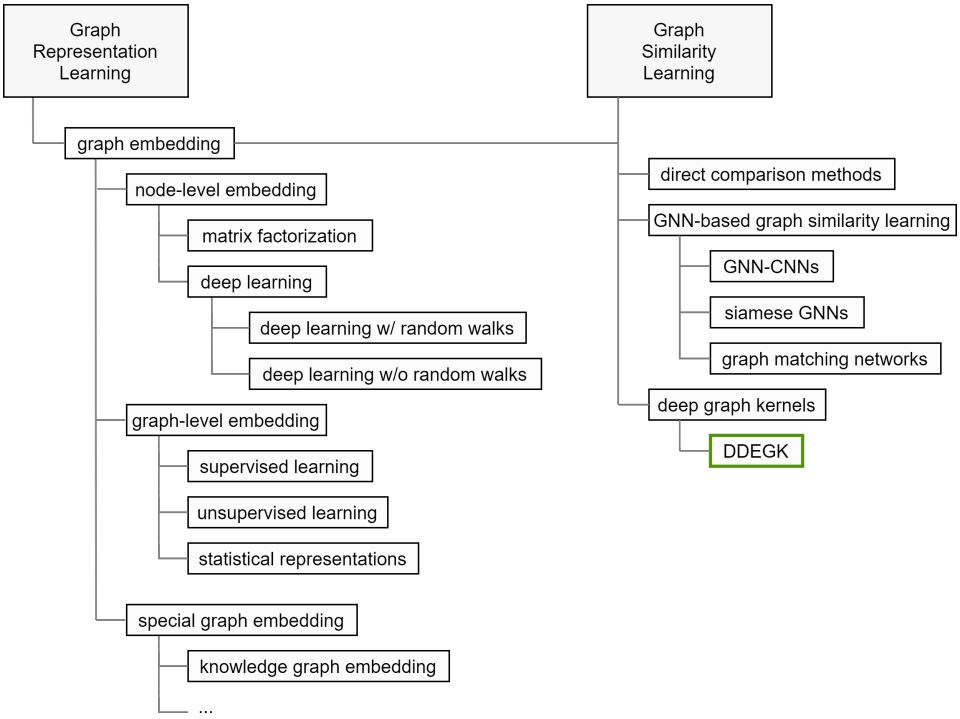


Figura 1.1: Divisione proposta per categorizzare la letteratura su representation graph e similarity learning. In verde evidenziamo dove si colloca DDEGK.

Capitolo 2

DDEGK

2.1 Nozioni teoriche

Prima di entrare nel dettaglio della nostra soluzione riportiamo le nozioni teoriche necessarie e le definizioni dei concetti chiave utilizzate nel resto della tesi.

Definizione 1 (Grafo Evento). Un grafo, indicato come $G = (V, E)$, costituito da un insieme finito di nodi, $V = v_1, \ldots, v_{|V|}$, e un insieme di archi, $E \subseteq V \times V$, $|E| = m$, dove un arco $e_{i,j}$ collega il nodo v_i con il nodo v_j. Nel caso dei grafi evento gli archi sono diretti, non pesati e non presentano cicli. Un nodo rappresenta un trigger o un'entità, mentre un arco modella una relazione entità-trigger o trigger-trigger e nel secondo caso parliamo di eventi innestati. Le connessioni tra nodi vengono rappresentate attraverso una matrice di adiacenza $A \in \mathbb{R}^{|V|\times|V|}$, dove $a_{ij} = 1$ se c'è un collegamento tra i nodi i e j, e 0 altrimenti. Sia i nodi che gli archi di G hanno informazioni sul tipo. Siano $\tau_v : V \to T^v$ una funzione per mappare il tipo di un nodo e $\tau_e : E \to T^e$ una funzione per mappare il tipo di un arco, dove T^v indica l'insieme dei tipi dei nodi (eventi o entità), e T^e l'insieme dei tipi degli archi (e quindi i ruoli che gli argomenti possono assumere). Ciascun nodo $v_i \in V$ ha uno specifico tipo, $\tau_v(v_i) \in T^v$; analogamente per ogni arco e_{ij}, $\tau_e(e_{ij}) \in T^e$. Dal momento che $|T^v| + |T^e| > 2$, i grafi evento costituiscono reti eterogenee. Un grafo evento è inoltre dotato di una funzione $\gamma_v : V \to \Gamma^v$ che assegna informazione sottoforma di testo libero a tutti i nodi. Diciamo quindi che $\gamma_v(v_i)$ è l'attributo continuo di v_i.

Definizione 2 (Isomorfismo di grafi e sottografi). Due grafi non etichettati G_1 e G_2 sono isomorfi, e si indica con $G_1 \simeq G_2$, se esiste una biiezione $\phi : V(G_1) \to V(G_2)$, tale che $(u, v) \in E(G_1)$ se $(\phi(u), \phi(v)) \in E(G_2)$ per tutti (u, v) in $E(G_1)$. Allora ϕ è un isomorfismo. Per i grafi con delle etichette, l'isomorfismo tiene solo nel caso in cui la biiezione è in grado di mappare nodi e archi con la medesima etichetta. Se parliamo di isomorfismo di sottografi stiamo

considerando una generalizzazione del problema dell'isomorfismo di grafi, in cui l'obiettivo è quello di determinare se G_1 contiene un sottografo che risulta isomorfo con G_2. Non esiste alcun algoritmo di complessità non-polinomiale per determinare se i due grafi siano in isomorfismo. Per la precisione, è stato dimostrato che verificare se esiste un isomorfismo tra sottografi è un problema NP-completo mentre lo stesso non si può dire per l'isomorfismo tra grafi che rimane NP.
Definizione 3 (Graph Kernel). Dati due vettori x e y all'interno di uno spazio delle feature $\mathbb{R}^n$, e una funzione di mapping $\varphi : \mathbb{R}^n \to \mathbb{R}^m$, definiamo una funzione kernel come $k(x, y) = \varphi(x)^T\varphi(y)$. In sostanza, la funzione kernel è una funzione di similarità – che soddisfa le condizioni di essere simmetrica e definita positiva – che può essere interpretata come il prodotto scalare tra due vettori dopo che sono stati proiettati all'interno di un nuovo spazio. Quindi, una funzione kernel può essere applicata per calcolare il prodotto scalare di due vettori nello spazio target delle feature (tipicamente ad elevata dimensionalità) senza il bisogno di andare a cercare in modo esplicito uno spazio in cui mappare i vettori. In structure mining, un graph kernel è semplicemente una funzione kernel che calcola un prodotto scalare su coppie di grafi, andando tipicamente a confrontare sottostrutture locali [115].
Definizione 4 (Whole Graph Representation Learning). Whole Graph Representation Learning punta a cercare una funzione Ψ che mappi un grafo discreto G in un vettore continuo $\Psi(G) \in \mathbb{R}^d$, riuscendo a mantenere le importanti proprietà del grafo. Se d è piccolo, parliamo di graph embedding. Il graph embedding può essere visto come una tecnica di riduzione della dimensionalità per dati strutturati in forma grafo, dove l'input è definito su un dominio non Euclideo, ad alta dimensionalità e discreto.
Definizione 5 (Graph Similarity Learning). Sia $\mathcal{G}$ un insieme di grafi di input, $\mathcal{G} = G_1, G_2, \ldots, G_n$, il graph similarity learning punta a trovare una funzione $\mathcal{S} : (G_i, G_j) \to \mathbb{R}$, che restituisca un punteggio di similarità s_{ij} per ciascuna coppia di grafi$(G_i, G_j) \in \mathcal{G}$.

2.2 Deep Divergence Event Graph Kernels

In questa sezione viene descritto il funzionamento di DDEGK. Ci saranno dei richiami al metodo originale DDGK e si andranno a sottolineare i nostri cambiamenti, inclusa la nuova funzione di loss e le specifiche modifiche necessarie per poter gestire gli eventi biomedici.

2.2.1 Definizione del problema

Data una famiglia di N grafi evento $\mathcal{T} = G_1, G_2, \ldots, G_N$, il nostro obiettivo è quello di costruire un embedding per ogni grafo evento G_i (che chiameremo grafo evento *target*), basandoci sulla sua distanza rispetto a un insieme di M grafi evento *prototipo* ($\mathcal{A}$). Per fare ciò, puntiamo a imparare una funzione graph kernel, definita nel seguente modo:

$$k(G_1, G_2) = ||\Psi(G_1) - \Psi(G_2)||^2, \tag{2.1}$$

dove la rappresentazione $\Psi(G \in \mathcal{T}) \in \mathbb{R}^M$. Nello specifico, per ogni elemento di $\mathcal{T}$, definiamo la dimensione i^{esima} della rappresentazione come:

$$\Psi(G)_i = \sum_{v_j \in V(G)} f_{a_i}(v_j), \tag{2.2}$$

dove $a_i \in \mathcal{A}$ e $f_{a_i}()$ è una funzione di predizione per qualche proprietà strutturale e semantica del grafo G parametrizzato dal grafo a_i. Gli insiemi di grafi sorgente e target ($\mathcal{A}$, $\mathcal{T}$) potrebbero essere disgiunti, sovrapposti o addirittura uguali.

In altre parole, proponiamo di apprendere la rappresentazione di un grafo evento mettendolo a confronto con una popolazione di altri grafi evento presi come riferimento e che andranno a costituire le basi del nostro spazio vettoriale obiettivo (Figura 2.1).

2.2.2 Representation Alignment di Grafi Evento

Per definire la similarità tra due grafi evento in una coppia $\langle target, sorgente \rangle$ (Equazione 2.2), ci affidiamo a una rete neurale deep. Viene addestrato un encoder per apprendere la struttura del grafo evento scelto come punto di riferimento. Il modello risultante viene quindi utilizzato per predire la struttura dell'evento target e ciò permette di poter ottenere una misura di divergenza. Se la coppia risulta simile, è naturale aspettarci che l'encoder del grafo sorgente sarà in grado di predire la struttura del grafo evento target. Si tiene a precisare che due grafi evento che non condividono gli stessi id dei nodi e che hanno dimensioni differenti possono risultare comunque simili. Allo stesso modo nodi di grafi equivalenti dal punto di vista strutturale possono avere attributi completamente differenti. Per questa ragione, utilizziamo un meccanismo di cross-graph attention per apprendere le corrispondenze esistenti tra i nodi del grafo target e quelli del grafo sorgente.

Adesso illustreremo il funzionamento delle diverse componenti in gioco. **Encoder dei Grafi Evento Prototipo**. La qualità degli embedding ottenuti dipende dalla misura in cui l'encoder è in grado di svelare la struttura dei suoi grafi prototipo. Pertanto, il ruolo dell'encoder è quello di ricostruire questa

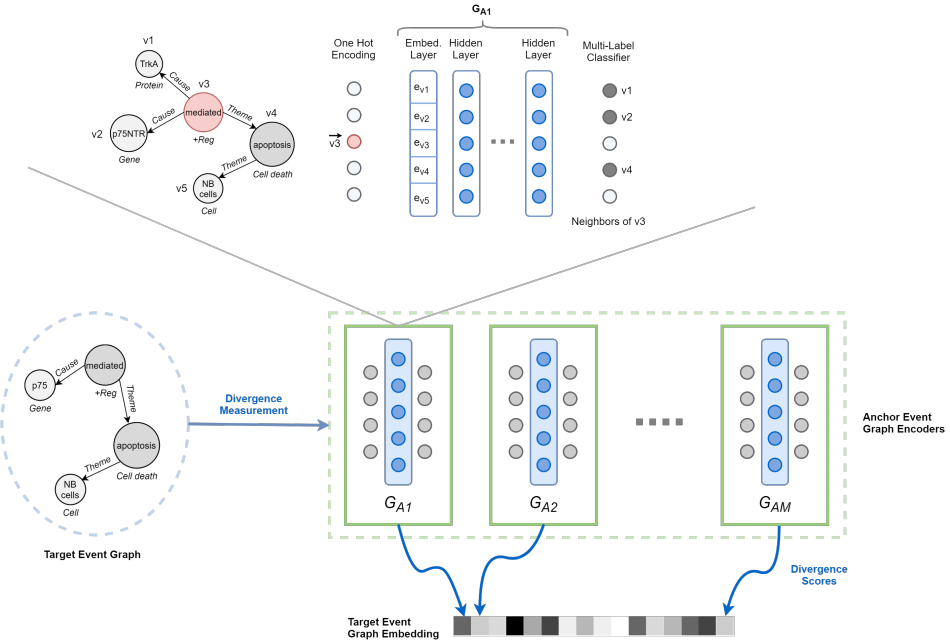


Figura 2.1: Panoramica di DDEGK. I punteggi di divergenza strutturale e semantica calcolati partendo da un insieme di grafi evento sorgente sono utilizzati per comporre la rappresentazione vettoriale di un grafo evento target. La divergenza viene misurata grazie agli encoder di modelli Node-to-Edges preaddestrati, uno per ogni singolo grafo sorgente.

struttura a partire da un'informazione parziale e distorta. Analogamente al lavoro di Al-Rfou et al. [30], scegliamo di utilizzare un modello Node-To-Edges, dove l'encoder viene allenato a predire i vicini di un singolo nodo di input. Nonostante sia indubbiamente possibile adottare altre strategie, questa si adatta molto bene alle caratteristiche dei grafi evento e risulta piuttosto semplice da elaborare e in grado di mantenere la sua efficienza. Modellando il problema come un task di classificazione multi-label, massimizziamo la seguente funzione obiettivo,

$$J(\theta) = \sum_{i} \sum_{\substack{j \\ e_{ij} \in E}} logPr(v_j|v_i, \theta). \tag{2.3}$$

All'inizio, ogni nodo v_i all'interno del grafo viene rappresentato da un vettore one-hot $\vec{v}_i$. Il secondo passo consiste nel moltiplicare il vettore one-hot con un linear layer $E \in \mathbb{R}^{|V| \times d}$, e si ottiene l'embedding di un nodo $e_{v_i} \in \mathbb{R}^d$, dove d è la dimensione dello spazio di embedding. Per terza cosa, l'embedding e_{v_i} (insieme di feature) così ottenuto viene passato a una fully connected deep neural network (DNN), che calcola i punteggi per ogni nodo in V (e quindi la

dimensione del layer di output sarà $|V|$). Infine i punteggi sono normalizzati, utilizzando una sigmoide per generare le predizioni finali.

