



UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Percorso di studio
Scienze e Tecnologie Informatiche

ELABORATO FINALE

Implementazione e valutazione di algoritmi per l'identificazione di comunità

Supervisore

Alberto Montresor

Laureando

Stefano Leonardi

Anno accademico 2017/2018

Ringraziamenti

Ringrazio tutti coloro che in questi anni mi sono stati vicini, mi hanno motivato a crescere e a non mollare anche quando tutto è andato storto. Grazie a chi ha creduto in me, fino a che l'obiettivo, da lontano e difficile, è diventato sempre più vicino e tangibile.

Ora che ho conquistato questo traguardo sono certo che sarà motivo di gioia ed orgoglio per i miei genitori, i parenti e gli amici tutti. Un grazie a tutti gli amici con cui ho condiviso la bellissima esperienza del mondo circense, da cui ho imparato a vivere col sorriso qualsiasi situazione. Altrettanto ringrazio gli amici di più vecchia data, che mi conoscono e m'accompagnano da molti anni.

Sono convinto che mi sarete tutti vicini anche per i progetti futuri a cui già sto pensando.

Un doveroso ringraziamento va poi a chi ha reso possibile questo lavoro di tesi, il dottorando Nasrullah Sheikh per la sua guida e al prof. Alberto Montresor per la pazienza e i preziosi consigli che mi son stati di grande aiuto.

Indice

1	Sommario	4
1.1	Introduzione	4
1.2	Cosa sono i grafi e cosa le comunità	4
1.3	Come si valutano le comunità	4
1.4	I cambiamenti apportati	5
1.5	Esperimenti	5
1.6	Com'è strutturata la tesi	6
2	Implementazione	7
2.1	Come funzionano le dinamiche di CNRL	7
2.2	Introduzione degli attributi	8
2.3	Grafo degli attributi	8
2.3.1	Grafo bipartito e ipergrafo, origine dei nomi	10
2.4	Creazione del grafo - esempio	11
3	Metodi di valutazione	13
3.1	Modularità con massimi (mod_withMax)	13
3.2	Modularità con sovrapposizione (mod_overlap)	14
3.3	Modularità modificata	16
4	Esperimenti	17
4.1	Origini dei grafi	17
4.2	Modello di un grafo	18
4.3	Link Prediction	19
4.3.1	Dinamiche di funzionamento	19
4.3.2	Esempio	20
4.3.3	Applicazioni	21
4.4	Classificazione dei nodi	23
4.4.1	Dinamiche di funzionamento	23
4.4.2	Esempio	24
4.4.3	Applicazioni	25
4.5	Valutazione dell'individuazione di comunità	26
4.5.1	Confronto numero comunità	27
4.5.2	Metriche di valutazione	27
4.5.3	Confronto metodi d'elaborazione	28
4.5.4	Confronto criteri di valutazione	29
5	Conclusioni	30
	Bibliografia	30

Elenco delle figure

2.1	Due nodi condividono un attributo, ecco come possono essere legati, 1-nessun arco, 2-ipergrafo (completo), 3-grafo bipartito (a stella)	9
2.2	Quattro nodi condividono un attributo, ecco come possono essere legati, 1-nessun arco, 2-ipergrafo (completo), 3-grafo bipartito (a stella)	9
2.3	Rappresentazione della struttura originaria del grafo	11
2.4	Rappresentazione dell'ipergrafo generato	12
2.5	Rappresentazione del grafo bipartito generato	12
4.1	Son visualizzati i nodi d'un grafo rappresentati da un modello che dispone di due dimensioni, mostrato in Tabella 4.2	18
4.2	Link Prediction con tre sezioni, $F = 3$	21
4.3	Andamento della metrica di Common Neighbors, su tre grafi distinti attraverso differenti percentuali di selezione	22
4.4	Andamento del grafo dolphins, su diverse metriche, attraverso differenti percentuali di selezione	22
4.5	Andamento del grafo Cora , con 3 differenti algoritmi di creazione del modello e 3 diverse interpretazioni iniziali, valutato mediante l'accuratezza	25
4.6	Andamento del grafo Citeseer , con 3 differenti algoritmi di creazione del modello e 3 diverse interpretazioni iniziali, valutato mediante l'accuratezza	26
4.7	Accuratezza rappresentativa al variare del numero delle comunità da generare	27
4.8	Prestazioni dei diversi procedimenti d'elaborazione del modello al variare del numero di comunità da ricercare (grafo Cora)	28

Elenco delle tabelle

2.1	Ad ogni nodo è associata la lista degli attributi che possiede	11
2.2	Ad ogni attributo è associata la lista dei nodi a cui appartiene	11
4.1	Per ogni grafo sono indicate le sue caratteristiche principali	17
4.2	Tabella rappresentativa del modello dei nodi in Figura 4.1	18
4.3	Tabella riassuntiva del procedimento di Link Prediction	20
4.4	Viene rappresentata una matrice di confusione	24
4.5	Significato delle sigle della matrice di confusione	24
4.6	Tabella riassuntiva del procedimento di classificazione dei vertici	24
4.7	Confronto fra i metodi di modularità spiegati, partendo dal grafo normale con 10 comunità	28
4.8	Misurazione delle comunità, generate attraverso due differenti metodi d’elaborazione, analizzate su due tipologie di valutazione	29

1 Sommario

Questa tesi va a ripercorrere il lavoro eseguito da Stefano Leonardi durante il tirocinio svolto all'interno dell'università, con la supervisione del dottorando Nasrullah Sheikh, per il professore Alberto Montresor.

1.1 Introduzione

Il tirocinio è basato sul documento intitolato "Community-enhanced Network Representation Learning for Network Analysis"[20] abbreviato dalla sigla **CNRL**. Questo testo propone un innovativo metodo per l'individuazione delle comunità all'interno dei grafi. Lo scopo di questo lavoro è quello di replicare i risultati ivi proposti. Una volta compreso il codice che ne è la base, abbiamo cercato di migliorare le prestazioni dell'algoritmo, anche sfruttando elementi non originariamente considerati.

L'algoritmo proposto nell'articolo utilizza diverse tecniche, alcune delle quali dettagliatamente spiegate in altri documenti:

- "DeepWalk: Online Learning of Social Representations"[15]
- "node2vec: Scalable Feature Learning for Networks"[5]
- "How exactly does word2vec work?"[13]

Segue la spiegazione del perché l'individuazione di comunità è una tecnica così importante.

1.2 Cosa sono i grafi e cosa le comunità

La base sono i grafi, una particolare struttura dati composta da **nodi/vertici** V (dove $n = |V|$) e **archi/bordi/lati** E (dove $m = |E|$). I primi possono rappresentare un'entità eventualmente anche con l'aiuto di attributi. Diversamente gli archi sono dei collegamenti che legano due nodi. Possono essere interamente **orientati/diretti** o **non orientati/indiretti**. Questa struttura dati permette di rappresentare molte situazioni e per tale motivo è importante.

Lo scopo dell'individuazione delle comunità è riconoscere dei gruppi di nodi simili fra loro.

Si fa notare che i nodi risultano essere simili per diverse ragioni. Possono avere le stesse caratteristiche strutturali in quanto sono dei cardini che legano molti altri elementi, oppure punti isolati da tutto il resto, o più semplicemente valori analoghi sugli attributi.

Gruppi di nodi simili possono essere gestiti in maniera omogenea; questo permette di sfruttarne le peculiarità. La versatilità d'applicazione dei grafi conferisce a questi algoritmi una grande importanza.

1.3 Come si valutano le comunità

Date due partizioni differenti su un grafo, è necessario capire quale di queste è la migliore. Se il grafo è molto piccolo, tanto da poterlo comprendere guardando unicamente una sua rappresentazione grafica, e le comunità individuate sono poche, si potrebbe visivamente decidere anche ad occhio qual è la partizione migliore.

Se invece le comunità individuate sono tante e il grafo non è piccolo, allora non è facile effettuare una scelta e ancor meno riuscire a giustificarla. Esistono metodi formali per analizzare e decidere quale partizione scegliere: questi metodi prendono il nome di **modularità**.