Cross-Graph Attention. Per essere in grado di mettere a confronto coppie di eventi che possono differire sia per dimensioni (numero di nodi) sia per struttura (numero di archi), occorre apprendere una corrispondenza tra i due grafi. Per fare ciò, sfruttiamo un meccanismo di attention che codifica una nozione rilassata di isorfismo di grafi. Questa *isomorphism attention* allinea bidirezionalmente i nodi del grafo target con quelli del grafo sorgente, andando a operare idealmente in assenza di una mappatura diretta tra i nodi e andando a disegnare delle corrispondenze non necessariamente uno a uno. Richiede due reti di attention separate. La prima rete, indicata con $\mathcal{M}_{\mathcal{T}\to\mathcal{A}}$, fa in modo che i nodi all'interno del grafo target ($G_T \in \mathcal{T}$) prestino una maggiore attenzione a quei nodi del grafo sorgente ($G_A \in \mathcal{A}$) che risultano maggiormente simili dal punto di vista della struttura. Nello specifico, assegna a ciascun nodo $t_i \in V(G_T)$ una distribuzione di probabilità (softmax) calcolata rispetto ai nodi $a_j \in V(G_A)$, e quindi si tratta di una mappatura 1:N. A livello implementativo, utilizziamo un classificatore multiclasse.

$$Pr(a_j|t_i) = \frac{e^{\mathcal{M}_{\mathcal{T}\to\mathcal{A}}(t_i,a_j)}}{\sum_{a_k\in V(G_A)} e^{\mathcal{M}_{\mathcal{T}\to\mathcal{A}}(t_i,a_k)}} \tag{2.4}$$

La seconda rete, che indichiamo con $\mathcal{M}_{\mathcal{A}\to\mathcal{T}}$, è una rete di reverse attention che mappa un gruppo di vicini all'interno del grafo sorgente in un gruppo di vicini del grafo target, questo significa che viene fatta una mappatura N:N. Lo implementiamo come un classificatore multi-label per mezzo di una sigmoide.

$$Pr(t_j|\mathcal{N}(a_i)) = \frac{1}{1 + e^{-\mathcal{M}_{\mathcal{A}\to\mathcal{T}}(\mathcal{N}(a_i),t_j)}} \tag{2.5}$$

Combinando l'encoder dei grafi sorgente con entrambe le reti di attention, siamo in grado di predire i vicini di ciascun nodo all'interno del grafo target e siamo in grado di fare ciò utilizzando la struttura del grafo sorgente. Questa isomorphism attention è in grado di catturare strutture tra grafi anche di ordine più alto, andando oltre i gruppi di vicini immediati.

Coerenza degli attributi. Come sappiamo i grafi evento non sono definiti solo dalla loro struttura ma anche dagli attributi dei loro nodi e archi. Di conseguenza, per poter apprendere dei legami di similarità che mantengano la semantica, aggiungiamo delle regularizing loss alle reti di attention e di reverse attention. A differenza di DDGK, aggiungiamo il supporto agli attributi continui, indispensabile all'interno del dominio degli eventi.

Per quanto riguarda il tipo del nodo $\tau_v()$, calcoliamo una distribuzione di probabilità degli attributi discreti rispetto al grafo target basata sui punteggi

di attention appresi che è definita nel seguente modo:

$$Q_{\tau_v}(y_i|t_j) = \sum_k \mathcal{M}_{\mathcal{T}\to\mathcal{A}}(y_i|a_k) Pr(a_k|t_j), \tag{2.6}$$

dove $\mathcal{M}_{\mathcal{T}\to\mathcal{A}}(y_i|a_k)$ verifica semplicemente se i nodi del grafo sorgente che sono stati predetti hanno etichetta y_i oppure no (1 o 0). Per definire la funzione di attention regularizing loss per i tipi dei nodi, adottiamo una average cross-entropy loss con l'obiettivo di misurare la differenza tra la distribuzione osservata degli attributi discreti $Pr(y_i|t_j)$ e quella dedotta $Q_{\tau_v}(y_i|t_j)$.

$$L_{\tau_v} = \frac{1}{|V(G_T)|} \sum_j^{|V(G_T)|} \sum_i Pr(y_i|t_j) log(Q_{\tau_v}(y_i|t_j)) \tag{2.7}$$

Per quanto riguarda il tipo di arco $\tau_e()$, impostiamo $Q_{\tau_e}(y_i|t_j)$ come il numero degli attributi normalizzati rispetto al numero di archi connessi al nodo t_j. Per esempio, se un nodo t_j ha tre archi tra cui due di essi etichettati come "Theme" e "Cause" e consideriamo $Q_{\tau_e}(Theme|t_j) = 0.67$. Sostituendo Q_{τ_v} con Q_{τ_e} nelle Equazioni 2.6 e 2.7, otteniamo una funzione di regularizing loss per gli attributi discreti degli archi.

Per quanto riguarda il testo del nodo $\gamma_v()$, minimizziamo la seguente funzione di loss, come la riduzione media della distanza coseno tra l'embedding delle etichette testuali dei nodi dei grafi sorgente e target, utilizzando come pesi i loro punteggi di similarità.

$$L_{\gamma_v} = \frac{1}{|V(G_T)| \times |V(G_A)|} \sum_j^{|V(G_T)|} \sum_i^{|V(G_A)|} cos\_dist(emb(\gamma_v(t_j)), emb(\gamma_v(a_i))) Pr(a_i|t_j). \tag{2.8}$$

Nel nostro lavoro, utilizziamo gli embedding preaddestrati di SciBERT [116]. L'idea chiave è quella di sfruttare la conoscenza linguistica e del dominio appresa dai language model biomedici, altrimenti non inclusa all'interno dei grafi evento.

Ispirati dal lavoro di DeepEventMine (stato dell'arte nell'event extraction in ambito biomedico) [3], abbiamo implementato la funzione $emb()$ utilizzando la seguente rappresentazione $m_{f,l}$, che indica la porzione di testo che va dalla prima parola f fino all'ultima l:

$$m_{f,l} = \begin{bmatrix} v_{f,1}; \dfrac{\sum_{i=f}^{l} \sum_{j=1}^{s_i} v_{i,j}}{\sum_{i=f}^{l} s_i}; v_{l,s_l} \end{bmatrix}, \tag{2.9}$$

dove $[;;]$ indica una concatenazione, mentre $v_{i,j}$ indica la rappresentazione del j–esimo token all'interno della i–esima parola. Per essere più specifici, $v_{f,1}$ è la rappresentazione del primo token della f–esima parola, mentre v_{l,s_l} è la rappresentazione dell'ultimo token della l–esima parola.

Le regularization loss presentate vengono utilizzate anche all'interno delle reti di reverse attention. In questo caso la distribuzione degli attributi fa riferimento ai vicini del nodo, e gli archi dei vicini del nodo sono quelli che appaiono a distanza di due "salti" dal nodo.

La Figura 2.2 mostra un esempio del funzionamento dell'encoder dei grafi target migliorato con la cross-graph attention.

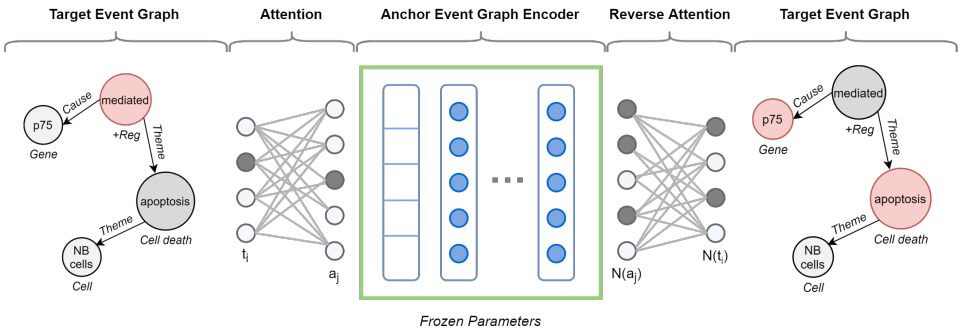


Figura 2.2: Encoder del grafo target basato sull'encoder del prototipo per la predizione di un punteggio di divergenza. I layer di attention mappano i nodi del grafo target sul grafo sorgente, rimanendo consapevoli degli attributi dei nodi e degli archi. La prima rete di attention ($\mathcal{M}_{\mathcal{T}\to\mathcal{A}}$) riceve un vettore one-hot che rappresenta un nodo (t_i) all'interno del grafo target e lo mappa sul nodo (a_j) del grafo sorgente che risulta strutturalmente e semanticamente più simile. Dopodichè, l'encoder del grafo evento sorgente predice i vicini di a_j, $N(a_j)$. Infine, la rete di reverse attention ($\mathcal{M}_{\mathcal{A}\to\mathcal{T}}$) prende $N(a_j)$ e li mappa nei vicini di t_i, $N(t_i)$.

2.2.3 Divergenza e Embedding di Grafi Evento

In questa sezione, proponiamo di utilizzare l'encoder "aumentato" per definire una misura di dissimilarità tra un grafo target e uno sorgente, dalla quale ricaviamo una funzione graph kernel (Equazioni 2.1 e 2.2). In altre parole, la misura di distanza viene appresa dai dati (distance metric learning) in un modo non supervisionato e viene poi convertita in un kernel.

Sottolineiamo che apprendere una metrica misurando i punteggi di divergenza tra una coppia di grafi G_A e G_T è possibile. Faremo riferimento all'encoder

allenato sul grafo sorgente G_A come H_{G_A} e al punteggio di divergenza dato al grafo target G_T come:

$$D'(G_T||G_A) = \sum_{v_i \in V(G_T)} \sum_{\substack{j \\ e_{ij} \in E(G_T)}} -logPr(vj|vi, H_{G_A}). \tag{2.10}$$

Questa operazione risulta equivalente a moltiplicare le probabilità di corretta previsione dei vicini di ciascun nodo target usando l'encoder del grafo sorgente con i punteggi di similarità semantica e strutturale. Se due grafi sono semanticamente e strutturalmente simili ci aspettiamo che il corrispondente punteggio di divergenza risulti basso. Dal momento che H_{G_A} non è in grado di predire perfettamente la struttura di G_A, possiamo assumere con certezza che $D'(G_A||G_A) \neq 0$. Per correggere questo problema e assicurare l'identità, definiamo:

$$D'(G_A||G_T) = D'(G_A||G_T) - D'(G_A||G_A), \tag{2.11}$$

che pone $D(G_A||G_A)$ a zero. Questa definizione però non è simmetrica dal momento che $D(G_T||G_A)$ non è necessariamente uguale a $D(G_A||G_T)$. Se la simmetria è richiesta possiamo utilizzare:

$$D(G_A, G_T) = D(G_A||G_T) + D(G_T||G_A). \tag{2.12}$$

Un recente lavoro, chiamato D2KE (dalle distanze a kernel e embedding) [91], propone una metodologia generica per ricavare un kernel definito-positivo da una qualsiasi funzione di distanza $d : G \times G \to R$. Utilizziamo D2KE per sviluppare una rete neurale non supervisionata per apprendere embedding a livello di grafo basata sull'architettura che abbiamo descritto in precedenza. Selezioniamo un sottoinsieme di campioni per l'addestramento che verranno considerati come prototipi e utilizziamo le distanze dai punti dentro questo gruppo come funzione per il calcolo delle feature. Infine, stabiliamo uno spazio vettoriale dove ciascuna dimensione corrisponde a un singolo grafo evento tra quelli scelti come sorgente. I grafi evento target sono quindi rappresentati come punti all'interno di questo spazio vettoriale:

$$\Psi(G_T) = [D(G_T||G_{A_0}), D(G_T||G_{A_1}), ..., D(G_T||G_{A_M})]. \tag{2.13}$$

Per creare un kernel partendo dagli embedding dei nostri grafi evento, utilizziamo la misura della distanza Euclidea misurata come sottolineato nella Equazione 2.2. In sostanza, il nostro graph kernel identifica le associazioni topologiche e semantiche tra nodi di singoli eventi.

2.2.4 Addestramento

Ottimizziamo l'implementazione originale utilizzando TensorFlow e Adam. Nello specifico, alleniamo M encoder di grafi sorgente, dopodichè congeliamo i parametri e aggiungiamo due layer di attention per fare una mappatura $target \leftrightarrow sorgente$ e le regularizing loss per conservare l'informazione semantica. Infine, gli encoder "aumentati", vengono allenati su ciascun grafo evento considerato in input, rappresentato come una matrice di adiacenza e accompagnato da una matrice aggiuntiva per ogni singolo tipo di attributo (τ_v, τ_e, e γ_v).

2.2.5 Scalabilità

Com'è tipico degli approcci kernel, il nostro metodo fa affidamento sul calcolo della similarità tra coppie di grafi e richiede $N \times M$ confronti per dare un punteggio a ciascuno degli N grafi evento rispetto agli M grafi prototipi (e quindi abbiamo complessità quadratica rispetto al numero e alla dimensione dei grafi). Si tiene però conto che nella Sezione 4 dimostriamo che non è necessario calcolare l'intera matrice kernel per raggiungere alte prestazioni. I migliori risultati, in generale, sono stati ottenuti utilizzando solo 64 prototipi (dimensioni di embedding) su una popolazione avente più di 1000 eventi biomedici.

Gli embedding di grafi evento all'interno di un grande database possono essere precalcolati e indicizzati e questo permette di potervi accedere in modo efficiente per mezzo di algoritmi per l'estrazione di vicini come il locality sensitive hashing [117].

Se le popolazioni di grafi prese in considerazione dovessero cambiare nel corso del tempo, per esempio perchè sono stati espressi nuovi grafi evento all'interno della letteratura biomedica, non sarà necessario ricalcolare gli embedding ma solo i valori di divergenza tra gli eventi più recenti e i nuovi prototipi (andranno utilizzati però gli encoder allenati in precedenza).

2.2.6 Configurazione Hardware e Software

Tutti gli esperimenti sono stati condotti su un server dotato di una GPU Titan Xp con 12 GB di memoria dedicata, 4 core di CPU (processore Intel i5-6400 2.70GHx), 24GB di RAM, con Ubuntu 16.04.6 LTS in esecuzione. Per task e verifiche che richiedevano minori risorse ci siamo spostati su Google Colab.

Capitolo 3

Dettagli implementativi

In questo capitolo si analizza il progetto entrando nel dettaglio dell'implementazione. Verranno riportati estratti di codice per mostrare quanto svolto e quanto appreso sulle tecnologie utilizzate all'interno del lavoro. Ho organizzato il codice in cinque gruppi ciascuno dei quali viene preso in considerazione nella sezione dedicata.