Le **partizioni** vengono così definite:

- Una partizione è un insieme di comunità

- Non tutte le comunità hanno le stesse dimensioni, ossia lo stesso numero di nodi che vi appartengono
- Le comunità non hanno né una dimensione massima né una dimensione minima. Si può considerare che la loro dimensione cada nell'intervallo $[1, n]$. Il limite inferiore è 1 perché devono contenere almeno un elemento, mentre il limite superiore è n perché non possono contenere più nodi di quelli esistenti nel grafo, che sono in numero di n
- È possibile che un nodo non appartenga a nessuna comunità
- Un nodo del grafo può appartenere a più di una comunità, potenzialmente anche a tutte quelle presenti. Se questo accade significa che c'è una sovrapposizione di comunità

La metrica di valutazione di una partizione è la modularità (il cui simbolo è Q). Questo nome può essere associato a diverse metriche. A seconda di cosa si cerca si avrà interesse a valorizzare alcuni aspetti e penalizzarne altri. Per tale motivo la modularità è solo il nome usato per indicare il valore dato dalla valutazione di una partizione.

Durante il tirocinio sono stati adottati tre metodi differenti: **mod_withMax**, **mod_overlap** e **modularità modificata**. I primi due nomi non sono ufficiali in quanto sono stati scelti al solo scopo di riconoscerli, mentre l'ultimo, modularità modificata, è quello utilizzato nell'articolo su CNRL [20]. I dettagli sul funzionamento e sulla motivazione del perché sono state scelte sono all'interno dell'apposito capitolo sulle metriche di valutazione (Capitolo 3).

1.4 I cambiamenti apportati

Per calcolare una partizione, il codice di CNRL utilizza un sistema di visite random (casuali) e con questo ricostruisce le possibili somiglianze fra i nodi. Non volendo toccare questa sezione, le modifiche da noi apportate si inseriscono all'interno della creazione dei cammini.

Una cammino è definito come una sequenza di nodi. Preso un elemento qualsiasi fra questi deve esser possibile arrivare all'elemento successivo attraversando un solo arco. I due elementi devono essere direttamente connessi, almeno nel senso di percorrenza.

L'algoritmo genera x cammini, ognuno di lunghezza massima l , sul grafo, dove $x = n \cdot w$:

- n : è il numero di nodi del grafo
- w : è il numero di cammini che si fanno partire da ogni singolo nodo
- l : è la lunghezza massima di ogni cammino

Con questi dati l'algoritmo di CNRL calcola la suddivisione in comunità. Si può notare come tutti questi dati vengano estratti esclusivamente dalla struttura del grafo, ossia da come nodi ed archi sono interconnessi.

L'intuizione sfruttata per modificare l'algoritmo si basa sull'esistenza di altri dati non utilizzati, ossia gli attributi dei nodi. Tramite la creazione di nuovi particolari cammini è possibile far risultare vicini due nodi in precedenza lontani, in quanto non direttamente connessi. Risultano vicini perché questi nodi condividono uno o più attributi. L'introduzione dei nuovi dati porta ad un radicale cambiamento della suddivisione in comunità.

L'idea di introdurre gli attributi per migliorare l'efficienza dell'algoritmo è sorta grazie agli articoli:

- "Attributed social network embedding"[10]
- "Network Representation Learning with Rich Text Information"[21]

1.5 Esperimenti

Gli esperimenti che sono stati eseguiti nel corso del tirocinio si dividono in tre principali gruppi. Ognuno di questi è dedicato ad un diverso algoritmo.

Le tre aree sono:

- **Link Prediction** : Mediante la creazione di archi verosimili e potenzialmente reali, ma che non appaiono in una particolare selezione denominata "test", va a verificare se sono più coerenti gli archi generati casualmente rispetto a quelli reali.
Per assegnare un valore comparabile ad un arco, si guarda la relazione, che lega i due nodi agli estremi del collegamento, valutata mediante apposite funzioni di similarità.
Gli archi dei due insiemi sono in seguito accoppiati e, sulla base del valore più alto, si assegna un punteggio. Calcolando il punteggio per ogni coppia si andrà a rappresentare l'attendibilità per predire in maniera corretta un nuovo arco.
- **Classificazione dei nodi** è una tecnica che necessita di una rappresentazione univoca dei vertici. Ognuno è definito come un punto all'interno di uno spazio a 64 dimensioni e se ne conosce la classe d'appartenenza. Questo permette di predire per un qualsiasi nuovo nodo rappresentato nella stessa forma degli altri, a quale classe andrà ad appartenere.
I test svolti analizzano con quale percentuale il classificatore, ossia la funzione che assegna la classe, riesce a predire in maniera esatta la classe di un nodo.
- La **valutazione dell'individuazione di comunità** si occupa in prima istanza di creare le comunità di nodi simili fra loro, secondo determinate caratteristiche. I metodi della modularità valutano se le partizioni individuate sono ben costruite oppure no.

1.6 Com'è strutturata la tesi

Il Capitolo 2 sull'implementazione spiega le dinamiche base dell'algoritmo principale, da cui è partito tutto il lavoro. Poi viene descritta l'idea per un incremento di prestazioni mediante gli attributi, tramite un esempio di come viene applicata.

Nel Capitolo 3 sono esposte le metriche di valutazione fulcro degli esperimenti principali. Per ognuna son spiegate le dinamiche e le motivazioni per cui son state scelte.

Il quarto (Capitolo 4) è dedicato agli esperimenti. Vengono fornite le conoscenze di base per comprendere i tre algoritmi su cui s'è lavorato. Per ognuno di questi vengono descritte le meccaniche, tramite un esempio e poi sono illustrati i risultati raggiunti.

Si termina con le conclusioni (Capitolo 5) e la bibliografia.

2 Implementazione

In questo capitolo vengono spiegati i metodi utilizzati per cercare di migliorare le prestazioni dell'algoritmo di CNRL [3].

2.1 Come funzionano le dinamiche di CNRL

In prima istanza vengono presi in input tutti i dati necessari, tra cui il grafo su cui si andrà a lavorare. Questo vien caricato in formato edgelist, ossia una lista d'archi identificati da nodo di partenza più nodo d'arrivo. Il grafo vien dunque preprocessato per utilizzare l'algoritmo di **Node2Vec** che effettuerà le visite. Node2Vec è un algoritmo che regola le visite sulla base di due parametri p e q . Questi due valori fanno variare l'esplorazione del grafo da una **visita in ampiezza (BFS)** ad una **visita in profondità (DFS)**, permettendo anche delle varianti intermedie.

I cammini individuati dalle visite effettuate con questo algoritmo sottostanno a ben definite caratteristiche. Da ogni nodo (in tutto n) partono w cammini, per un totale di $w \cdot n$ cammini generati tutti di lunghezza al più l . Tali cammini sono lunghi al più l e non esattamente l in quanto quest'algoritmo gestisce sia grafi orientati che non. Di conseguenza se si visita un grafo orientato è possibile imbattersi in un pozzo. Si può immaginarlo come un vicolo cieco. In tali situazioni la visita si ferma, in quanto non è più possibile procedere, pertanto si avrà una lunghezza inferiore a l . Diversamente se si parla di grafi non orientati, tale problema non si pone poiché, anche se si arrivasse ad un pozzo, sarebbe sempre possibile tornare indietro, infatti Node2Vec permette di ritornare su nodi già visitati.