3.1 DDEGK

Partendo dal progetto di DDGK pubblicato sul repository Github ufficiale di *google-research*[1] sono state introdotte alcune modifiche. Fondamentale per lavorare con gli eventi è l'introduzione della loss sugli embedding dei nodi.

```
def NodesEmbeddings(graph):
  # embeddings size is (graph_num_node, D) where D is the size of
      embeddings
  embeddings = [graph.node[n]['embedding'] for n in graph.nodes()]
  assert len(set(map(len, embeddings))) == 1, 'Some embeddings have
      different size'
  return tf.convert_to_tensor(embeddings, dtype=tf.float64)

def NodeEmbeddingLoss(source, source_node_prob, target):
  # source_embeddings size is (source_num_node, D) and is normalized
  source_embeddings = tf.linalg.l2_normalize(NodesEmbeddings(source),
      axis=1)

  # target_embeddings size is (target_num_node, D) and is normalized
```

[1] https://github.com/google-research/google-research/tree/master/graph_embedding/ddgk

```
target_embeddings = tf.linalg.l2_normalize(NodesEmbeddings(target),
    axis=1)

predicted_embeddings = tf.matmul(source_node_prob,
    source_embeddings)

# The reduction is actually a mean
return tf.cast(tf.losses.cosine_distance(predicted_embeddings,
    target_embeddings, axis=1,
    reduction=tf.losses.Reduction.SUM_OVER_BATCH_SIZE), tf.float64)
```

Listato 3.1: Loss sugli embedding dei nodi

3.2 Utilities

3.2.1 Ranking vicini

Nel corso del lavoro è risultato particolarmente utile, soprattutto per effettuare un'analisi di tipo qualitativo sulla bontà dei risultati, prendere in considerazione un evento e osservare le caratteristiche dei cinque grafi più vicini. Questo è stato implementato grazie alla classe **KDTree** di SciPy che permette di costruire un indice su un insieme di punti anche in uno spazio ad elevata dimensionalità e che può essere usato proprio per ottenere i vicini.

```
#K -> number of neighbours
#index -> graph id

#tree
tree = scipy.spatial.KDTree(embeddings)

#K nearest neighbours
dist, idx = tree.query(embeddings[index], k=K+1)

#sort
temp = [(dist[i], idx[i]) for i in range(len(dist))]
neighbours = [(embeddings[i], d, i) for d, i in temp]
neighbours.sort(key=lambda e:e[1])
#to skip the event itself we consider neighbours[1:]
```

Listato 3.2: Codice per ottenere i K vicini ordinati rispetto a un grafo scelto

3.2.2 Sampling stratificato

Dal momento che i dataset di partenza che sono stati presi in considerazione risultano molto grandi, è stato necessario effettuare un sampling stratificato. Questo ci ha permesso di lavorare con dei campioni che rappresentassero in modo significativo il rispettivo dataset e di ridurre i tempi necessari per gli esperimenti. Il codice è stato realizzato all'interno di un jupyter notebook facendo uso sia di Python che di R.

```
#this command unable R in the notebook
%load_ext rpy2.ipython

#we install the package needed for the sampling
%%R
install.packages("splitstackshape")
```

Listato 3.3: Attivazione di R all'interno del notebook

```
#dataframe
df = pd.DataFrame(graphs)

#add event type column
# Obtain the root node of each graph
roots = [g[1]['root'] for g in df['graph'].items()]
# Obtain a list with the information of each node in each graph
nodes_list = [n[1] for n in df['nodes'].items()]
# For each event graph, we take the type information from the trigger
    node (id matching)
event_type = [n['type'] for i in range(len(graphs)) for n in
    nodes_list[i] if n['id'] == roots[i]]
df['event_type'] = event_type

#add column for number of nodes
lenghts = [len(nodes_list[i]) for i in range(len(df))]
df['len'] = lenghts
#use it as a range
df['range_len'] = df['len'].copy()
df['range_len'][df['len'] >= 7] = 7
df['range_len'] = [str(n) for n in df['range_len']]
```

Listato 3.4: Aggiornamento delle colonne all'interno del dataframe

```
#grouping on dataset name, event type, and graph size range
%%R -i data
```

```
samples <- stratified(data, c('event_type', 'range_len'), 10,
    replace=FALSE, keep.rownames=TRUE)
```

Listato 3.5: Sampling

3.2.3 Statistiche del dataset

Di seguito viene riportato il codice per ricavare alcune importanti statistiche. Consideriamo il dataframe costruito nella sezione 3.2.2.

```
#maximum and minimum number of nodes
max(len_nodes), min(len_nodes)
#average number of nodes
avg_nodes = np.mean(lenghts)

#average number of edges
edges_list = [n[1] for n in df['links'].items()]
len_edges = [len(edges_list[i]) for i in range(len(df))]
avg_edges = np.mean(len_edges)

#number of different event types
len(set(event_type))

#number of different entity types
# For each event graph, we take the type information from all the
#nodes except the trigger node (id matching)
entity_type = [n['type'] for i in range(len(graphs)) for n in
    nodes_list[i] if n['id'] != roots[i]]
len(set(entity_type))

#number of different edge types
edges_type = [el[0]['key'] for el in edges_list]
len(set(edges_type))

#distribution based on number of nodes
df['range_len'].value_counts()

#distribution based on event type as pie chart
df['event_type'].value_counts().plot.pie(autopct='%1.1f%%',radius=3.3)
```

Listato 3.6: Codice per calcolare le statistiche relative a nodi archi e etichette

3.2.4 Indici di clustering

Per essere in grado di valutare la bontà degli embedding calcolati da un punto di vista differente, è stato realizzato del codice per eseguire un esperimento di clustering di grafi andando a calcolare due indici di clustering: **silhouette score** e **rand index**.

```
#silhouette score from sklearn
metrics.silhouette_score(embeddings, event_type)
```

Listato 3.7: Esempio per calcolare il silhouette score dati gli embedding dei grafi e il rispettivo tipo di evento

```
#labels
event_names = list(set(event_types))
labels = [event_names.index(et) for et in event_type]

#KMean clustering
KMean= KMeans(n_clusters=len(event_names), random_state=42)
KMean.fit(embeddings)
label=KMean.predict(embeddings)

#adjusted rand index score
adjusted_rand_score(labels,label)
```

Listato 3.8: Esempio per calcolare il rand index dati gli embedding dei grafi e il rispettivo tipo di evento

3.3 Lavoro fatto sulla baseline

In questa sezione mostrerò il codice che ho realizzato per poter lavorare con i metodi della baseline. Mi concentrerò sul lavoro fatto su ciascun metodo entrando nel dettaglio delle modifiche apportate.

3.3.1 Deep Graph Convolutional Neural Network (DGCNN)

Per utilizzare il modello **DGCNN**[60] ho fatto riferimento all'implementazione fornita dalla libreria **StellarGraph**[118] e ho utilizzato come punto di partenza il notebook [2] di esempio che StellarGraph mette a disposizione. È stato necessario esprimere i grafi come **stellargraph**. Dal momento che

[2] https://colab.research.google.com/github/stellargraph/stellargraph/blob/master/demos/graph-classification/dgcnn-graph-classification.ipynb

per lavorare con i nostri grafi, memorizzati all'interno di un file json, abbiamo prevalentemente utilizzato la libreria networkx, ho sfruttato la possibilità di convertire gli eventi letti grazie a networkx in stellargraph.

```
#read graphs stored in a .json file with networkx
#read labels for the classification
with open('graphs_file.json', 'r') as file:
    graphs = [nx.node_link_graph(g) for g in json.load(file)]
    roots = [g.graph['root'] for g in graphs]
    nodes_list = [list(g.nodes.data('type')) for g in graphs]
    graph_labels = [n[1] for i in range(len(graphs)) for n in
        nodes_list[i] if n[0]== roots[i]]

# Normalize nodes ids
for graph in graphs:
  mapping = {n: i for i,n in enumerate(graph.nodes)}
  nx.relabel_nodes(graph, mapping, copy=False)

# Normalize node labels (type)
all_node_labels = sorted(set(t for g in graphs for _,t in
    g.nodes.data('type')))
mapping = pd.get_dummies(pd.Series(all_node_labels))
for graph in graphs:
  for n,t in graph.nodes.data('type'):
    #this is the node feature we are going to consider
    graph.nodes[n]['feature'] = mapping[t]

#labels as Series
graph_labels = pd.Series(graph_labels)

#conversion
stellar_graphs = [StellarGraph.from_networkx(graphs[i],
    node_features="feature", edge_type_attr="key") for i in
    range(len(graphs))]
```

Listato 3.9: Codice per la conversione del dataset per StellarGraph.

3.3.2 graph2vec

L'implementazione di **graph2vec** [66] utilizzata è quella fornita dalla libreria **karateclub**[119]. In questo caso è sufficiente passare i grafi letti con networkx.

```
# read file
with open('graphs_file.json') as ff:
```

```
  graphs = [nx.node_link_graph(g) for g in json.load(ff)]

# Normalize nodes ids
for graph in graphs:
  mapping = {n: i for i,n in enumerate(graph.nodes)}
  nx.relabel_nodes(graph, mapping, copy=False)

# Normalize node labels (type)
all_node_labels = sorted(set(t for g in graphs for _,t in
    g.nodes.data('type')))
node_labels_mapping = {t: i for i,t in enumerate(all_node_labels)}
for graph in graphs:
  for n,t in graph.nodes.data('type'):
    graph.nodes[n]['feature'] = node_labels_mapping[t]

# graph2vec
model = Graph2Vec(attributed=True, epochs=100)
model.fit(graphs)
emb = model.get_embedding()
```

Listato 3.10: Codice necessario per utilizzare graph2vec

3.3.3 node2vec

Dal momento che anche per node2vec[39] è stata utilizzata l'implementazione fornita da karateclub, il lavoro risulta analogo al precedente. Essendo un metodo che calcola gli embedding a livello dei nodi e non di grafo si è deciso di considerare le aggregazioni come media, somma e massimo.

```
# read file
with open('graphs_file.json') as ff:
  graphs = [nx.node_link_graph(g) for g in json.load(ff)]

# Normalize nodes ids
for graph in graphs:
  mapping = {n: i for i,n in enumerate(graph.nodes)}
  nx.relabel_nodes(graph, mapping, copy=False)
  graph.graph['root'] = mapping[graph.graph['root']]

embeddings = []
for graph in graphs:
  #node2vec
  model = Node2Vec()
```

```
  model.fit(graph)
  embeddings.append(model.get_embedding())
# final aggregation with mean
embs = [e.mean(0) for e in embeddings]
```

Listato 3.11: Codice necessario per utilizzare node2vec

3.3.4 node2vec-PCA

Per lavorare con node2vec-PCA[120] ho utilizzato l'implementazione fornita dagli autori su Github [3]. L'algoritmo è pensato per lavorare con grafi di grandi dimensioni e delle porzioni di codice riflettono questa idea andando a filtrare e quindi a scartare grafi con un esiguo numero di nodi. Dal momento che i grafi da noi considerati possono avere anche soli due nodi è stato necessario apportare delle piccole modifiche ai sorgenti per poter prendere in considerazione tutti gli eventi.
L'unico modo per poter lavorare con tutti i nostri grafi è quello di utilizzare un solo canale e rimuovere il requisito di avere almeno una decina di nodi. Sono stati inoltre necessari ulteriori accorgimenti perchè il codice originale, scritto in Python 2.7, non risultava compatibile.

```
#d=max(20,max_n_channels*2)
d = max_n_channels*2
```

Listato 3.12: Modifica all'interno del file get_node2vec.py

Risulta necessario preparare i file contenenti le matrici di adiacenza di ciascun grafo.

```
for i in range(len(graphs)):
  g = graphs[i]
  m = nx.convert_matrix.to_numpy_matrix(g).astype(int)
  with open(adj_matrix_path+str(i)+'.txt', 'w') as f:
    for line in m:
          np.savetxt(f, line, fmt='%i')
```

Listato 3.13: Codice per la memorizzazione delle matrici di adiacenza

Di seguito vengono riportati i comandi necessari per mandare in esecuzione l'algoritmo.

```
#node2vec
```

[3] https://github.com/Tixierae/graph_2D_CNN

```
! python graph_2D_CNN/code/get_node2vec_mod.py
    graph_2D_CNN/code/node2vec/ graph_2D_CNN/datasets/data_as_adj/
    graph_2D_CNN/datasets/raw_node2vec/ graph_2D_CNN/statistiche/
    st09 1 1

#histograms
! python graph_2D_CNN/code/get_histograms_mod.py
    graph_2D_CNN/datasets/raw_node2vec/
    graph_2D_CNN/datasets/tensors/ st09 1 1 14 1

#model
!python graph_2D_CNN/code/main.py graph_2D_CNN/datasets/ st09 1 1 14 1
```

Listato 3.14: Esempio di istruzioni necessarie per avviare node2vec-PCA

3.3.5 Embedding con SciBERT e BioBERT

In questo caso consideriamo gli embedding ottenuti grazie a un language model BERT_based. In particolare otteniamo gli embedding delle parole all'interno dei nodi e prendiamo come embedding a livello di grafo l'aggregazione ottenuta calcolando la media, la somma o il massimo. Abbiamo lavorato con i language model preaddestrati resi disponibili da **HuggingFace** SciBERT[121] e BioBERT[122].

```
tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
model = AutoModel.from_pretrained("dmis-lab/biobert-v1.1")
```

Listato 3.15: Inizializzazione di BioBERT

```
tokenizer =
    BertTokenizer.from_pretrained('allenai/scibert_scivocab_uncased')
model = BertModel.from_pretrained('allenai/scibert_scivocab_uncased',
    output_hidden_states=True)
```

Listato 3.16: Inizializzazione di SciBERT

```
#storing graphs as a list of node names
graphs = []
with open(DIR + 'graphs_file.json') as ff:
    for g in json.load(ff):
      graphs.append([n['name'] for n in g['nodes']])

emb = []
```

```
  for text in graph:
    #tokenize
    tokens = tokenizer.encode(text)
    input_ids = torch.tensor(tokens).unsqueeze(0) # Batch size 1

    #embedding
    with torch.no_grad():
      outputs = model(input_ids)

    #consider only the last hidden layer
    tokens_emb = outputs[0]

    #store the embeddings of each entity
    emb.append(entity_emb)

#mean
torch.stack(emb).mean(0)
#sum
# torch.stack(emb).sum(0)
#max
# torch.max(torch.stack(emb), 0)[0]
```

Listato 3.17: Esempio di come ricavare gli embedding

3.4 Classificazione

Questa sezione è dedicata al codice utilizzato per valutare la bontà degli embedding nel task di classificazione. Abbiamo utilizzato SVM (Support Vector Machines) con ten-fold cross validation e effettuando una Grid Search per provare diversi parametri.

```
#X is the dataframe embedding
#y is the classes column

#split
X_train, X_val, y_train, y_val = train_test_split(X, y,
    test_size=0.3, random_state=42, stratify=y)

#stratified k fold
K = 10
skf = StratifiedKFold(K, shuffle=True, random_state=42)

#SVM
```

```
model = Pipeline([
    ("scaler", StandardScaler()),
    ("svc", SVC(random_state=42, max_iter=1_000_000))
])

#grid
grid = {
  "scaler" : [None, StandardScaler()],
  "svc__kernel": ["linear", "rbf", "poly", "sigmoid"],
  "svc__C": np.logspace(1, 9, 9)
}

#training
gs_svm = GridSearchCV(model, grid, cv=skf, n_jobs=-1)
%time gs_svm.fit(X_train, y_train)

#f1-score
y_pred = gs_svm.predict(X_val)
f1 = f1_score(y_val, y_pred, average="weighted")
print(f"f1-score: {f1}")
```

Listato 3.18: Classificazione con SVM

3.5 Configurazione dell'ambiente di lavoro

Parte degli esperimenti relativi a DDEGK sono stati lanciati nella macchina 42 del laboratorio messa a disposizione dal professore Gianluca Moro. In questa sezione mostrerò tutto quello che è servito per configurare l'ambiente di lavoro in cui mandare in esecuzione DDEGK e l'utilizzo che è stato fatto di **scp** e del tool **screen**.

3.5.1 Configurazione del container

Come punto di partenza abbiamo creato un container utilizzando un'immagine che integrasse Tensorflow. Dal momento che il codice di DDEGK risulta scritto in python 3.7 e che la versione che viene installata con l'immagine scelta è la 3.5.2 è stato necessario includere anche questa versione all'interno del container creato.

```
# update
apt update
sudo apt install software-properties-common
```

```
# add to source list
sudo add-apt-repository ppa:deadsnakes/ppa

# install python 3.7
sudo apt install python3.7

# to ensure everything is allright
python3.7 --version
```

Listato 3.19: Installazione python 3.7

Dopo avere eseguito le istruzioni per l'installazione della versione 3.7 e aver verificato che sia avvenuta correttamente, i comandi python e python3 puntano ancora alla versione 3.5.2 pertanto si dovranno lanciare gli esperimenti con il comando python3.7.
Come ultimo passo andranno installati i requisiti specifici di DDEGK. Per farlo è necessario aggiornare **pip** e farlo puntare a python3.7 in modo da evitare che facendo pip install delle librerie queste siano montate su python 3.5.2 invece che su python 3.7: questo renderebbe impossibile trovare le librerie necessarie quando si lancia un'istruzione con python3.7.

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3.7 get-pip.py
```

Listato 3.20: Aggiornamento pip

Ora si può utilizzare pip per installare gli specifici requirement.

3.5.2 Connessione alla macchina in remoto

Per potermi connettere alla macchina del laboratorio ho utilizzato **PuTTY** un client SSH combinato con un emulatore di terminale. Ho lanciato gli esperimenti all'interno di un docker container e ho utilizzato **screen** un terminal multiplexer che mi ha permesso di creare sessioni e gestire esperimenti in parallelo.

```
# start a session
screen -S session_name
# detach a session
# Ctrl+A Ctrl+D
# reattach a session
screen -r session_name
```

Listato 3.21: Comandi fondamentali per gestire una sessione con screen

Infine è stato utilissimo il comando **scp** che mi ha permesso di trasferire dalla mia macchina a quella remota e viceversa i file di cui ho avuto bisogno.

```
# from remote to local
scp -P port_number username@ip_address:source_path destination_path