Ecco alcuni casi limite, perfettamente gestiti dall'algoritmo, che possono accadere durante la creazione dei cammini:

- **nodo isolato su grafo orientato:** si avrà un cammino di lunghezza 1, che comprende unicamente quel nodo
- **nodo isolato su grafo non orientato:** si avrà un cammino lungo l contenente l'ID di quel nodo in continuazione
- **loop su grafo orientato:** questa è una sequenza di nodi che si ripete costante attraverso il cammino, è possibile iniziare con un nodo già presente nel loop o arrivarci dopo svariati passaggi
- **sequenze fisse finali su grafo orientato:** è possibile che tanti cammini terminino con la stessa sequenza di nodi. Questo accade perché sul grafo vi è un percorso senza possibilità di scelta che porta ad un pozzo: una volta imboccato tale percorso il cammino andrà a terminare sempre con la stessa sequenza

Una volta che Node2Vec ha generato i cammini è compito di un'altra sezione ricavarne le comunità, questo può essere fatto mediante due approcci differenti. Il primo, il più noto, è **Word2Vec** che va a considerare gli ID dei nodi come se fossero parole. La conseguenza è un cammino ossia un array (vettore) di nodi che vien considerato come un array di parole interpretabili come una frase. I dettagli del funzionamento si possono trovare nell'articolo "How exactly does word2vec work?"[13].

Diversamente il codice di CNRL utilizza, di default, un secondo metodo chiamato Latent Dirichlet Allocation (**LDA**)[1] o una sua variante detta **HalfLDA**, entrambe le varianti si basano ancora su Word2Vec. Le meccaniche esatte di questi metodi, non erano di nostro interesse, pertanto non sono state e non verranno approfondite.

In questo caso non è rilevante quale algoritmo viene utilizzato, in quanto entrambi restituiscono un insieme di comunità, che formano la partizione cercata. Per dare una misurazione al lavoro svolto è ora possibile calcolare la modularità (spiegata in dettaglio nell'apposito capitolo sui metodi di valutazione).

2.2 Introduzione degli attributi

Per individuare le comunità esistenti CNRL sfrutta unicamente la struttura del grafo.

Tuttavia entrambi i dataset su cui si è lavorato durante il tirocinio, denominati Cora e Citeseer (meglio spiegati nel capitolo sugli esperimenti), presentano oltre al file, che delinea la struttura del grafo, anche un secondo file, con estensione ".content", che contiene per ogni nodo una lista di valori binari, il cui significato è:

- 0 il nodo non ha l'attributo identificato da questa colonna
- 1 il nodo ha l'attributo identificato da questa colonna

Tutte le informazioni contenute in questo file non sono considerate dal codice di CNRL per questo motivo si è deciso d'introdurle. La finestra di codice, in cui è possibile apportare modifiche, è compresa fra il caricamento dei dati dall'input e l'inizio del processo d'estrazione delle comunità, ciò significa che la sola sezione in cui è possibile intervenire apportando delle modifiche è la generazione dei cammini lungo il grafo.

I cammini generati allo stato attuale non danno informazioni sugli attributi, pertanto è necessario, oltre a questi, aggiungerne altri che considerino tali informazioni. In particolare due nodi sono legati unicamente se vi è un arco che va da uno all'altro, intuitivamente si può dire che due nodi sono vicini e quindi legati se sono simili.

Questa somiglianza, se riportata sugli attributi, è possibile definirla: due nodi sono simili se condividono uno o più attributi, contrariamente non sono simili se non condividono nessun attributo. Data la definizione di somiglianza di due nodi attraverso gli attributi, si fa notare che vi è simmetria, pertanto verrà rappresentata attraverso un arco non orientato invece che uno orientato.

Si crea quindi un nuovo grafo indiretto che tiene conto di tutte le somiglianze derivanti dagli attributi, e su questo si richiama l'algoritmo di Node2Vec per generare i nuovi cammini.

2.3 Grafo degli attributi

In questa sezione viene formalizzato il concetto di grafo degli attributi, una volta descritte le due alternative vengono esposti i necessari paragoni.

In partenza si rimuovono gli archi originari del grafo, poiché, se si tenessero, si rischierebbe di generare una seconda volta gli stessi cammini. Diversamente i nodi vanno conservati, in quanto sono questi i punti che si vogliono nuovamente collegare, da ora in avanti questo insieme di nodi prende il nome di **N1**.

Se due nodi condividono lo stesso attributo, sono simili e quindi connessi. Esistono tuttavia due metodologie per connettere i due nodi, a e b : in maniera diretta o indiretta.

- **diretta** consiste nel creare un arco non orientato (l'arco (a, b)) fra i due nodi, è il metodo semplice
- **indiretta** consiste nel creare un nuovo nodo c , che verrà inserito nell'insieme **N2**, e creare due archi che collegano ogni nodo originario a c , quindi si creano i due archi (a, c) e (b, c) .
Questo metodo andrà a formare un nodo per ogni attributo esistente sul grafo, e andrà ad inserirli in **N2**

Nelle Figure 2.1 e 2.2 sono mostrati i due possibili metodi di collegamento dei nodi attraverso gli attributi. In ambedue le figure possiamo osservare tre scene:

1. ci sono unicamente i nodi appartenenti all'insieme **N1** a cui vanno aggiunti gli archi
2. è mostrato il collegamento diretto. Da ogni punto raggiungo qualsiasi altro elemento attraversando esattamente un arco, ossia devo fare un singolo passo.
Questa nuova struttura prende il nome di **ipergrafo**
3. il collegamento è indiretto in quanto si passa attraverso il nodo 2 in Figura 2.1 e attraverso il nodo 4 in Figura 2.2. Il nodo di collegamento è quello generato dagli attributi e pertanto apparterrà ad **N2**.

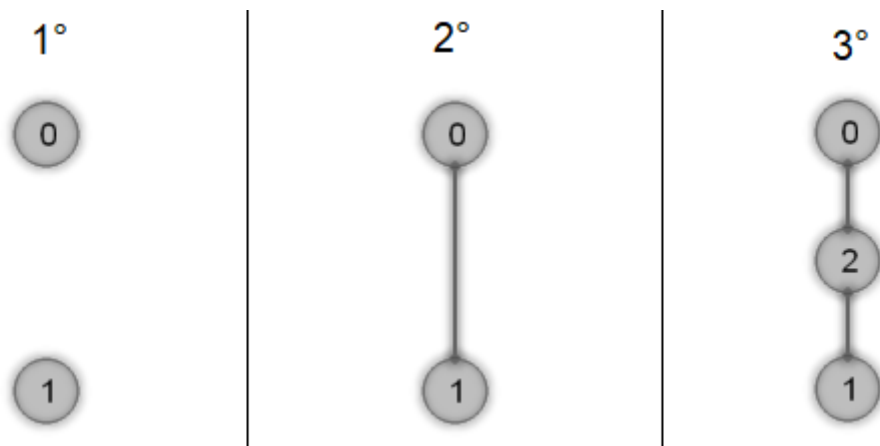


Figura 2.1: Due nodi condividono un attributo, ecco come possono essere legati, 1-nessun arco, 2-ipergrafo (completo), 3-grafo bipartito (a stella)

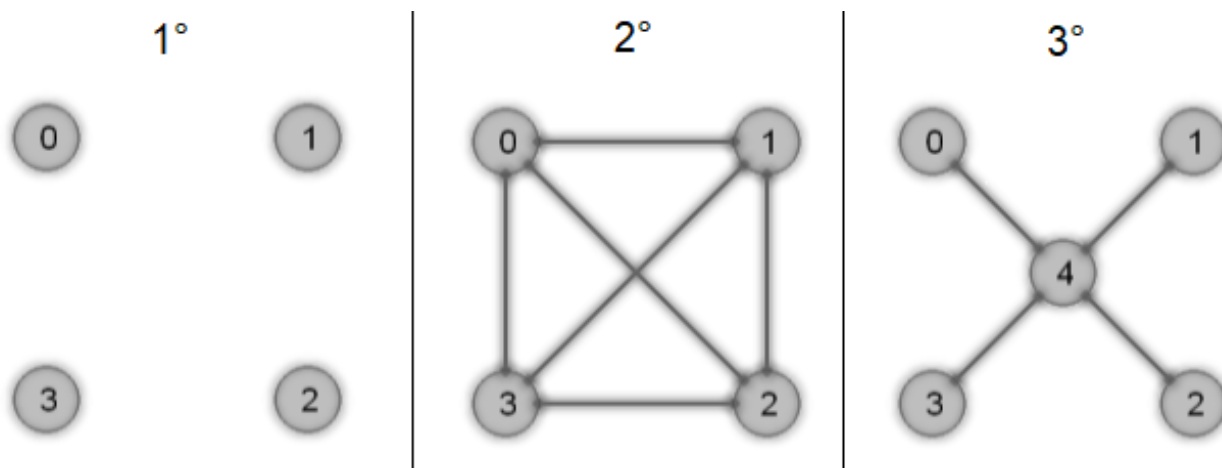


Figura 2.2: Quattro nodi condividono un attributo, ecco come possono essere legati, 1-nessun arco, 2-ipergrafo (completo), 3-grafo bipartito (a stella)