# from local to remote
scp -P port_number destination_path username@ip_address:source_path
```

Listato 3.22: Comando di esempio per il trasferimento di un file tra remoto a locale

Capitolo 4

Esperimenti e confronto con la baseline

Una volta che gli embedding sono stati calcolati usando DDEGK, essi possono essere sfruttati per diversi task che lavorano con grafi quali classificazione, clustering e similarity search. In questa sezione eseguiamo una serie di esperimenti sia quantitativi che qualitativi per dimostrare l'efficacia del nostro metodo mettendolo a confronto con gli altri. Per prima cosa, mostriamo che gli embedding dei grafi evento appresi da DDEGK rappresentano un insieme di feature sufficiente per predire sia l'ambito biologico sia il tipo a grana fine di un largo numero di interazioni scientifiche. Quindi, mostreremo che le rappresentazioni di eventi simili dal punto di vista della struttura e della semantica vengono raggruppati correttamente, valutando la qualità dello spazio geometrico del kernel tramite indici di clustering e tecniche di visualizzazione. In più, presentiamo i vantaggi che si hanno nell'utilizzare le rappresentazioni degli eventi rispetto ai tradizionali embedding di frasi all'interno del task di semantic textual similarity (STS). Infine, sottolineiamo i benefici che la cross-graph attention porta ai fini dell'interpretabilità.

4.1 Dataset

Per gli esperimenti sono stati usati nove dataset, progettati originariamente per l'estrazione di eventi in ambito biomedico. La maggior parte di essi sono benchmark, con etichette selezionate manualmente dagli esperti del settore, introdotti dalla serie tuttora in corso di competizioni BioNLP-ST, una delle community più popolari per il suo impegno relativo al text mining in ambito biomedico. Ciascun dataset tratta argomenti relativi a diverse branche della biologia.

- **BioNLP-ST 2009 (ST09)** [6]. Dataset preso dalla prima challenge di BioNLP-ST, contenente una sotto-porzione del corpus di eventi di GENIA. Esso comprende 13,623 eventi (in totale considerando i set di training, validation e test) menzionati all'interno di 1,210 abstract di MEDLINE sulle cellule del sangue e i fattori di trascrizione.

- **Genia Event 2011 (GE11)** [123]. Si tratta della versione estesa di ST09, che include ≈4,500 eventi ricavati da 14 articoli di PMC full-text.

- **Epigenetics and Post-translational Modifications (EPI11)** [124]. Dataset sulle mutazioni epigenetiche e sulle modifiche post-traduzionali sulle proteine comuni. Contiene 3,714 eventi estratti da 1,200 abstract.

- **Infectious Diseases (ID11)** [125]. Dataset sui sistemi regolatori formati da due componenti; 4,150 eventi riconosciuti all'interno di 30 paper completi.

- **Multi-Level Event Extraction (MLEE)** [5]. Dataset sullo sviluppo dei vasi sanguigni partendo dal livello subcellulare fino a quello dell'intero organismo; 6,667 eventi ottenuti da 262 abstract.

- **Genia Event 2013 (GE13)** [126]. Versione aggiornata di GE11, con 9,364 eventi estratti esclusivamente da 30 paper completi.

- **Cancer Genetics (CG13)** [127]. Dataset sulla biologia dei tumori, con 17,248 eventi presi da 600 abstract.

- **Pathway Curation (PC13)** [128]. Dataset su reazioni, pathway e cure; 12,125 eventi da 525 abstract.

- **Gene Regulation Ontology (GRO13)** [129]. Dataset sulle regolazioni e trascrizioni genetiche; 5,241 eventi da 300 abstract.

4.1.1 Preprocessing e Sampling dei dati

Le istanze all'interno di questi dataset sono costituite da documenti testuali (*.txt*) e da due file contenenti le annotazioni espresse in un formato standard adottato da BioNLP-ST, il file (*.a1*) contiene le entità "gold" mentre quello (*.a2*) riporta i diversi trigger con le loro relazioni. Facciamo il parsing dei file in formato standoff (*.a**) per convertire in modo automatico le porzioni di testo in grafi evento (Figura 4.1).

Dato l'ampio numero di righe, costruiamo un campione di ≈1,000 eventi[1] provenienti da ciascun dataset utilizzando un campionamento stratificato e casuale (senza sostituzioni/ripetizioni) per conservare l'informazione statistica della popolazione. Stratifichiamo su più variabili, ovvero sul tipo dell'evento (nel caso di eventi innestati si prende in considerazione il tipo principale, cioè la radice del grafo evento) e il numero di nodi (questo discretizzato considerando i range di nodi 2, 3, 4, 5, 6, >7) riuscendo a mantenere le proporzioni originali. Rimuoviamo i tipi di evento rappresentati da meno di dieci elementi a causa di dati fortemente sbilanciati. Per poter fare esperimenti sulla mappatura di eventi provenienti da fonti diverse su un singolo spazio condiviso, abbiamo creato un dataset artificiale che indicheremo, d'ora in avanti, con "bio_all"; esso è il risultato della combinazione dei nove dataset che abbiamo introdotto in precedenza stratificato ulteriormente sulla base del dataset di provenienza.
Un riassunto conciso e una visualizzazione ricca di dettagli sui dataset finali vengono resi disponibili, rispettivamente nella Tabella 4.1 e nella Figura 5.1. La limitata dimensione dei grafi evento è compensata dalla vasta diversità di classi (17, in media, in contrasto con le 2-6 etichette delle più comuni benchmark di GRL in ambito biomedico [134]).

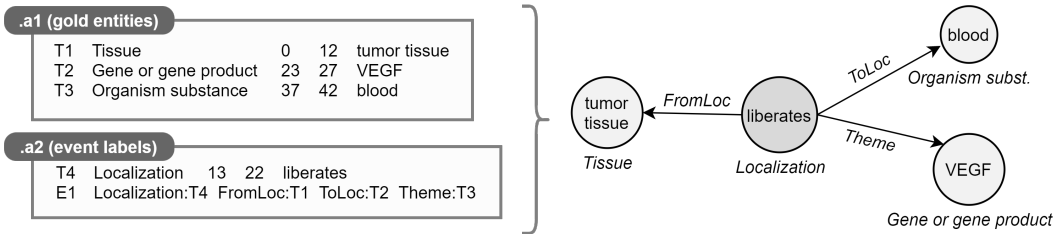


Figura 4.1: Esempio di parsing dei file .a*

[1]Una dimensione paragonabile o superiore a quella di altri dataset utilizzati per la classificazione di grafi nell'ambito della chemioinformatica e bioinformatica come D&D [130] (1,178), PTC-MR [131] (344), ENZYMES [132] (600), e MUTAG [133] (188).

Tabella 4.1: Statistiche dei campioni di dataset considerati. La prima colonna indica il numero di istanze (gli eventi innestati valgono uno). Le ultime tre colonne fanno riferimento al numero distinto di tipi di evento (i trigger), di entità e dei ruoli che possono assumere gli argomenti all'interno di ciascun dataset, e quindi servono per mettere in evidenza il numero di possibili classi dei nodi e degli archi.

Dataset	**# Graphs**	**# Nodes**			**# Edges**	**# Labels**		
		min	mean	max	mean	graph	node	edge
ST09	1007	2	4	14	3	9	11	3
GE11	1,001	2	4	14	3	9	11	2
EPI11	1002	2	3	6	2	10	11	1
ID11	1,001	2	3	14	2	9	16	3
MLEE	1012	2	4	15	3	15	39	8
GE13	1,011	2	3	13	2	8	13	2
CG13	1033	2	3	13	2	23	50	8
PC13	1,020	2	4	18	3	15	25	9
GRO13	1006	2	3	5	2	18	140	4
BIO_ALL	1,216	2	3	14	3	53	101	12

4.2 Classificazione di Grafi Evento

Le rappresentazioni degli eventi che abbiamo appreso rispettano sia la struttura che gli attributi discreti e continui. Perciò, possono essere utilizzate all'interno di un task di classificazione di grafi dove la struttura, gli attributi dei nodi e degli archi sono in grado di trasmettere un significato. Per dimostrare ciò, utilizziamo le rappresentazioni ottenute con DDEGK sui dataset di eventi che abbiamo descritto nella Sezione 4.1 come feature per predire l'etichetta dei grafi, andando ad apprendere una funzione di mappatura tra gli embedding degli eventi e le classi. Nello specifico, andiamo a valutare la significatività delle rappresentazioni apprese su due task di classificazione: (i) predire il principale tipo di evento di ciascun grafo, e quindi, la natura dell'interazione espressa; (ii) predire l'ambito biologico di ciascun grafo, che viene approssimativamente rappresentato dal dataset di appartenenza. Il secondo punto fa riferimento allo spazio condiviso degli eventi di "bio_all" che racchiude eventi provenienti da dataset eterogenei.

4.2.1 Metodi della Baseline

Mettiamo a confronto le prestazioni di DDEGK con una serie di modelli per l'embedding di grafi – sia supervisionati che non supervisionati – che riescono

a conservare proprietà diverse dei grafi evento. Questi modelli sono rappresentativi delle principali tecniche di embedding e possono essere considerati come appartenenti a quattro differenti categorie.

- *Node embedding flat pooling.* Ciascun evento viene rappresentato come l'aggregazione non supervisionata dei vettori dei nodi che lo costituiscono. Abbiamo fatto esperimenti con diverse strategie di flat pooling, nello specifico abbiamo considerato le aggregazioni ottenute calcolando media, somma e valore massimo. Per le rappresentazioni dei nodi, abbiamo esaminato (i) embedding di parole ottenuti grazie a modelli preaddestrati su testi scientifici e biomedici che permettono di tenere conto del contesto in cui vengono utilizzate le parole, (ii) node2vec [39]. Per il primo punto abbiamo applicato SciBERT [116] e BioBERT [135] (embedding con 768 dimensioni) al testo corrispondente ai trigger e alle entità; un approccio comune utilizzato nell'ambito del Graph Representaion Learning di eventi [111]. In questo modo è in grado di racchiudere una sintesi a livello semantico di un evento basandosi sulle entità coinvolte; questo approccio ignora totalmenete la struttura e i ruoli degli argomenti. Al contrario, node2vec è una baseline per i metodi sequenziali che è in grado di lavorare in modo efficiente a diversi livelli di prossimità. Come parametri abbiamo considerato un valore di 80 per la default walk length, il numero di walk per nodo è 10, gli iper-parametri return e in-out sono a 1 e la dimensione degli embedding è 128.

- *Node embedding + CNN.* Utilizziamo node2vec-PCA [53] (con d=2 e quindi un solo canale), che compone le matrici dei grafi ottenute a partire da node2vec per poi applicare una CNN per effettuare una classificazione supervisionata.

- *Whole-graph embedding.* Utilizziamo graph2vec [66] per generare rappresentazioni a livello di grafi, consapevoli della struttura. Dal momento che lavoriamo su grafi etichettati, abbiamo utilizzato come etichette degli identificatori numerici per i tipi di evento e per le entità. Abbiamo ottenuto embedding con 128 dimensioni (misura di default) e scelto un numero di epoche pari a 100.

- *GNN + supervised pooling.* Utilizziamo DGCNN [60], un modello end-to-end per la classificazione di grafi composto da reti GCN e di un layer di sort pooling introdotto allo scopo di ricavare degli embedding di grafi insensibili alle permutazioni. Quindi, una 1D-CNN è in grado di estrarre le feature assieme a un fully-connected layer. La dimensione del grafo normalizzato k è 35.

Nel corso degli esperimenti, abbiamo utilizzato le configurazioni di iper-parametri di default suggerite dagli autori come descritto sopra.

4.2.2 Ricerca degli iperparametri

Per prima cosa otteniamo gli embedding di tutti i grafi evento per ogni modello e poi li passiamo a SVM (ad eccezione di DGCNN dal momento che è un algoritmo supervisionato). Dividiamo i dataset degli eventi in set per l'addestramento e per la valutazione. Per scegliere gli iperparametri di DDEGK (Tabella 4.2), abbiamo effettuato delle grid search per ciascun dataset. In questa fase, abbiamo notato empiricamente che, nei dataset considerati, in media i risultati migliori sono stati ottenuti con un rapporto semantica-struttura di circa 20:1, dando lo stesso peso al contributo di ciascuna etichetta. Per SVM, abbiamo utilizzato l'implementazione di scikit-learn [136] e 10-fold cross validation. Facciamo variare il kernel tra {linear,rbf,poly,sigmoid} e il coefficiente di regolarizzazione C tra 10 e 10^9. Scegliamo la combinazione di DDEGK e gli iperparametri del classificatore che massimizzano l'accuratezza sul set di valutazione. In aggiunta abbiamo condotto degli esperimenti sulla scelta dei grafi prototipo: (i) un sottoinsieme casuale di grafi scelti all'interno del gruppo originale e (ii) una versione più bilanciata rispetto ai tipi di evento. Il secondo metodo è stato ideato con l'obiettivo di ottenere vettori maggiormente interpretabili, dove ogni evento viene rappresentato come un miscuglio dei suoi tipi (risulta analogo alla modellazione degli argomenti [137]). Per mantenere le computazioni sostenibili, si è deciso di condurre esperimenti con 32, 64, e 128 prototipi da selezionare nel campionamento.

Tabella 4.2: Valori degli iperparametri testati all'interno della grid search per le rappresentazioni dei grafi di DDEGK.

Hyperparameter	**Values**
Node embedding	2, 4, 8, 16, 32
Encoder layers	1, 2, 3, 4
Learning rate	10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1
Encoding epochs	100, 300, 600
Scoring epochs	100, 300, 600
γ^v, τ_v, τ_e preserving loss coefficients	{7, 7, 7}, {8, 8, 4}, {10, 6, 4}, {15, 0, 5}

4.2.3 Risultati

I risultati della classificazione sono mostrati nella Tabella 4.3. Notiamo che DDEGK se la cava sorprendentemente bene per un metodo non supervisionato e senza feature specifiche. Il nostro metodo sorpassa significativamente i modelli

della baseline sui principali task di classificazione sul tipo di evento (abbiamo margini assoluti di 22.46 e 6.86 rispettivamente per il migliore modello non supervisionato e il migliore supervisionato). Riesce anche a ottenere risultati competitivi nella classificazione basata sul dataset (ambito biomedico), anche se viene superato da DGCNN (metodo supervisionato). La differenza di prestazioni tra gli approcci puramente semantici e quelli puramente strutturali (come quelli basati sul pooling di embedding di parole mettendoli a confronto con node2vec e node2vec-PCA) mostrano chiaramente quanto sia centrale il ruolo delle etichette per modellare gli eventi. Vediamo graph2vec come un compromesso tra i due approcci. Pertanto, l'utilizzo delle informazioni ausiliarie sembra il motivo principale dietro la bontà della classificazione. Calcolare la media degli embedding generati da un modello basato su BERT sembra essere la tecnica di pooling più efficace, con i risultati di SciBERT che sembrano leggermente più solidi rispetto a quelli ottenuti con BioBERT. Dal momento che la nostra soluzione prende in considerazione sia i collegamenti tra i nodi sia il significato semantico di nodi/archi, la profonda differenza tra la nostra soluzione e SciBERT AVG conferma la validità di una strategia di embedding ibrida. Le prestazioni tendono a calare in proporzione alla dimensione dei grafi e al numero di etichette. Come ci si aspetta, node2vec si dimostra fortemente inefficiente sul dataset GRO13 a causa delle poche differenze in termini di struttura e dalle informazioni limitate sul contesto. La parziale sovrapposizione delle annotazioni tra grafi [138] giustifica i punteggi più bassi ottenuti all'interno degli spazi di eventi che considerano eventi provenienti da fonti eterogenee. Infine possiamo dire che le due strategie per la selezione dei grafi evento prototipo sono paragonabili in termini di prestazioni raggiunte.

Tabella 4.3: Accuracy media (support-weighted F1) in ten-fold cross validation sui task di classificazione del tipo di evento e del dataset. I metodi sono raggruppati in base al tipo di approccio e al livello di supervisione durante l'apprendimento delle rappresentazioni dei grafi. Il più alto F1-score per ciascun dataset viene evidenziato in grassetto.

Method		**Unsupervised**	**Event type classification**											**Dataset**
			ST09	**GE11**	**EPI11**	**ID11**	**MLEE**	**GE13**	**CG13**	**PC13**	**GRO13**	**BIO_ALL**	**AVG**	**classification**
SciBERT	AVG	x	72.16	73.24	89.55	85.45	72.18	82.53	69.50	72.59	78.08	58.72	75.40	53.11
	SUM	x	56.32	58.91	94.34	76.97	58.35	81.02	52.68	53.51	65.35	42.63	64.01	47.68
	MAX	x	64.54	53.95	90.47	75.95	56.16	74.35	63.91	60.39	72.16	51.11	66.30	49.22
BioBERT	AVG	x	71.43	70.11	90.60	85.40	71.91	79.81	66.77	70.40	76.72	54.59	73.77	52.61
	SUM	x	61.49	58.73	93.32	73.22	56.03	76.94	56.36	56.53	59.60	43.64	63.59	50.17
	MAX	x	56.38	53.95	90.47	75.95	56.16	74.35	57.26	56.86	76.72	45.72	64.38	51.82
node2vec	AVG	x	19.65	21.03	23.81	14.18	18.58	19.24	7.51	12.60	9.91	11.38	15.79	14.05
	SUM	x	25.90	23.43	23.71	22.05	15.85	19.37	13.62	15.19	8.89	8.68	17.67	14.31
	MAX	x	28.13	22.06	24.31	29.26	16.45	22.80	13.00	15.42	14.91	11.21	19.76	15.20
node2vec-PCA		x	26.71	24.17	32.04	31.06	22.53	34.43	17.90	24.51	23.16	16.54	25.31	18.26
graph2vec		x	54.12	58.60	57.78	62.25	41.34	63.47	44.59	40.51	33.81	40.48	49.70	43.06
DGCNN			89.19	89.04	90.40	88.70	93.19	86.65	95.73	93.35	94.23	89.55	91.00	**89.55**
DDEGK (*ours*)														
w/ random anchors		x	99.00	**97.99**	**100**	**100**	98.01	**98.02**	**92.89**	**99.67**	**100**	**92.28**	**97.86**	63.39
w/ random anchors per type		x	**99.01**	**97.99**	**100**	**100**	**98.02**	97.70	90.32	**99.67**	**100**	90.87	97.36	62.79

4.3 Between-graph Clustering

Fare clustering di grafi risulta particolarmente utile per scoprire delle community e per numerose altre applicazioni, ad esempio per il raggruppamento di proteine con simili proprietà. Nel nostro contesto permette l'identificazione automatica di eventi simili menzionati all'interno della letteratura biomedica, permettendo sia di categorizzarli sia di quantificarli, andando per esempio a contare quante volte una specifica interazione tra una proteina e un sintomo è stata menzionata. Mettiamo a confronto i diversi metodi utilizzati per fare embedding di grafi evento per esaminare la bontà dei clustering e per capire meglio, dal punto di vista quantitativo, la struttura globale degli spazi di encoding. Nello specifico andiamo a verificare che istanze che condividono lo stesso tipo o lo stesso ambito biomedico di origine abbiano posizioni vicine all'interno dello spazio vettoriale. Dal momento che parliamo di embedding a livello di grafo, effettuiamo clustering tra grafi: partendo da un vasto numero di grafi, tentiamo di ottenere dei cluster di essi (non delle loro componenti) basandoci sui sottostanti comportamenti strutturali e semantici. In contrasto con il convezionale clustering delle componenti all'interno dei grafi, dove l'obiettivo è quello di dividere i nodi all'interno di singoli grafi [139], il nostro obiettivo è più ambizioso perchè dobbiamo trovare dei collegamenti tra sottostrutture. Questa prospettiva ricorda molto i problemi per i quali vengono applicate soluzioni di Deep Learning.

La nostra valutazione prende in considerazione due diverse metriche: silhouette score (SS) e adjusted rand index (ARI). Il primo punteggio misura quanto simile un evento sia al suo cluster (valutando la coesione, la distanza tra i cluster) se messo a confronto con gli altri (valutando la separazione e quindi la distanza dai cluster più vicini). Il secondo calcola la similarità tra due clustering di eventi andando a contare il numero di coppie che sono state assegnate nel medesimo cluster o in cluster diversi confrontando i cluster predetti e quelli veri. Entrambi lavorano nel range $[-1, 1]$ e valori alti indicano migliori risultati. Nel nostro studio utilizziamo la distanza Euclidea. Il calcolo di rand index viene eseguito considerando il clustering migliore ottenuto in seguito a dieci esecuzioni consecutive dell'algoritmo K-means (implementazione di scikit-learn, con $k = \#event\_types$ or $k = \#datasets$). I risultati degli esperimenti sono riportati nella Tabella 4.4.

4.3.1 Risultati

Possiamo vedere che la nostra soluzione ottiene i risultati migliori rispetto a entrambe le metriche. Per facilità di comprensione, di seguito, riportiamo i risultati in percentuale. Per quanto riguarda l'ARI, DDEGK mediamente

supera SciBERT, node2vec, e graph2vec con più del 10.7%, 14.3%, and 11.5%, rispettivamente. Mentre per quanto riguarda SS, il divario si aggira attorno a 3.4%, 23.3%, e 3.8%. Questi risultati rafforzano quanto dedotto dagli esperimenti di classificazione; i punteggi ottenuti dai modelli nel task di clustering sono in linea con quelli di classificazione. Significativamente, il clustering rende evidente l'impatto derivante dalla diversa scelta dei prototipi. La selezione di prototipi casuali per ciascun tipo di evento porta a uno spazio con migliori caratteristiche geometriche ma clustering meno efficace di quelli ottenuti con una scelta interamente casuale.

Tabella 4.4: I risultati di clustering dei grafi in base al silhouette score (righe bianche) e range index (righe grigie). Vengono evidenziati i risultati migliori ottenuti per ciascun dataset.

Method	**Event type clustering**											**Dataset clustering**
	ST09	**GE11**	**EPI11**	**ID11**	**MLEE**	**GE13**	**CG13**	**PC13**	**GRO13**	**BIO_ALL**	**AVG**	
SciBERT (AVG)	-0.00915	-0.00940	0.04082	0.01347	-0.01139	-0.00784	0.00087	-0.01251	-0.00343	**-0.01974**	-0.00183	-0.01123
	0.03638	0.06641	0.06644	0.09005	0.05810	0.04079	0.06644	0.04022	0.07669	0.04075	0.05823	0.05373
node2vec (AVG)	-0.48715	-0.48165	-0.58339	-0.58004	-0.46614	-0.57659	-0.51334	-0.49366	-0.54964	-0.58171	-0.53133	-0.27637
	-0.02987	-0.02539	-0.00234	-0.02405	-0.05510	-0.03748	-0.04139	-0.03689	-0.03148	-0.03830	-0.03223	0.00068
graph2vec	-0.25687	-0.20579	-0.14446	-0.27879	-0.35015	-0.28666	-0.39072	-0.36219	-0.41060	-0.45318	-0.31394	-0.16632
	0.03789	0.04679	0.09058	0.14508	0.03394	0.07483	0.02883	0.02123	0.01443	0.02412	0.05177	0.02761
DDEGK (*ours*)												
w/ random anchors	0.21371	0.17281	0.27985	0.09267	0.08572	0.22813	-0.05122	-0.04587	0.05914	-0.32140	0.07108	**0.00686**
	0.33177	0.34138	**0.54129**	**0.44621**	**0.16441**	**0.51622**	**0.17242**	**0.13066**	**0.39070**	**0.14349**	**0.31786**	0.14349
w/ random anchors per type	**0.23428**	**0.24598**	**0.28723**	**0.15141**	**0.10136**	**0.31270**	**0.08123**	**0.01809**	**0.08047**	-0.29036	**0.12224**	0.00007
	0.35625	**0.38622**	0.39765	0.29661	0.15441	0.43387	0.15185	0.09296	0.23732	0.08320	0.25903	**0.28101**

4.3.2 Influenza delle dimensioni degli embedding

A livello pratico, trovare la dimensione ottimale per gli embedding non è semplice [27]. Un embedding più lungo tende a conservare di più l'informazione dei grafi evento originali al prezzo di maggiori costi computazionali e di memorizzazione, ma rischia anche di mantenere il rumore. D'altro canto, una rappresentazione con un minor numero di dimensioni risulta più efficiente in termini di risorse ma con il rischio di perdere informazioni critiche e di assistere a un forte calo delle prestazioni. I ricercatori sono costretti a dover fare un trade-off basato sulle loro esigenze; Gli articoli di GRL riportano che, solitamente, embedding di 128 e 256 dimensioni siano sufficienti per la maggior parte dei task. Studiamo ora l'effetto che la riduzione delle dimensioni del nostro spazio di embedding ha sulla bontà dei clustering e della classificazione.

Effettuando diversi test su tre dataset bioinformatici con una media di 570 istanze, gli autori di DDGK [30] sono in grado di ottenere risultati stabili e competitivi utilizzando meno del 20% dei grafi come prototipi. Per arricchire queste scoperte cerchiamo di esaminare, entrando maggiormente nel dettaglio, la complessità computazionale e la scalabilità del problema. Costruiamo insiemi differenti di grafi prototipo partendo da quelli originali, adottando le strategie per la selezione dei prototipi elencati nella Sezione 4.2.2. Per ciascuna

configurazione, apprendiamo i punteggi di divergenza rispetto a tutti i grafi target e utilizziamo gli embedding ridotti come feature per la predizione di classi e cluster. Riassumiamo i risultati nella Figura 4.2.

Osserviamo che i punteggi di classificazione su singoli dataset rimangono pressoché uguali al variare della dimensione nel caso di una scelta di prototipi casuali. Abbiamo solo un calo di prestazioni di circa 2% quando la dimensione degli embedding passa da 128 a 32. In media, l'f1-score più alto per la classificazione dei tipi di evento si ottiene con una dimensione di embedding di 64, e questo significa che non è richiesto un numero più elevato di prototipi. Quindi, sono necessarie davvero poche dimensioni per ottenere i nostri risultati finali, descritti nella Sezione 4.2 (molti di meno di quelli che si aspettano i metodi della baseline).

Un più alto numero di dimensioni è invece necessario per il task più complesso di classificazione di grafi evento eterogenei. In questo caso le prestazioni migliorano con l'aumentare della dimensionalità degli embedding, dove rappresentazioni più dettagliate sono in grado in catturare diverse sfaccettature. Diversamente da una scelta completamente casuale, nel caso di prototipi bilanciati rispetto al tipo di evento le prestazioni tendono a peggiorare con l'aumentare delle dimensioni, talvolta sperimentando dei cali considerevoli come nel caso di CG13 e ID11. Collegare i risultati di classificazione e di clustering aiuta a comprendere la bontà complessiva degli embedding e la corretta configurazione degli iperparametri. Per esempio, DDEGK ottiene dei punteggi di accuratezza molto alti nella classificazione su PC13 ma ARI è peggiore; mentre nel caso di EPI11, nonostante la classificazione risulti perfetta con tutte le dimensioni, possiamo vedere come a livello di clustering lo spazio di encoding risulti migliore con 64 dimensioni.

4.4 Visualizzazione

La visualizzazione degli embedding risulta utile per realizzare a livello qualitativo quanto bene DDGK apprende dalla struttura e dalle etichette di nodi e archi dei grafi evento. Dal momento che le nostre rappresentazioni catturano le similarità degli eventi come vicinanze all'interno dello spazio Euclideo, le prestazioni degli embedding dei grafi evento possono essere valutate visualizzando il grado di aggregazione degli eventi (punti colorati). Idealmente, i vettori dei grafi evento all'interno della stessa classe dovrebbero risultare fortemente aggregati all'interno di una regione locale dello spazio di embedding. Allo stesso tempo, i vettori degli eventi appartenenti a classi diverse dovrebbero essere il più possibile separabili per rafforzare i modelli di machine learning. Pertanto conduciamo, intuitivamente, un controllo visuale per osservare come

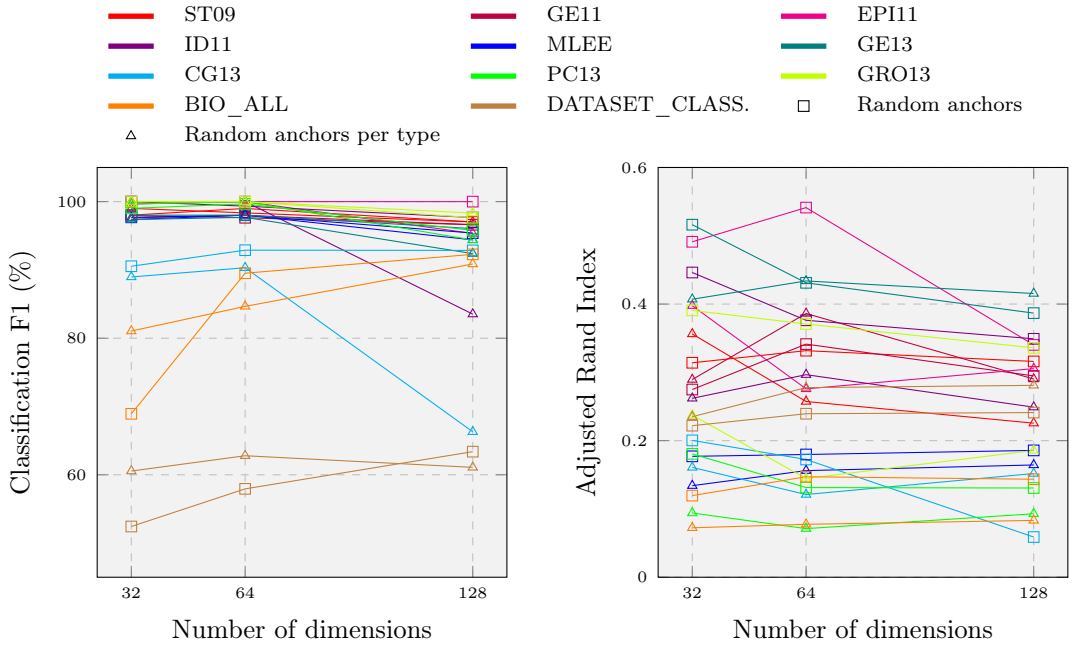


Figura 4.2: Effetto della riduzione del numero di prototipi sui task di classificazione e di clustering dei grafi evento in relazione a ogni dataset biomedico (linee colorate). Facciamo variare il numero di grafi prototipo tra 32, 64 e 128, impiegando due differenti strategie per la scelta dei prototipi (simboli).

gli embedding dei grafi evento biomedici, approssimati con t-SNE [140] risultano raggruppati in una proiezione 2D. In particolare mostreremo la visualizzione dello spazio bidimensionale associato a EPI11, lo spazio che risulta migliore dal punto di vista degli indici di clustering (Figura 4.3). DDGEK fa in modo che gli embedding di grafi evento che hanno tipi simili si collochino vicini all'interno dello spazio di embedding, permettendo di ottenere cluster distinguibili. Notiamo che potrebbero formarsi dei cluster separati dello stesso tipo di evento, spesso a causa di strutture diverse [2] oppure insiemi di attributi differenti.

Nella Figura 4.4, mostriamo tre esempi di event retrieval. Il primo mostra un semplice caso in cui la query è unbounded, ovvero contiene un'entità con una descrizione generica "cells" e i grafi restituiti forniscono informazioni più dettagliate su quel partecipante. Nel secondo e terzo esempio, gli eventi più simili mantengono la struttura e gli attributi del grafo selezionato come punto di riferimento (hanno gli stessi tipi di partecipanti e ricoprono lo stesso ruolo nell'interazione), suggerendo entità correlate o aggiuntive. Di conseguenza, si

[2] Un tipo di evento all'interno di uno schema può specificare la cardinalità di partecipanti prevista, le cui istanze potrebbero essere a volte opzionali o molteplici. Di conseguenza gli eventi dello stesso tipo potrebbero avere strutturre diverse

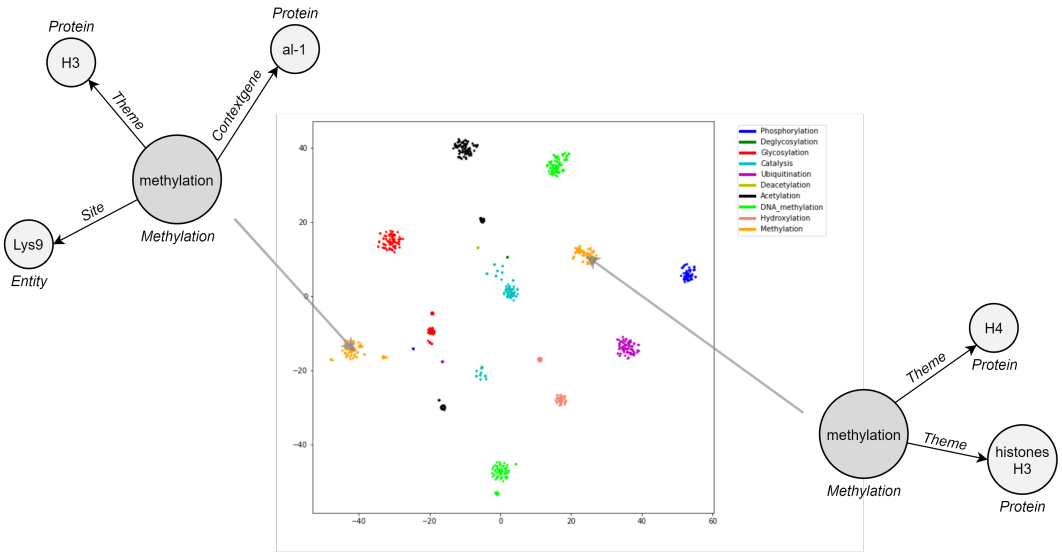


Figura 4.3: Mostriamo gli Embedding dei 1,002 eventi biomedici provenienti dal dataset EPI11, proiettati all'interno di uno spazio 2D con t-SNE. I grafi evento appartenenti allo stesso tipo sono rappresentati da embedding vicini tra loro. Nel caso di differenze strutturali o semantiche possono formarsi cluster per lo stesso tipo di evento come, mostrato a livello qualitativo, per gli eventi di Methylation.

può assumere che eventi simili facciano riferimento a eventi con forti relazioni semantiche piuttosto che solo a relazioni di identità.

4.5 Cross-graph Attention

L'obiettivo è quello di capire come due grafi evento biomedici siano messi in relazione tra loro da DDEGK. Dato che non esiste uno standard per le corrispondenze tra due gruppi di nodi, non possiamo valutare quantitativamente la bontà dei legami di similarità cross-graph. Forniamo però un esempio qualitativo nella Figura 4.5 – che non è da intendere come una valutazione finale del sistema – che mostra come DDEGK tipicamente dia maggiore attenzione alle entità o ai trigger corretti ignorando quelli non significativi.

4.6 Semantic Textual Similarity

Il linguaggio naturale è fortemente ambiguo, esistono molti modi per esprimere lo stesso concetto, frequenti fenomeni linguistici ad alto livello, e retroscena

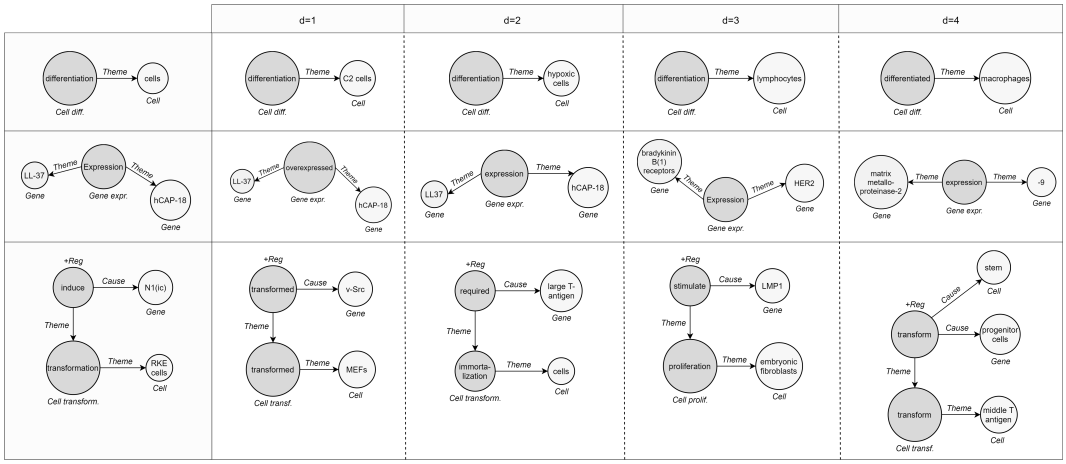


Figura 4.4: Query su grafi di eventi biomedici di tipi e dimensioni differenti. In cascuna demo, la prima colonna mostra la query e gli altri il grado di similarità dei grafi ottenuti. DDEGK è in grado di restituire correttamente eventi simili per struttura o per la semantica di nodi/archi ed è capace di gestire sinonimi e acronimi grazie a SciBERT.

non espressi. L'organizzazione superficiale di una frase risulta quasi irrilevante per l'identificazione del vero e più profondo contenuto semantico, determinato invece da techtogrammatics [141]. Gli eventi ci permettono di rimuovere il rumore e di focalizzarci solo sulla conoscenza relazionale non ambigua che coinvolge le entità rilevanti. Per addentrarci nella ricerca che vede l'utilizzo degli eventi in sostituzione a frasi all'interno di applicazioni di NLP, introduciamo un esperimento sulla Semantic Textual Similarity.
Consideriamo tre dataset per la sentence similarity in ambito biomedico BIOSSES [142], MedSTS [143], e CTR [144], per un totale di 271 coppie di frasi (499 singole frasi) e punteggi normalizzati nel range $[0, 1]$. Facciamo processing di ciascuna frase con sette modelli di DeepEventMine preaddestrati (GE11, EPI11, ID11, MLEE, GE13, CG13, PC13) [3], che sono in grado di estrarre 184, 8, 142, 331, 198, 392, e 236 eventi, rispettivamente. 324 frasi hanno almeno un evento (264 con più di uno), ottenendo un dataset di 127 coppie di frasi (80:20 rapporto train-test), eventi e punteggi di similarità. Successivamente, convertiamo gli eventi in grafi evento e calcoliamo gli embedding delle entità come descritto nelle Sezioni 4.1.1 e 2. Rimuovendo i duplicati delle istanze, passiamo da 1491 grafi evento a 890. Quindi, eseguiamo DDEGK sulla popolazione dei grafi evento (128 prototipi, 4 come dimensione degli embedding dei nodi, 3 layers di encoding, 300 epoche di encoding e scoring, learning rate pari a 10^{-1}, [7,7,7] γ^v, τ_v, τ_e coefficienti della loss), facendo pooling calcolando la media nel caso di molteplici eventi all'interno di una frase. Infine, calcoliamo embedding di frasi