Da ogni nodo di N1 è necessario utilizzare esattamente due passi per raggiungere un altro nodo di N1

Questa nuova struttura prende il nome di **grafo bipartito**

Si può notare come i due modelli siano molto differenti. Nella seconda parte viene creato un **grafo completo**, ciò sta ad indicare che il numero di archi presenti è altissimo poiché segue la legge binomiale $\binom{n}{2}$ dove n rappresenta il numero di nodi coinvolti.

Per il caso in Figura 2.2 vengono aggiunti $\binom{4}{2} = 6$ archi però il numero di nodi rimane invariato.

Diversamente nella terza parte viene costruito un **grafo a stella**. Vi è un nodo centrale che fa da fulcro per collegare tutti gli altri nodi. Questo comporta che sempre in Figura 2.2 vengano aggiunti $n = 4$ archi e un nodo.

Dipendentemente dalla situazione si andrà a preferire un metodo piuttosto che l'altro: ecco le motivazioni che ci hanno guidati nella scelta.

Se vi sono tanti attributi e ognuno di questi appartiene ad un estremamente ristretto numero di nodi, allora ci si può permettere di creare degli ipergrafi. Diversamente, se ci sono anche pochi attributi che sono condivisi fra un alto numero di nodi, la scelta ricade sul grafo bipartito.

Un esempio di quest'ultimo caso è facile da trovare. Se esiste un attributo condiviso fra 100 nodi questo significa che con il grafo bipartito andremo ad aggiungere un nodo e 100 archi. Con l'ipergrafo non si aggiungerà nessun nodo ma in compenso bisogna creare $\binom{100}{2} = 4950$ nuovi archi.

Situazioni come quella presentata in questo esempio sono state affrontate diverse volte durante il tirocinio. Per limitare il numero di archi su cui lavorare si è optato per il grafo bipartito.

È importante far presente che quando si effettuano visite sul grafo bipartito si incontrano tutti i nuovi nodi dell'insieme N2, sconosciuti alle altre parti dell'algoritmo di CNRL. Per questo motivo, una volta che Node2Vec ha generato i cammini sul grafo, si deve rimuovere da ognuno di questi, gli ID dei nuovi nodi generati, altrimenti nei futuri passaggi potrebbero verificarsi degli errori. Questo non sarebbe necessario se si fosse scelto l'ipergrafo, poiché non va a creare nessun nuovo nodo.

2.3.1 Grafo bipartito e ipergrafo, origine dei nomi

Si è parlato di ipergrafo e grafo bipartito senza però spiegarne le origini. Le Figure 2.1 e 2.2 mostrano la situazione di alcuni nodi prendendo in considerazione un unico attributo. Tuttavia nei casi reali vi sono molti attributi. Ne consegue che entrambe le nuove strutture sono un insieme di tante piccole sezioni: il grafo bipartito è un insieme di grafi a stella e l'ipergrafo è un insieme di grafi completi.

Nel dettaglio:

Grafo bipartito

Il grafo è detto bipartito in quando vi sono due insiemi di nodi, quelli che in precedenza sono stati chiamati N1 (nodi originali) ed N2 (nodi derivanti dagli attributi). Tutti gli archi del grafo vanno da un nodo di N1 ad un nodo di N2 e viceversa visto che sono indiretti. Non esistono archi che vanno da N1 a N1 o da N2 a N2.

Questo è esattamente quello che ci si aspetta. Un arco da N1 ad N1 è un arco originario, quindi non necessita dell'attributo per passare da un punto ad un altro, poiché questi sono stati rimossi. Mentre un arco da N2 a N2 semplicemente non avrebbe significato, due attributi legati non hanno motivo d'esistere.

Ipergrafo

Gli ipergrafi nella loro definizione formale non presentano semplici archi, ma dispongono di iperarchi. I normali archi possono essere rappresentati mediante una linea, poiché presentano due estremità entrambe legate ad un nodo. Diversamente gli iperarchi non possono essere rappresentati da una linea. Potrebbero arrivare ad avere infinite estremità, o meglio n estremità dove n è il numero di nodi nel grafo. Tutti i nodi che sono estremità di un iperarco possono raggiungere una qualsiasi delle altre estremità con un passo di lunghezza uno. Se si volesse rappresentare questa situazione su un grafo

normale è necessario costruire un grafo completo per ogni iperarco.

È dunque chiaro il collegamento con la situazione presentata, ogni attributo altro non è che un iperarco. Nel nostro caso è stato necessario rappresentare gli iperarchi come archi normali in quanto l'algoritmo di Node2Vec non è pensato per gestire la reale sintassi degli ipergrafi.

2.4 Creazione del grafo - esempio

In questa sezione viene mostrato come un grafo viene trasformato partendo dalla struttura di base fino a diventare un grafo bipartito o ipergrafo.

nodì	ID attributi
1	11
2	11 12
3	10 12
4	11 13
5	11 12
6	12 13
7	13 14

Tabella 2.1: Ad ogni nodo è associata la lista degli attributi che possiede

att	ID nodi
10	3
11	1 2 4 5
12	2 3 5 6
13	4 6 7
14	7

Tabella 2.2: Ad ogni attributo è associata la lista dei nodi a cui appartiene

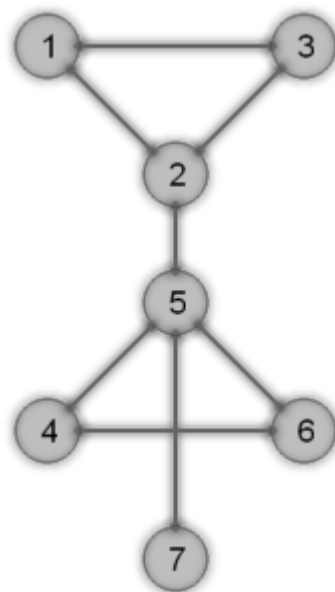


Figura 2.3: Rappresentazione della struttura originaria del grafo

Nella Figura 2.3 si vede la struttura del grafo di partenza.

Nella Tabella 2.1 si ha la lista di attributi che ogni nodo possiede.

Mentre la Tabella 2.2 è inversa alla precedente (Tabella 2.1) in quanto per ogni attributo sono mostrati tutti i nodi a cui appartiene.

Sono riportate entrambe le tabelle per comodità di comprensione, normalmente solo una è sufficiente. Ecco come i due tipi di grafi vengono generati.

Si osservi per primo l'ipergrafo mostrato in Figura 2.4.

- si può vedere come alcuni archi non sono più presenti. Due esempi sono (1, 3) e (5, 7), si ricorda che non sono questi due archi ad essere stati rimossi poiché tutti sono stati rimossi, semplicemente tutti gli altri sono stati nuovamente aggiunti diversamente da questi due

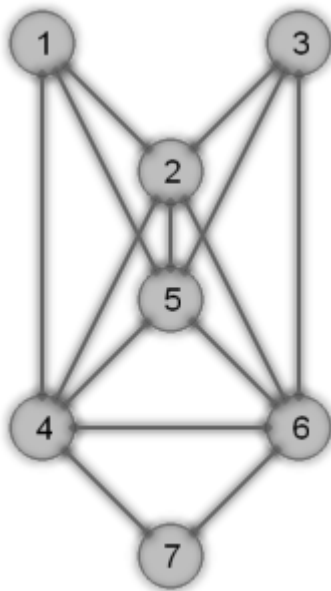


Figura 2.4: Rappresentazione dell'ipergrafo generato

- i nodi che condividono gli stessi attributi vanno a formare un grafo completo, due esempi sono $[1, 2, 4, 5]$ con l'attributo 11 e $[4, 6, 7]$ con l'attributo 13
- se solo un nodo dispone di un particolare attributo, nessun arco sarà creato. Questo è il caso dei due attributi 10 e 14.