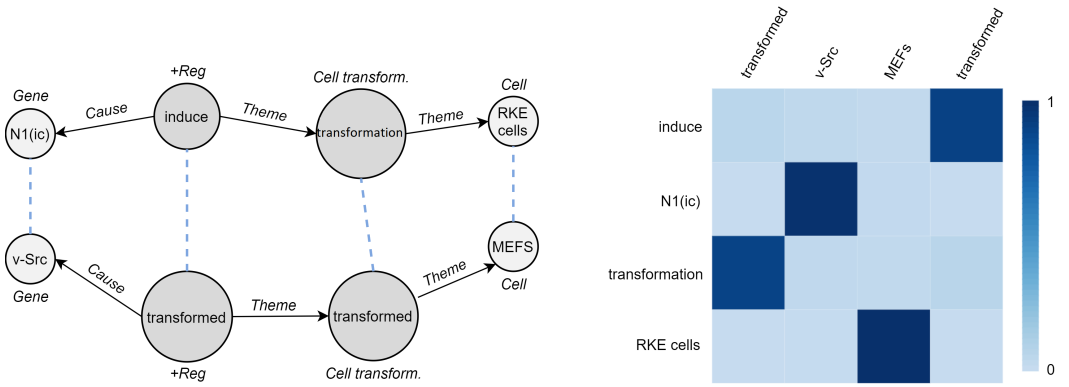


Figura 4.5: Esempio qualitativo di cross-graph attention, che mette a confronto due eventi biomedici di CG13. I pesi dell'attention vengono visualizzati mediante una heatmap, mentre i collegamenti più forti tra nodi sono direttamente riportati sui grafi evento (linee blu tratteggiate).

semanticamente ricchi di significato con SBERT [145] (`all-mpnet-base-v2` di HuggingFace[3]). Definiamo un semplice modello di regressione composto da due fully connected linear layers (hidden size 1000) con una sigmoide, utilizzando l'ottimizzatore Adam, regolarizzazione L1, 1000 epoche e un learning rate pari a 10^{-4}. Lo addestriamo su tre configurazioni di input: (i) solo frasi (SBERT), (ii) solo eventi (DDEGK), (iii) frasi + eventi (concatenati esattamente in questo ordine). Nel corso degli esperimenti, registriamo un errore quadratico medio sul test set di 0.1379 per (i), 0.1216 per (ii), e 0.1598 per (iii). Questi risultati avvalorano l'efficacia degli eventi nel cogliere la conoscenza essenziale menzionata all'interno dei documenti di testo. Gli embedding di eventi ottenuti con DDEGK portano a migliori risultati sul task di Sentence Text Similarity rispetto agli approcci basati su sequenze di token che sono lo stato dell'arte, utilizzando dimensioni sei volte più piccole.

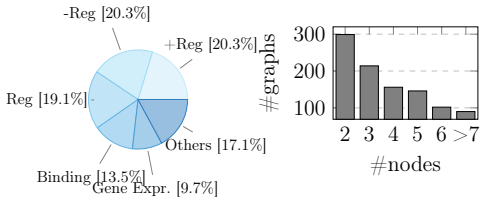
[3] `https://huggingface.co/sentence-transformers/stsb-mpnet-base-v2`

Capitolo 5

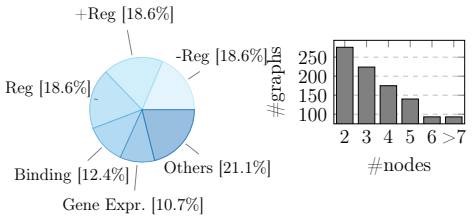
Conclusioni

In questa tesi è stata presentata Deep Divergence Event Graph Kernels, una tecnica non supervisionata per l'apprendimento a livello di grafo di rappresentazioni di eventi biomedici e delle loro similarità. Il nostro metodo confronta i grafi evento con un insieme di grafi prototipo senza richiedere specifiche feature. Basandosi sulla divergenza strutturale e semantica, misurata anche da SciBERT, è in grado di apprendere degli embedding indipendenti da un task e un graph kernel su di essi. Utilizzando un meccanismo di isomorphic attention, allineiamo i nodi di due grafi senza il bisogno di conoscere una corrispondenza tra gli indici dei nodi, ma semplicemente mantenendo la struttura e la coerenza semantica degli attributi dei nodi e degli archi (sia discreti che continui). La nostra analisi sperimentale mostra che, nonostante sia allenato solo su archi di grafi, le rappresentazioni apprese sono in grado di codificare numerose informazioni locali e globali permettendo di ottenere un potente spazio di embedding. In più, quando gli embedding degli eventi appresi sono utilizzati come feature in task come la classificazione e il clustering di grafi, essi risultano superiori o competitivi con quelli prodotti dalle soluzioni che sono lo stato dell'arte (anche di quelle supervisionate). Inoltre, l'eccellente capacità di riconoscere la similarità semantica tra frasi biomediche sottolinea come l'introduzione di feature dense e compatte basate sugli eventi all'interno dei sistemi NLP possa essere una promettente area di ricerca. Infine, oltre a essere espressivi, i modelli DDEGK sono incredibilmente informativi. Il controllo sugli eventi prototipi e la presenza di pesi determinati dalla cross-graph attention garantisce un alto livello di comprensione delle corrispondenze tra i grafi evento, rendendo la loro similarità interpretabile.
In futuro, approfondiremo (i) l'identificazione automatica di prototipi ortogonali tra loro, (ii) la composizione di uno score di similarità rispetto a legami di similarità tra sotto-grafi anzichè singoli nodi, (iii) la creazione di spazi multimodali grazie alla combinazione di eventi e testo, (iv) l'integrazione di eventi e language

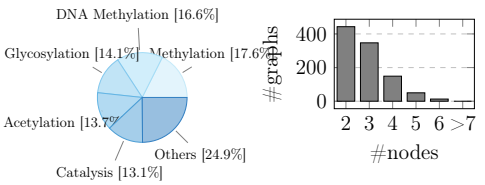
model, (v) l'applicazione a eventi non menzionati nella letteratura biomedica ma espressi da pazienti e infermieri all'interno di post [146, 147, 148, 149].



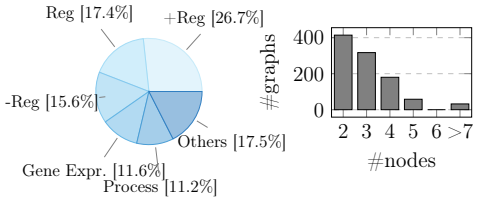
(a) ST09



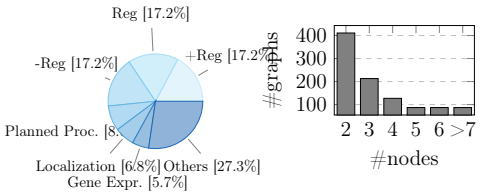
(b) GE11



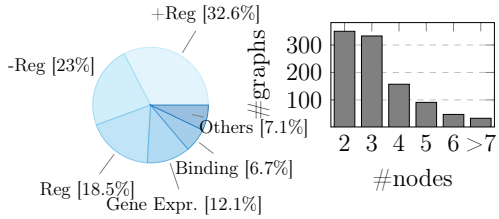
(c) EPI11



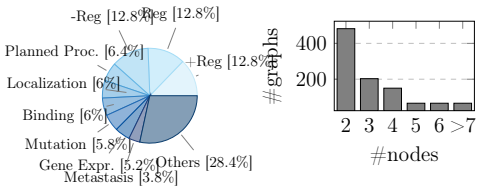
(d) ID11



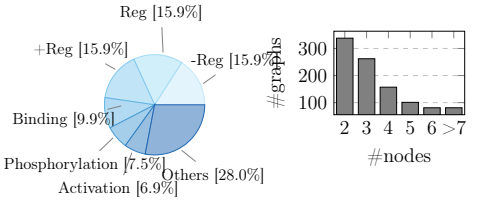
(e) MLEE



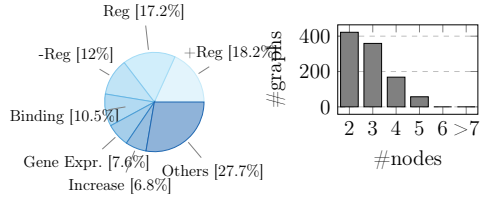
(f) GE13



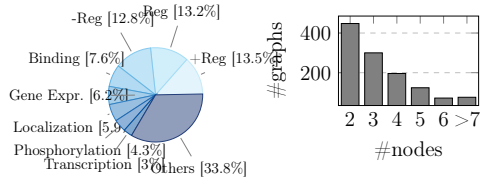
(g) CG13



(h) PC13



(i) GRO13



(j) ALL

Figura 5.1: Distribuzione dei tipi di evento (tipo del nodo radice nel caso di eventi innestati).

Ringraziamenti

Vorrei ringraziare il professore Gianluca Moro per avermi dato la possibilità di lavorare a questo progetto che ho trovato davvero interessante soprattutto per la connessione con l'ambito biomedico e le possibili applicazioni. Vorrei inoltre ringraziare il dottore Giacomo Frisoni per l'aiuto che mi ha dato durante tutto il percorso di tesi e per essersi sempre dimostrato pronto a rispondere ai miei dubbi in modo chiaro e adeguato.

Ringrazio le mie migliori amiche per i consigli che mi hanno dato e per essersi sempre preoccupate per me.

Un ringraziamento speciale a Elisa e Sara che hanno alleggerito sin dal primo giorno questa avventura e a Filippo e Denys che, nel corso di tre anni, hanno sempre trovato il tempo per confrontarsi con me.

Infine voglio ringraziare la mia famiglia che non mi ha mai fatto mancare niente, mi ha incoraggiata e ha continuato a ricordarmi cosa sia davvero importante.

Bibliografia

[1] Esther Landhuis. Scientific literature: Information overload. *Nature*, 535(7612):457–458, 2016.

[2] Sophia Ananiadou, Sampo Pyysalo, Jun'ichi Tsujii, and Douglas B Kell. Event extraction for systems biology by text mining the literature. *Trends in biotechnology*, 28(7):381–390, 2010.

[3] Hai-Long Trieu, Thy Thy Tran, Anh-Khoa Duong Nguyen, Anh Nguyen, Makoto Miwa, and Sophia Ananiadou. DeepEventMine: end-to-end neural nested event extraction from biomedical texts. *Bioinform.*, 36(19):4910–4917, 2020.

[4] Quoc-Chinh Bui and Peter M. A. Sloot. A robust approach to extract biomedical events from literature. *Bioinform.*, 28(20):2654–2661, 2012.

[5] Sampo Pyysalo, Tomoko Ohta, Makoto Miwa, Han-Cheol Cho, Junichi Tsujii, and Sophia Ananiadou. Event extraction across multiple levels of biological organization. *Bioinform.*, 28(18):575–581, 2012.

[6] Jin-Dong Kim, Tomoko Ohta, Sampo Pyysalo, Yoshinobu Kano, and Jun'ichi Tsujii. Overview of BioNLP'09 Shared Task on Event Extraction. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task, BioNLP@HLT-NAACL 2009 - Shared Task, Boulder, Colorado, USA, June 5, 2009*, pages 1–9. Association for Computational Linguistics, 2009.

[7] Jin-Dong Kim, Sampo Pyysalo, Tomoko Ohta, Robert Bossy, Ngan L. T. Nguyen, and Jun'ichi Tsujii. Overview of BioNLP Shared Task 2011. In Jun'ichi Tsujii, Jin-Dong Kim, and Sampo Pyysalo, editors, *Proceedings of BioNLP Shared Task 2011 Workshop, Portland, Oregon, USA, June 24, 2011*, pages 1–6. Association for Computational Linguistics, 2011.

[8] Claire Nédellec, Robert Bossy, Jin-Dong Kim, Jung-Jae Kim, Tomoko Ohta, Sampo Pyysalo, and Pierre Zweigenbaum. Overview of BioNLP

Shared Task 2013. In Claire Nédellec, Robert Bossy, Jin-Dong Kim, Jung-Jae Kim, Tomoko Ohta, Sampo Pyysalo, and Pierre Zweigenbaum, editors, *Proceedings of the BioNLP Shared Task 2013 Workshop, Sofia, Bulgaria, August 9, 2013*, pages 1–7. Association for Computational Linguistics, 2013.

[9] Yoshimasa Tsuruoka, Makoto Miwa, Kaisei Hamamoto, Jun'ichi Tsujii, and Sophia Ananiadou. Discovering and visualizing indirect associations between biomedical concepts. *Bioinform.*, 27(13):111–119, 2011.

[10] Sam Henry and Bridget T. McInnes. Literature Based Discovery: Models, methods, and trends. *J. Biomed. Informatics*, 74:20–32, 2017.

[11] Jari Björne, Filip Ginter, Sampo Pyysalo, Jun'ichi Tsujii, and Tapio Salakoski. Complex event extraction at pubmed scale. *Bioinform.*, 26(12):382–390, 2010.

[12] Makoto Miwa, Tomoko Ohta, Rafal Rak, Andrew Rowley, Douglas B. Kell, Sampo Pyysalo, and Sophia Ananiadou. A method for integrating and ranking the evidence for biochemical pathways by mining reactions from text. *Bioinform.*, 29(13):44–52, 2013.

[13] Tianran Zhang, Muhao Chen, and Alex A. T. Bui. Diagnostic Prediction with Sequence-of-sets Representation Learning for Clinical Events. In Martin Michalowski and Robert Moskovitch, editors, *Artificial Intelligence in Medicine - 18th International Conference on Artificial Intelligence in Medicine, AIME 2020, Minneapolis, MN, USA, August 25-28, 2020, Proceedings*, volume 12299 of *Lecture Notes in Computer Science*, pages 348–358. Springer, 2020.

[14] Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D. Manning. Modeling Biological Processes for Reading Comprehension. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Association for Computational Linguistics, 2014.

[15] Jari Björne and Tapio Salakoski. TEES 2.1: Automated Annotation Scheme Learning in the BioNLP 2013 Shared Task. In Claire Nédellec, Robert Bossy, Jin-Dong Kim, Jung-Jae Kim, Tomoko Ohta, Sampo Pyysalo, and Pierre Zweigenbaum, editors, *Proceedings of the BioNLP*

Shared Task 2013 Workshop, Sofia, Bulgaria, August 9, 2013, pages 16–25. Association for Computational Linguistics, 2013.

[16] Lvxing Zhu and Haoran Zheng. Biomedical event extraction with a novel combination strategy based on hybrid deep neural networks. *BMC Bioinform.*, 21(1):47, 2020.

[17] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, 2017.

[18] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021.

[19] Alexey Strokach, David Becerra, Carles Corbi-Verge, Albert Perez-Riba, and Philip M Kim. Fast and flexible protein design using deep graph neural networks. *Cell Systems*, 11(4):402–411, 2020.

[20] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*, pages 974–983. ACM, 2018.

[21] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph Neural Networks for Friend Ranking in Large-scale Social Platforms. In *WWW*, pages 2535–2546. ACM / IW3C2, 2021.

[22] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

[23] Zhaoping Xiong, Dingyan Wang, Xiaohong Liu, Feisheng Zhong, Xiaozhe Wan, Xutong Li, Zhaojun Li, Xiaomin Luo, Kaixian Chen, Hualiang Jiang, et al. Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism. *Journal of medicinal chemistry*, 63(16):8749–8760, 2019.

[24] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing Stars: On Approximating Graph Edit Distance. *Proc. VLDB Endow.*, 2(1):25–36, 2009.

[25] David B. Blumenthal and Johann Gamper. On the exact computation of the graph edit distance. *Pattern Recognit. Lett.*, 134:46–57, 2020.

[26] Guixiang Ma, Nesreen K. Ahmed, Theodore L. Willke, and Philip S. Yu. Deep graph similarity learning: a survey. *Data Min. Knowl. Discov.*, 35(3):688–725, 2021.

[27] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. Graph representation learning: A survey. *APSIPA Transactions on Signal and Information Processing*, 9, 2020.

[28] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3835–3845. PMLR, 2019.

[29] Rita Torres Sousa, Sara Silva, and Catia Pesquita. Supervised biomedical semantic similarity. *bioRxiv*, 2021.

[30] Rami Al-Rfou, Bryan Perozzi, and Dustin Zelle. DDGK: learning graph representations for deep divergence graph kernels. In Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia, editors, *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 37–48. ACM, 2019.

[31] Amr Ahmed, Nino Shervashidze, Shravan M. Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed large-scale natural graph factorization. In *WWW*, pages 37–48. International World Wide Web Conferences Steering Committee / ACM, 2013.

[32] Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning Graph Representations with Global Structural Information. In *CIKM*, pages 891–900. ACM, 2015.

[33] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric Transitivity Preserving Graph Embedding. In *KDD*, pages 1105–1114. ACM, 2016.

[34] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *WSDM*, pages 459–467. ACM, 2018.

[35] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. In *WWW*, pages 1509–1520. ACM, 2019.

[36] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: online learning of social representations. In *KDD*, pages 701–710. ACM, 2014.

[37] James Demmel, Ioana Dumitriu, and Olga Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108(1):59–91, 2007.

[38] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *WWW*, pages 1067–1077. ACM, 2015.

[39] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864. ACM, 2016.

[40] Zhilin Yang, Jie Tang, and William W. Cohen. Multi-Modal Bayesian Embeddings for Learning Social Knowledge Graphs. In *IJCAI*, pages 2287–2293. IJCAI/AAAI Press, 2016.

[41] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD*, pages 135–144. ACM, 2017.

[42] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. HARP: Hierarchical Representation Learning for Networks. In *AAAI*, pages 2127–2134. AAAI Press, 2018.

[43] Benedek Rozemberczki and Rik Sarkar. Fast sequence-based embedding with diffusion graphs. In *International Workshop on Complex Networks*, pages 99–107. Springer, 2018.

[44] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A. Alemi. Watch Your Step: Learning Node Embeddings via Graph Attention. In *NeurIPS*, pages 9198–9208, 2018.

[45] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. GEMSEC: graph embedding with self clustering. In *ASONAM*, pages 65–72. ACM, 2019.

[46] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network Representation Learning with Rich Text Information. In *IJCAI*, pages 2111–2117. AAAI Press, 2015.

[47] Seong-Jin Ahn and Myoung Ho Kim. Variational Graph Normalized Auto-Encoders. *CoRR*, abs/2108.08046, 2021.

[48] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR*, abs/1609.02907, 2016.

[49] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *CoRR*, abs/1710.10903, 2017.

[50] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NIPS*, pages 1024–1034, 2017.

[51] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching Node Embeddings for Graph Similarity. In *AAAI*, pages 2429–2435. AAAI Press, 2017.

[52] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *Int. J. Comput. Vis.*, 40(2):99–121, 2000.

[53] Antoine J.-P. Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Graph Classification with 2D Convolutional Neural Networks. In *ICANN (Workshop)*, volume 11731 of *Lecture Notes in Computer Science*, pages 578–593. Springer, 2019.

[54] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine Learning on Graphs: A Model and Comprehensive Taxonomy. *CoRR*, abs/2005.03675, 2020.

[55] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[56] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NIPS*, pages 2224–2232, 2015.

[57] Asma Atamna, Nataliya Sokolovska, and Jean-Claude Crivello. SPI-GCN: a simple permutation-invariant graph convolutional network. 2019.

[58] Jiawei Zhang. Graph Neural Distance Metric Learning with Graph-Bert. *CoRR*, abs/2002.03427, 2020.

[59] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*, pages 4805–4815, 2018.

[60] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4438–4445. AAAI Press, 2018.

[61] Hongyang Gao and Shuiwang Ji. Graph U-Nets. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2083–2092. PMLR, 2019.

[62] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-Attention Graph Pooling. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 2019.

[63] Amir Hosein Khas Ahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. Memory-Based Graph Networks. In *ICLR*. OpenReview.net, 2020.

[64] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2014–2023. JMLR.org, 2016.

[65] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B. Aditya Prakash. Distributed Representations of Subgraphs. In *ICDM Workshops*, pages 111–117. IEEE Computer Society, 2017.

[66] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning Distributed Representations of Graphs. *CoRR*, abs/1707.05005, 2017.

[67] Shengchao Liu, Mehmet Furkan Demirel, and Yingyu Liang. N-Gram Graph: Simple Unsupervised Representation for Graphs, with Applications to Molecules. In *NeurIPS*, pages 8464–8476, 2019.

[68] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In *WSDM*, pages 384–392. ACM, 2019.

[69] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[70] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *J. Internet Serv. Appl.*, 1(1):19–30, 2010.

[71] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. NetSimile: A Scalable Approach to Size-Independent Network Similarity. *CoRR*, abs/1209.2684, 2012.

[72] Christos Faloutsos, Danai Koutra, and Joshua T. Vogelstein. DELTACON: A principled massive-graph similarity function. In *SDM*, pages 162–170. SIAM, 2013.

[73] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Network similarity via multiple social theories. In *ASONAM*, pages 1439–1440. ACM, 2013.

[74] Liwei Cai and William Yang Wang. KBGAN: Adversarial Learning for Knowledge Graph Embeddings. In *NAACL-HLT*, pages 1470–1480. Association for Computational Linguistics, 2018.

[75] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*, pages 2787–2795, 2013.

[76] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*, pages 1112–1119. AAAI Press, 2014.

[77] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *ICLR (Poster)*. OpenReview.net, 2019.

[78] Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning Hierarchy-Aware Knowledge Graph Embeddings for Link Prediction. In *AAAI*, pages 3065–3072. AAAI Press, 2020.

[79] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A Three-Way Model for Collective Learning on Multi-Relational Data. In *ICML*, pages 809–816. Omnipress, 2011.

[80] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *ICLR (Poster)*, 2015.

[81] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic Embeddings of Knowledge Graphs. In *AAAI*, pages 1955–1961. AAAI Press, 2016.

[82] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016.

[83] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *NIPS*, pages 926–934, 2013.

[84] Petar Ristoski and Heiko Paulheim. RDF2Vec: RDF Graph Embeddings for Data Mining. In *ISWC (1)*, volume 9981 of *Lecture Notes in Computer Science*, pages 498–514, 2016.

[85] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D Knowledge Graph Embeddings. In *AAAI*, pages 1811–1818. AAAI Press, 2018.

[86] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In *ESWC*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018.

[87] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. Representation Learning of Knowledge Graphs with Entity Descriptions. In *AAAI*, pages 2659–2665. AAAI Press, 2016.

[88] Quan Wang, Pingping Huang, Haifeng Wang, Songtai Dai, Wenbin Jiang, Jing Liu, Yajuan Lyu, Yong Zhu, and Hua Wu. CoKE: Contextualized Knowledge Graph Embedding. *CoRR*, abs/1911.02168, 2019.

[89] Linmei Hu, Mengmei Zhang, Shaohua Li, Jinghan Shi, Chuan Shi, Cheng Yang, and Zhiyuan Liu. Text-Graph Enhanced Knowledge Graph Representation Learning. *Frontiers Artif. Intell.*, 4:697856, 2021.

[90] Martin Grohe. word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data. In *PODS*, pages 1–16. ACM, 2020.

[91] Lingfei Wu, Ian En-Hsu Yen, Fangli Xu, Pradeep Ravikumar, and Michael Witbrock. D2KE: From Distance to Kernel and Embedding. *CoRR*, abs/1802.04956, 2018.

[92] Horst Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognit. Lett.*, 1(4):245–253, 1983.

[93] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognit. Lett.*, 19(3-4):255–259, 1998.

[94] Yongjiang Liang and Peixiang Zhao. Similarity Search in Graph Databases: A Multi-Layered Indexing Approach. In *ICDE*, pages 783–794. IEEE Computer Society, 2017.

[95] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast Suboptimal Algorithms for the Computation of Graph Edit Distance. In *SSPR/-SPR*, volume 4109 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 2006.

[96] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.*, 27(7):950–959, 2009.

[97] Évariste Daller, Sébastien Bougleux, Benoit Gaüzère, and Luc Brun. Approximate Graph Edit Distance by Several Local Searches in Parallel. In *ICPRAM*, pages 149–158. SciTePress, 2018.

[98] Yunsheng Bai, Hao Ding, Yizhou Sun, and Wei Wang. Convolutional Set Matching for Graph Similarity. *CoRR*, abs/1810.10866, 2018.

[99] Sofia Ira Ktena, Sarah Parisot, Enzo Ferrante, Martin Rajchl, Matthew C. H. Lee, Ben Glocker, and Daniel Rueckert. Metric learning with spectral graph convolutions on brain connectivity networks. *NeuroImage*, 169:431–442, 2018.

[100] Guixiang Ma, Nesreen K. Ahmed, Theodore L. Willke, Dipanjan Sengupta, Michael W. Cole, Nicholas B. Turk-Browne, and Philip S. Yu. Deep Graph Similarity Learning for Brain Data Analysis. In *CIKM*, pages 2743–2751. ACM, 2019.

[101] Shen Wang, Zhengzhang Chen, Xiao Yu, Ding Li, Jingchao Ni, Lu-An Tang, Jiaping Gui, Zhichun Li, Haifeng Chen, and Philip S. Yu. Heterogeneous Graph Matching Networks for Unknown Malware Detection. In *IJCAI*, pages 3762–3770. ijcai.org, 2019.

[102] Yunsheng Bai, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. Unsupervised inductive graph-level representation learning via graph-graph proximity. *arXiv preprint arXiv:1904.01098*, 2019.

[103] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-Path Kernels on Graphs. In *ICDM*, pages 74–81. IEEE Computer Society, 2005.

[104] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, volume 5 of *JMLR Proceedings*, pages 488–495. JMLR.org, 2009.

[105] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

[106] Risi Kondor and Horace Pan. The Multiscale Laplacian Graph Kernel. In *NIPS*, pages 2982–2990, 2016.

[107] Pinar Yanardag and S. V. N. Vishwanathan. Deep Graph Kernels. In *KDD*, pages 1365–1374. ACM, 2015.

[108] Wei Xiang and Bang Wang. A Survey of Event Extraction From Text. *IEEE Access*, 7:173111–173137, 2019.

[109] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep Learning for Event-Driven Stock Prediction. In *IJCAI*, pages 2327–2333. AAAI Press, 2015.

[110] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Knowledge-Driven Event Embedding for Stock Prediction. In *COLING*, pages 2133–2142. ACL, 2016.

[111] Noah Weber, Niranjan Balasubramanian, and Nathanael Chambers. Event Representations With Tensor-Based Compositions. In *AAAI*, pages 4946–4953. AAAI Press, 2018.

[112] Xiao Ding, Kuo Liao, Ting Liu, Zhongyang Li, and Junwen Duan. Event Representation Learning Enhanced with External Commonsense Knowledge. In *EMNLP/IJCNLP (1)*, pages 4893–4902. Association for Computational Linguistics, 2019.

[113] Pride Kavumba, Naoya Inoue, and Kentaro Inui. Exploring supervised learning of hierarchical event embedding with poincaré embeddings. 2019.

[114] Huan Gui, Jialu Liu, Fangbo Tao, Meng Jiang, Brandon Norick, Lance M. Kaplan, and Jiawei Han. Embedding Learning with Events in Heterogeneous Information Networks. *IEEE Trans. Knowl. Data Eng.*, 29(11):2428–2441, 2017.

[115] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 5(1):6, 2020.

[116] Iz Beltagy, Kyle Lo, and Arman Cohan. SciBERT: A Pretrained Language Model for Scientific Text. In *EMNLP/IJCNLP (1)*, pages 3613–3618. Association for Computational Linguistics, 2019.

[117] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *VLDB*, pages 518–529. Morgan Kaufmann, 1999.

[118] CSIRO's Data61. Stellargraph machine learning library. `https://github.com/stellargraph/stellargraph`, 2018.

[119] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, page 3125–3132. ACM, 2020.

[120] Antoine J-P Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Graph classification with 2d convolutional neural networks. In *International Conference on Artificial Neural Networks*, pages 578–593. Springer, 2019.

[121] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In *EMNLP*. Association for Computational Linguistics, 2019.

[122] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[123] Jin-Dong Kim, Ngan L. T. Nguyen, Yue Wang, Jun'ichi Tsujii, Toshihisa Takagi, and Akinori Yonezawa. The genia event and protein coreference tasks of the bionlp shared task 2011. *BMC Bioinform.*, 13(S-11):S1, 2012.

[124] Tomoko Ohta, Sampo Pyysalo, and Jun'ichi Tsujii. Overview of the Epigenetics and Post-translational Modifications (EPI) task of BioNLP Shared Task 2011. In Jun'ichi Tsujii, Jin-Dong Kim, and Sampo Pyysalo, editors, *Proceedings of BioNLP Shared Task 2011 Workshop, Portland, Oregon, USA, June 24, 2011*, pages 16–25. Association for Computational Linguistics, 2011.

[125] Sampo Pyysalo, Tomoko Ohta, Rafal Rak, Daniel E. Sullivan, Chunhong Mao, Chunxia Wang, Bruno W. S. Sobral, Jun'ichi Tsujii, and Sophia Ananiadou. Overview of the Infectious Diseases (ID) task of BioNLP Shared Task 2011. In Jun'ichi Tsujii, Jin-Dong Kim, and Sampo Pyysalo, editors, *Proceedings of BioNLP Shared Task 2011 Workshop, Portland, Oregon, USA, June 24, 2011*, pages 26–35. Association for Computational Linguistics, 2011.

[126] Jin-Dong Kim, Yue Wang, and Yasunori Yamamoto. The Genia Event Extraction Shared Task, 2013 Edition - Overview. In Claire Nédellec, Robert Bossy, Jin-Dong Kim, Jung-Jae Kim, Tomoko Ohta, Sampo Pyysalo, and Pierre Zweigenbaum, editors, *Proceedings of the BioNLP Shared Task 2013 Workshop, Sofia, Bulgaria, August 9, 2013*, pages 8–15. Association for Computational Linguistics, 2013.

[127] Sampo Pyysalo, Tomoko Ohta, and Sophia Ananiadou. Overview of the Cancer Genetics (CG) task of BioNLP Shared Task 2013. In Claire Nédellec, Robert Bossy, Jin-Dong Kim, Jung-Jae Kim, Tomoko Ohta, Sampo Pyysalo, and Pierre Zweigenbaum, editors, *Proceedings of the BioNLP Shared Task 2013 Workshop, Sofia, Bulgaria, August 9, 2013*, pages 58–66. Association for Computational Linguistics, 2013.

[128] Tomoko Ohta, Sampo Pyysalo, Rafal Rak, Andrew Rowley, Hong-Woo Chun, Sung-Jae Jung, Sung-Pil Choi, Sophia Ananiadou, and Jun'ichi

Tsujii. Overview of the Pathway Curation (PC) task of BioNLP Shared Task 2013. In Claire Nédellec, Robert Bossy, Jin-Dong Kim, Jung-Jae Kim, Tomoko Ohta, Sampo Pyysalo, and Pierre Zweigenbaum, editors, *Proceedings of the BioNLP Shared Task 2013 Workshop, Sofia, Bulgaria, August 9, 2013*, pages 67–75. Association for Computational Linguistics, 2013.

[129] Jung-jae Kim, Xu Han, Vivian Lee, and Dietrich Rebholz-Schuhmann. GRO Task: Populating the Gene Regulation Ontology with events and relations. In Claire Nédellec, Robert Bossy, Jin-Dong Kim, Jung-Jae Kim, Tomoko Ohta, Sampo Pyysalo, and Pierre Zweigenbaum, editors, *Proceedings of the BioNLP Shared Task 2013 Workshop, Sofia, Bulgaria, August 9, 2013*, pages 50–57. Association for Computational Linguistics, 2013.

[130] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[131] Hannu Toivonen, Ashwin Srinivasan, Ross D. King, Stefan Kramer, and Christoph Helma. Statistical Evaluation of the Predictive Toxicology Challenge 2000-2001. *Bioinform.*, 19(10):1183–1193, 2003.

[132] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alexander J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. In *ISMB (Supplement of Bioinformatics)*, pages 47–56, 2005.

[133] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[134] Scott Freitas, Yuxiao Dong, Joshua Neil, and Duen Horng Chau. A Large-Scale Database for Graph Representation Learning. *CoRR*, abs/2011.07682, 2020.

[135] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinform.*, 36(4):1234–1240, 2020.

[136] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine Learning in Python. *CoRR*, abs/1201.0490, 2012.

[137] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. In *NIPS*, pages 601–608. MIT Press, 2001.

[138] Makoto Miwa, Sampo Pyysalo, Tomoko Ohta, and Sophia Ananiadou. Wide coverage biomedical event extraction using multiple partially overlapping corpora. *BMC Bioinform.*, 14:175, 2013.

[139] Satu Elisa Schaeffer. Graph clustering. *Comput. Sci. Rev.*, 1(1):27–64, 2007.

[140] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[141] Petr Sgall, Eva Hajicová, Eva Hajicová, Jarmila Panevová, and Jarmila Panevova. *The meaning of the sentence in its semantic and pragmatic aspects*. Springer Science & Business Media, 1986.

[142] Gizem Soganciogl u, Hakime Öztürk, and Arzucan Özgür. BIOSSES: a semantic sentence similarity estimation system for the biomedical domain. *Bioinform.*, 33(14):i49–i58, 2017.

[143] Yanshan Wang, Naveed Afzal, Sunyang Fu, Liwei Wang, Feichen Shen, Majid Rastegar-Mojarad, and Hongfang Liu. MedSTS: a resource for clinical semantic textual similarity. *Language Resources and Evaluation*, 54(1):57–72, 2020.

[144] Oscar Lithgow-Serrano, Socorro Gama-Castro, Cecilia Ishida-Gutiérrez, Citlalli Mejía-Almonte, Víctor H. Tierrafría, Sara Martínez-Luna, Alberto Santos-Zavaleta, David A. Velázquez-Ramírez, and Julio Collado-Vides. Similarity corpus on microbial transcriptional regulation. *J. Biomed. Semant.*, 10(1):8:1–8:14, 2019.

[145] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP/IJCNLP (1)*, pages 3980–3990. Association for Computational Linguistics, 2019.

[146] Giacomo Frisoni, Gianluca Moro, and Antonella Carbonaro. Learning Interpretable and Statistically Significant Knowledge from Unlabeled Corpora of Social Text Messages: A Novel Methodology of Descriptive Text Mining. In *DATA*, pages 121–132. SciTePress, 2020.

[147] Giacomo Frisoni and Gianluca Moro. Phenomena Explanation from Text: Unsupervised Learning of Interpretable and Statistically Significant Knowledge. In *DATA (Revised Selected Papers)*, volume 1446 of *Communications in Computer and Information Science*, pages 293–318. Springer, 2020.

[148] Giacomo Frisoni, Gianluca Moro, and Antonella Carbonaro. Unsupervised Descriptive Text Mining for Knowledge Graph Learning. In *KDIR*, pages 316–324. SCITEPRESS, 2020.

[149] Giacomo Frisoni, Gianluca Moro, and Antonella Carbonaro. Towards Rare Disease Knowledge Graph Learning from Social Posts of Patients. In *RIIFORUM*, pages 577–589. Springer, 2020.