Questo può essere spiegato intuitivamente: si farebbe partire un arco da un nodo, ma questo non avrebbe mai una seconda estremità, e quindi non ha motivo d'esistere. O matematicamente: è un caso limite infatti $\binom{1}{2} = \text{error}$ la linea numero 1 del triangolo di tartaglia è "1 1", non esiste il terzo elemento

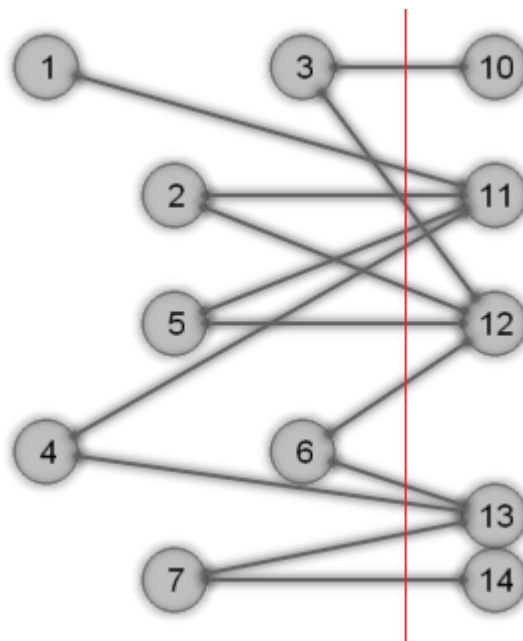


Figura 2.5: Rappresentazione del grafo bipartito generato

Si osservi ora la più complessa situazione del grafo bipartito mostrato in Figura 2.5. Non è facile capire cosa succede.

I nodi dall'1 al 7, sulla sinistra, sono i nodi originali, mentre i nodi dal 10 al 14 (maggiori o uguali a 10), sulla destra, sono i nuovi nodi generati dagli attributi.

- i nodi sulla sinistra e sulla destra sono separati da un immaginaria linea verticale, e rappresentano i due insiemi di cui si parlava in precedenza, $N1$ e $N2$
- tutti gli archi attraversano la linea immaginaria, andando da sinistra a destra e viceversa
- tutti gli archi presenti nella struttura originaria di Figura 2.3 non sono più presenti in quanto son stati rimossi e mai più rimpiazzati, perché non possono esistere archi da $N1$ a $N1$
- i nodi che condividono un attributo creano un grafo a stella, due esempi sono $[1, 2, 4, 5]$ con l'attributo 11 e $[4, 6, 7]$ con l'attributo 13
- se solo un nodo possiede un particolare attributo questo non fa la differenza, i due attributi 10 e 14 sono comunque legati alla parte sinistra, come si può capire dalle tabelle

3 Metodi di valutazione

Questo capitolo spiega di come si valuta una partizione creata dagli algoritmi d'individuazione delle comunità. Come citato nel sommario, il metodo scelto nel corso del tirocinio è stato la modularità, il cui simbolo è Q .

La modularità altro non è che un valore solitamente compreso all'interno dell'intervallo $[-1, 1]$. Tuttavia questo non è sempre vero, poiché dipende strettamente da quale metodo d'implementazione si va ad utilizzare. Le prime persone che hanno lavorato sulla modularità sono state Newman e Girvan, che necessitavano di un metodo formale per decidere fra due partizioni di nodi quale fosse la migliore. Newman e Girvan quando hanno scritto le prime applicazioni della modularità sono partiti da una semplice assunzione: un nodo non può appartenere a più di una comunità. Questo elimina tutti i problemi dovuti alla sovrapposizione delle comunità. Per i casi studiati durante il tirocinio quest'assunzione non sempre è valida. Inizialmente è molto comoda poiché permette di ridurre notevolmente la complessità del calcolo della modularità.

Durante tutto il lavoro svolto si è passati attraverso tre differenti metodi, tutti e tre verranno ora mostrati ed ove possibile spiegati. Si parte dal più semplice fino ad arrivare al più complesso.

Durante la spiegazione di ogni metodo viene assunto di lavorare con grafi non orientati, alla fine di ogni sezione sono indicati gli eventuali cambiamenti da applicare in caso si abbia invece a che fare con grafi orientati.

3.1 Modularità con massimi (`mod_withMax`)

È stato scelto questo nome in quanto questo metodo gestisce al massimo una comunità per ogni nodo, andando a ricalcare l'assunzione base di **Newman e Girvan**.

L'idea di base è che questo metodo va a calcolare un valore di modularità per ogni comunità. Tutti questi elementi sono poi sommati fra loro per andare a creare il valore di modularità del grafo in se. È necessario far notare che questo metodo non permette la sovrapposizione delle comunità, per l'assunzione fatta, di conseguenza è impossibile che la sommatoria finale vada a considerare più di una volta i valori raccolti, da un qualsiasi elemento del grafo.

Questo metodo si basa sull'osservazione che è possibile identificare una comunità sulla base della frequenza degli archi contenuti al suo interno. Di fatto se un insieme di nodi dispone di una grande quantità di archi che li collegano direttamente, questi nodi dovrebbero essere una comunità a sé stante o almeno far parte della stessa. Se così non fosse, allora la valutazione della partizione sul grafo deve essere penalizzata in quanto non ha considerato questi nodi come parte dello stesso gruppo.

Il motivo per cui è stato scelto questo metodo, oltre al fatto che è semplice da calcolare, è proprio perché va a penalizzare le partizioni che non considerano una comunità molto evidente.

Di seguito la formula completa:

$$Q = \sum_c^{C_r} \left(\frac{l_c}{L} - \left(\frac{d_c}{2L} \right)^2 \right) \quad (3.1)$$

Dove gli elementi dell'equazione sono:

- Q è il simbolo di modularità
- C_r è l'insieme delle comunità calcolate
- c è l'iteratore sulle comunità
- L è il numero totale di archi all'interno del grafo, solitamente indicato con $m = |E|$
- l_c è il numero totale di archi interni alla comunità c , questo significa che ognuno degli archi qui contati ha ambo i nodi alle estremità appartenenti alla comunità c
- d_c è il grado della comunità c , definito come la sommatoria dei gradi dei nodi che vi appartengono

Si può notare che $\frac{l_c}{L}$ è il peso della comunità c rispetto al resto del grafo, in quanto si calcola la percentuale di archi di cui la comunità c dispone. Mentre $\frac{d_c}{2L}$ è la densità di archi tramite cui la comunità c interagisce, ossia quanto, rispetto al totale, sono importanti i nodi che appartengono a questa comunità.

Quando si parla di grafo orientato l'unica parte da modificare è $\frac{d_c}{2L}$ che diventa $\frac{d_c}{L}$. In un grafo non orientato un arco incrementa di uno il grado di due nodi, quello di partenza e quello d'arrivo, mentre, se l'arco fosse orientato questo non accadrebbe in quanto aumenterebbe solamente il grado del nodo di partenza, ecco il motivo del fattore moltiplicativo 2 di differenza.

3.2 Modularità con sovrapposizione (mod_overlap)

L'algoritmo qui descritto è preso dall'articolo intitolato "Modularity measure of networks with overlapping communities"[7].

Come si può intuire dal nome, questo metodo permette a due comunità di condividere uno o più nodi e quindi d'aver una sovrapposizione. In linea con diversi altri metodi la modularità generata da questo algoritmo cade nell'intervallo $[-1, 1]$.

Se nel metodo precedente si calcolava un valore di modularità per ogni comunità e poi li si sommava in quanto non era prevista la sovrapposizione, qui la gestione è simile. Si calcola un valore per ogni comunità e poi, invece di sommarli, se ne ricava una media poiché la sovrapposizione è concessa. Uno stesso nodo può portare il suo contributo a diversi elementi della sommatoria.

Di seguito la formula completa:

$$Q = M^{ov} = \frac{1}{K} \sum_{r=1}^K \left[\frac{\sum_{i \in c_r} \left(\frac{\sum_{j \in c_r, i \neq j} (a_{ij}) - \sum_{j \notin c_r} (a_{ij})}{d_i \cdot s_i} \right)}{n_{c_r}} \cdot \frac{n_{c_r}^e}{\binom{n_{c_r}}{2}} \right] \quad (3.2)$$

Dove gli elementi dell'equazione sono:

- C_r è l'insieme delle comunità calcolate
- K è il numero di comunità calcolate, pari a $|C_r|$
- c_r è la comunità attuale
- n_{c_r} rappresenta il numero di nodi interni alla comunità c_r
- $n_{c_r}^e$ rappresenta il numero di archi interni alla comunità c_r
- $\binom{n_{c_r}}{2}$ è il numero massimo di archi potenzialmente presenti nella comunità c_r

- d_i è il grado del nodo i
- s_i è il numero di comunità a cui il nodo i appartiene
- a_{ij} vale 1 se l'arco dal nodo i al nodo j esiste, altrimenti assume il valore 0

Ecco spiegati i dettagli presenti all'interno di quest'Equazione 3.2.

Come per il metodo precedente, si calcola la densità della comunità (attraverso la Formula 3.3). Ora si confronta il numero di archi interni presenti, rispetto al potenziale numero massimo, ossia il numero di archi presenti in un grafo completo con lo stesso numero di nodi.

$$\frac{n_{c_r}^e}{\binom{n_{c_r}}{2}} \quad (3.3)$$

In aggiunta, si considera la relazione fra il numero di archi interni e quelli uscenti (tramite la Formula 3.4). Di conseguenza sembrerebbe che se il numero di archi uscenti è maggiore rispetto al numero di archi interni, il valore di modularità vada ad assumere un valore negativo, in caso contrario positivo.

$$\sum_{i \in c_r} \left(\frac{\sum_{j \in c_r, i \neq j} (a_{ij}) - \sum_{j \notin c_r} (a_{ij})}{d_i \cdot s_i} \right) \quad (3.4)$$

In realtà non è sempre così, si deve tener conto anche del peso degli elementi d_i e s_i . Questi due parametri hanno il compito di assegnare ad ogni arco un valore rappresentativo della sua importanza, interno o uscente è completamente indifferente. Si può notare come il valore di un arco possa variare all'interno dell'intervallo $]0, 1]$. Un arco ha inizialmente valore 1 e viene poi diviso per due interi positivi moltiplicati fra loro, in seguito viene considerato con peso positivo se favorisce la comunità ossia se è interno, negativo se così non è.

d_i è importante perché va a premiare gli archi che partono da un nodo con un basso grado. Il nodo avendo pochi archi, fa sì che ognuno di questi sia molto importante, in quanto, la somma totale dei pesi degli archi di un nodo, è un valore costante uguale per tutti i nodi. Inoltre, tramite s_i , si premiano i nodi che appartengono a poche comunità, poiché tali nodi meglio rappresentano la loro comunità d'appartenenza. Se un nodo è comune a tutte le comunità del grafo non è per nulla rappresentativo e quindi ogni volta avrà un'influenza estremamente bassa.

È stato scelto questo metodo poiché permette la sovrapposizione di comunità. Inoltre andando a calcolare un valore di modularità per ogni comunità individuata nel grafo si può facilmente capire quale di queste dà un maggior contributo al risultato finale e quale invece tende solo a penalizzare la partizione a causa della sua bassa coesione.

Compresa la formula generatrice, si può notare che un particolare valore di modularità può rivelare due importanti dettagli del grafo:

- un valore negativo sta a significare che i gruppi di nodi non sono densamente connessi al loro interno ma piuttosto legati a punti esterni. Pur con un valore negativo è possibile che gli archi interni siano in numero maggiore rispetto agli archi verso l'esterno, ma, se ciò accade, è perché i parametri di d_i e s_i hanno dato maggior rilevanza agli archi uscenti
- se la modularità ha un valore, che in assoluto si avvicina molto allo zero, sta a significare che il grafo valutato è estremamente sparso. Il fattore moltiplicativo mostrato nell'Equazione 3.3 va a collassare i vari elementi della formula vicino allo zero, di conseguenza anche la sommatoria non si allontanerà di molto da quel valore

Quando si parla di grafo orientato l'unica differenza, da considerare, la si applica alla Formula 3.3. Si dà il caso che il numero massimo di archi in un grafo orientato sia il doppio del numero di archi per lo stesso grafo non orientato, questo per il semplice motivo che si possono creare due archi leganti gli stessi due nodi se presentano versi opposti. Pertanto il numero massimo cambia da $\binom{n_{c_r}}{2}$ per diventare $2\binom{n_{c_r}}{2}$.

3.3 Modularità modificata

Questo algoritmo[25] è preso dall'articolo intitolato "Incorporating Implicit Link Preference Into Overlapping Community Detection"[24].

Questo è l'algoritmo che è stato scelto da chi ha scritto l'articolo di CNRL e, per questa ragione, è stato adottato anche da noi. È estremamente importante perché permette di replicare i risultati illustrati nel documento cui facciamo riferimento. Si ha quindi un punto fisso da cui partire e con cui all'occorrenza confrontarsi.

Sfortunatamente questo è il più lento dei tre metodi che vengono illustrati in questo capitolo, per il semplice motivo che è il più complesso, sia logicamente che computazionalmente.

Di seguito vi sono due versioni della stessa formula (senza sistema 3.5 e con sistema 3.6):

$$Q = \frac{1}{M} \cdot \sum_{u,v \in V} \left[\left(A[u, v] - \frac{d_{in}(u) \cdot d_{out}(v)}{M} \right) \cdot |C_u \cap C_v| \right] \quad (3.5)$$

$$Q = \frac{1}{M} \cdot \sum_{u,v \in V} \begin{cases} 0 & |C_u \cap C_v| = 0 \\ \left[\left(A[u, v] - \frac{d_{in}(u) \cdot d_{out}(v)}{M} \right) \cdot |C_u \cap C_v| \right] & altrimenti \end{cases} \quad (3.6)$$

Dove gli elementi delle equazioni sono:

- M può assumere due valori a seconda che il grafo sia orientato o no, ossia misura m se indiretto assume invece il valore $2m$ se diretto.
In tutto questo m è il numero di archi del grafo, ossia come di norma $m = |E|$
- u, v sono gli iteratori sui nodi del grafo, chiamati come di consueto V
- $A[u, v]$ rappresenta il peso assunto dall'arco (u, v) . Se tale arco non esiste il valore è 0, mentre se l'arco esiste il peso è un valore positivo, di default 1
- $d_{in}(u)$ e $d_{out}(u)$ sono i gradi entranti e uscenti del nodo u , questi rispettivamente contano il numero di archi entranti e uscenti dal nodo.
Se il grafo è indiretto, questi due valori, preso un qualsiasi nodo, saranno perfettamente uguali
- C_u è l'insieme delle comunità a cui il nodo u appartiene
- $|C_u \cap C_v|$ è il numero totale di comunità condivise fra i nodi u e v

Come per i metodi precedenti segue una spiegazione delle varie sezioni della Formula 3.5.

Si prende l'elemento $|C_u \cap C_v|$: questo fattore moltiplicativo ci permette d'ignorare gli archi che legano nodi che non condividono nessuna comunità. Per capire il suo scopo si può pensare ad una partizione che non presenta sovrapposizioni, in questo caso tale componente permetterà di ignorare tutti gli archi che non sono interni ad una singola comunità, ma passano da una all'altra. Ignorando ora l'assunzione base di Newman e Girvan si può dire che gli unici archi considerati sono quelli che rimangono interni ad almeno una comunità, anche se è permesso loro essere di collegamento fra altre. Il funzionamento di questo componente è illustrato bene nella seconda versione (Formula 3.6), grazie al sistema si nota che se la condizione è verificata si può ignorare tutta la prima parte.

Oltre a filtrare gli archi utilizzabili, questo componente assegna un'importanza maggiore a quegli archi che connettono nodi che condividono molte comunità.

$$\frac{d_{in}(u) \cdot d_{out}(v)}{M} \quad (3.7)$$

La base che si ignora o moltiplica è un valore positivo, se l'arco esiste (questo grazie ad $A[u, v]$), mentre assume un valore negativo se l'arco non c'è, in quanto al peso 0 dell'arco inesistente si sottrae il fattore definito nella Formula 3.7.

Quest'espressione rappresenta la rilevanza di un arco, indipendentemente dalla sua esistenza. Se tale arco collega due nodi con gradi elevati risulta essere uno fra molti e quindi ha un'importanza limitata,

se diversamente collega nodi con gradi bassi allora assume molta più importanza perché è un caso isolato e quindi prezioso.

4 Esperimenti

In questo capitolo vi è una raccolta rappresentativa degli esperimenti svolti, con le relative analisi e commenti.

4.1 Origini dei grafi

In questa sezione vengono spiegati i dataset utilizzati e cosa rappresentano.

Si possono dividere in due gruppi. Perché presentano caratteristiche d'affinità, inoltre i grafi dei due gruppi sono stati utilizzati nel corso del tirocinio per applicazioni differenti.

Nella Tabella 4.1 sono elencati i dettagli tecnici di ogni dataset.

nome	nodi	archi	archi/nodi	diretto	etichette	attributi
Cora	2708	5429	2.0	sì	7	1433
Citeseer	3312	4732	1.4	sì	6	3703
BlogCatalog	10312	333983	32.4	no	39	0
Gnutella	6301	20777	3.3	no	0	0
Dolphins	62	159	2.6	no	0	0
Karate	35	78	2.2	no	0	0

Tabella 4.1: Per ogni grafo sono indicate le sue caratteristiche principali

Esperimento principale: Cora - Citeseer

[12][18] Il dataset di **Cora** è un grafo diretto che rappresenta una rete di articoli scientifici sull'apprendimento automatico. Ogni nodo è un articolo, mentre ogni arco rappresenta il collegamento fra un documento e l'altro, se il documento A cita il documento B si avrà l'arco (A, B) . La rete è costruita in modo che ogni nodo abbia almeno un arco entrante o uscente.

Ogni articolo appartiene esattamente ad una classe identificata da un'etichetta. Ogni attributo rappresenta la presenza o meno di una determinata parola nel testo del documento (deve apparire almeno 10 volte).

Il dataset di **Citeseer** è un grafo diretto che rappresenta una rete analoga a quella di Cora, anche qui si hanno articoli scientifici che si citano vicendevolmente. L'etichetta è la classe d'appartenenza di cui l'articolo fa parte, e gli attributi sono nuovamente la presenza o meno di certe parole all'interno del testo.

Grafi per esperimenti minori

Questi quattro grafi vengono presentati in ordine decrescente nel numero dei nodi, così come fatto nella seconda parte della Tabella 4.1.

- **BlogCatalog**[8] rappresenta la rete di relazioni, di conoscenza, fra gli utenti di alcune piattaforme per blog
- **Gnutella**[9][16] questo grafo si rifà alla rete peer-to-peer per lo scambio di file di Gnutella nell'agosto 2002. Ogni nodo rappresenta un host e ogni arco un collegamento fra due host
- **Soc-Dolphins**[4][17] riporta la rete sociale di un gruppo di delfini al largo della Nuova Zelanda nel 2003
- **Karate**[23] raccoglie lo storico degli incontri in un club di karate universitario nel 1977. Ogni nodo è un membro del club, ogni arco indica uno scontro terminato in pareggio

4.2 Modello di un grafo

In questa sezione viene spiegato cosa si intende per modello di un grafo, e perché è tanto importante per gli algoritmi che vi sono costruiti sopra.

Come descritto nel capitolo sull'implementazione è necessario visitare il grafo per generare su questo dei cammini. Tali vettori di nodi vengono presi in input dall'algoritmo di Word2Vec che, tramite una rete neurale va a descrivere ogni nodo coinvolto tramite un vettore di valori reali. CNRL utilizza invece un metodo alternativo chiamato Latent Dirichlet Allocation (LDA)[1] o una sua variante detta HalfLDA, entrambe sono comunque basate su Word2Vec .

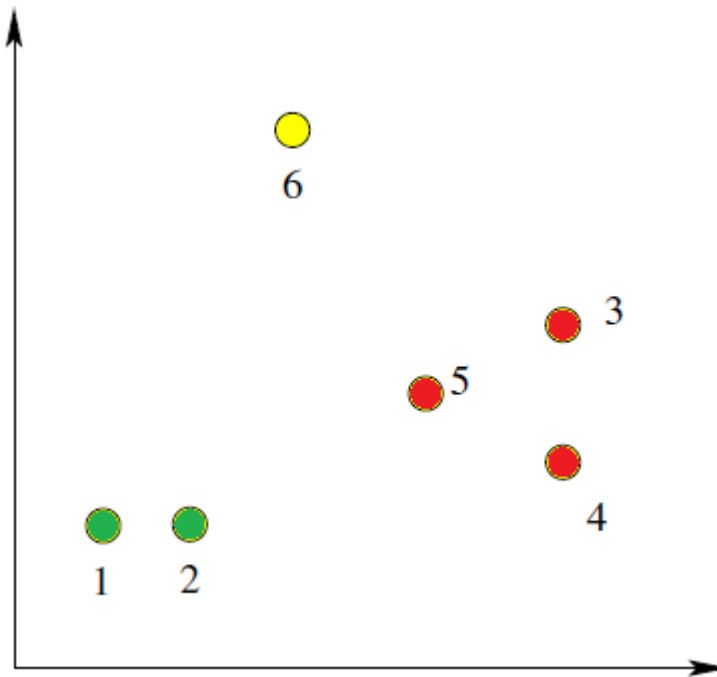


Figura 4.1: Son visualizzati i nodi d'un grafo rappresentati da un modello che dispone di due dimensioni, mostrato in Tabella 4.2

ID	X1	X2
1	1	2
2	2	2
3	7	5
4	7	3
5	5	4
6	3	8

Tabella 4.2: Tabella rappresentativa del modello dei nodi in Figura 4.1

Di norma ogni vettore contiene 64 numeri reali e ognuno rappresenta una posizione lungo un asse in uno spazio a 64 dimensioni. È possibile pensare anche ad un caso più semplice. In Figura 4.1 si possono vedere sei punti che si può immaginare appartenessero ad un grafo e fossero legati fra loro in qualche modo. Una volta generati i cammini su tale grafo, e questi forniti ad uno dei due algoritmi, viene calcolata la Tabella 4.2. Ad ogni ID di un nodo vengono associati due valori (qui interi per semplicità). Se ne usano due in quanto facili da mostrare, con 64 sarebbe pressoché impossibile. Si possono dunque immaginare come coordinate su un piano cartesiano, e così si arriva alla Figura 4.1, visivamente si può dire che esistono 3 gruppi distinti evidenziati dai colori, abbiamo quindi $(1, 2) - (3, 4, 5) - (6)$.

Escludendo l'influenza causata dai colori, il nostro occhio ci suggerisce l'esistenza di questi 3 gruppi in quanto i nodi che vi fanno parte sono vicini fra loro. Questo ricalca alla perfezione la definizione

di somiglianza: tanto più due nodi sono vicini tanto più questi sono simili. Si ricorda che esistono comunque molte definizioni formali di somiglianza e la distanza euclidea è solo una di queste.

Se due nodi molto simili, e quindi vicini, vengono collegati da un arco, allora tale arco assume un valore molto elevato, vicino a 1, diversamente assumerà un valore molto basso, vicino a 0, se i nodi che lega sono diversi ossia molto lontani.

Questo valore verrà chiamato d'ora in avanti coerenza di un arco¹.

4.3 Link Prediction

Questo è il primo di tre algoritmi, che verranno illustrati in questo capitolo, tutti e tre basati sul modello del grafo appena spiegato.

L'algoritmo di Link Prediction ha lo scopo d'assegnare un valore al grafo in input. Questo valore è una percentuale e per tale motivo ricade nell'intervallo $[0, 1]$. Questa percentuale sta a rappresentare la coerenza degli archi del grafo. Nel dettaglio, un arco è molto coerente se lega due nodi simili, un grafo è tanto più coerente quanti più archi coerenti ha.

4.3.1 Dinamiche di funzionamento

In prima istanza viene caricato il grafo, sempre come lista d'archi, sono questi il fulcro dell'algoritmo. Si va poi a spezzare gli archi in due gruppi, dopo averli mescolati per non creare delle divisioni sempre uguali. Tale separazione viene fatta sulla base del parametro F , che indica quale sarà la percentuale degli archi adibiti all'addestramento del modello e quali invece saranno adibiti alla fase di test.

Il numero degli archi usati per il test viene dato dalla Formula 4.1

$$m \cdot \frac{1}{F} \quad (4.1)$$

$$m \cdot \left(1 - \frac{1}{F}\right) \quad (4.2)$$

mentre quelli utilizzati per l'addestramento corrispondono alla Formula 4.2, dove m è il numero di archi del grafo.

Il nuovo grafo (**G1**) creatosi, inseguito alla rimozione degli archi per il test, viene utilizzato per generare il modello.

Per ogni arco appartenente all'insieme di test (**Test=T**) se ne genera un altro (**Check=C**) tramite una coppia di nodi casuali, assicurandosi che non corrispondano ad un arco già esistente. Si hanno quindi tre insiemi, il primo adibito all'addestramento, stabilisce il modello necessario a discernere quale delle due restanti selezioni è la migliore. Per ogni arco, di T e di C, si va a veder quanto sono simili i nodi che legano e se ne calcola così la coerenza, tramite l'apposita funzione di similarità, ora disponibile. Gli archi possono essere valutati sulla base di questo nuovo parametro.

Si è interessati a capire quanto sono buoni gli archi originari del grafo, T, rispetto a quelli generati casualmente, C.

$$\frac{(n_1 + \frac{n_2}{2})}{n} \quad (4.3)$$

Si usa perciò la metrica detta **AUC**[6], che si basa sulla Formula 4.3, i cui elementi sono:

- n è il numero di archi presenti nell'insieme di test, calcolato mediante la Formula 4.1
- n_1 sono le volte in cui un arco di test T è migliore di un arco casuale di C
- n_2 sono le volte in cui un arco di test T ha valore pari o quasi rispetto ad un arco casuale di C

Per ogni arco di T se ne associa casualmente un altro di C. Si guarda quanto valgono i due archi coinvolti. Se l'arco di T vale più dell'arco di C si incrementa di 1 il valore di AUC, se uguali o quasi (differenza inferiore ad una soglia detta ϵ) si incrementa di 0.5. Avendo fatto ciò con ogni arco se ne

¹Non esiste un vero e proprio termine tecnico per questa definizione

calcola la media.

Il valore finale della metrica AUC dipende fortemente da quale metodo si adotta per abbinare gli archi appartenenti a T e C. Un accoppiamento sbagliato può fortemente sbilanciare il risultato, è per questo motivo che si va ad utilizzare un abbinamento casuale.

Il parametro F influisce sul risultato finale. Più questo si avvicina a 1 più cresce il numero di archi da utilizzare per la selezione di test, di conseguenza diminuiscono i dati su cui fare affidamento per creare un buon modello che rappresenti appieno il grafo di partenza. Il valore finale da aspettarsi è quindi tendenzialmente più basso.

È necessario ricordare che la stima qui calcolata dipende fortemente dalla selezione iniziale. È per questo motivo che tutto il processo non viene svolto solo una volta bensì F volte, così da andare a stabilizzare il valore calcolato tramite la media di tutte le iterazioni. Inoltre questo permette di utilizzare via via tutti gli archi per addestrare il modello di **G1**.

L'intero algoritmo si basa sul confronto degli archi grazie al valore che li rappresenta, per individuare i più rilevanti. La parte critica è la generazione di tale valore mediante la funzione di similarità. Tale funzione può essere definita in svariate maniere in quanto si possono pesare diversi aspetti. Un esempio classico è basato sulla distanza euclidea: più due nodi sono distanti nella rappresentazione del modello meno sono simili. Verranno mostrate le applicazioni di altre funzioni di similarità basate su diversi principi che però non verranno qui riportati.

4.3.2 Esempio

arco	autentico	dist	AUC
3-4	sì	2	1
2-6	no	$\sqrt{37}$	
4-5	sì	$\sqrt{5}$	0.5
3-5	no	$\sqrt{5}$	
5-6	sì	$\sqrt{20}$	0
2-5	no	$\sqrt{13}$	
1-2	sì	1	1
1-3	no	$\sqrt{45}$	

Tabella 4.3: Tabella riassuntiva del procedimento di Link Prediction

Si consideri il modello descritto tramite la Figura 4.1 e la Tabella 4.2. Si sa inoltre che $F = 2$, gli archi originali adibiti al test sono $(1, 2), (3, 4), (4, 5), (5, 6)$ mentre quelli generati casualmente sono $(3, 5), (2, 5), (2, 6), (1, 3)$. I due insiemi sono nello stesso numero per creazione e per essere facilmente confrontati.

Nelle prime due colonne della Tabella 4.3 sono riassunti gli archi e la loro origine. Le colonne seguenti rappresentano invece i passaggi successivi dell'algoritmo.

Nella terza colonna si può notare che ad ogni arco è stato associato un valore rappresentativo della distanza euclidea misurata sul piano cartesiano fra i due estremi dell'arco. La funzione di similarità premia i valori più bassi a discapito degli altri. Inoltre la Tabella è stata ordinata in modo che tutti gli archi di un insieme siano associati con uno dell'altro in maniera casuale. Si può notare che nella seconda colonna "sì" e "no" s'alternano.

Nell'ultima colonna viene rappresentato l'incremento che si andrà ad apportare alla metrica AUC grazie alla valutazione dei due archi corrispondenti. Questo valore dovrà poi esser diviso per il numero delle coppie in gioco per riportarlo nell'intervallo $[0, 1]$, in questo caso risulta $\frac{(1+0.5+0+1)}{4} = 0.625$.

Si possono osservare alcune peculiarità:

- l'accoppiamento degli archi viene fatto in maniera casuale per evitare uno sbilanciamento
- nel caso mostrato una selezione a favore degli archi autentici porterebbe ad avere un incremento nullo del valore di AUC, al contrario si potrebbe avere un incremento di due unità se si favorissero gli archi generati casualmente

- non si hanno archi ripetuti perché viene impedito durante la creazione, ma questo non vuol dire che un arco di C non possa corrispondere ad un arco usato in principio per generare il modello
- si può immaginare che più il valore di F è grande e quindi più è grande la selezione di archi per l'addestramento (grazie alla Formula 4.2), più il valore di AUC calcolato cresca

L'algoritmo prevederebbe una seconda iterazione di tutto questo procedimento utilizzando gli archi in principio scartati come nuova base per il modello per poi calcolare la media delle percentuali. Si omette questa fase perché uguale a quella appena illustrata.

4.3.3 Applicazioni

Le similarità utilizzate in questi esempi sono[11]:

- **DW** = DeepWalk
- **CN** = Common Neighbors[14]
- **Salton** = Salton Index[2]
- **Jaccard** = Jaccard Index
- **RA** = Resource Allocation[26]

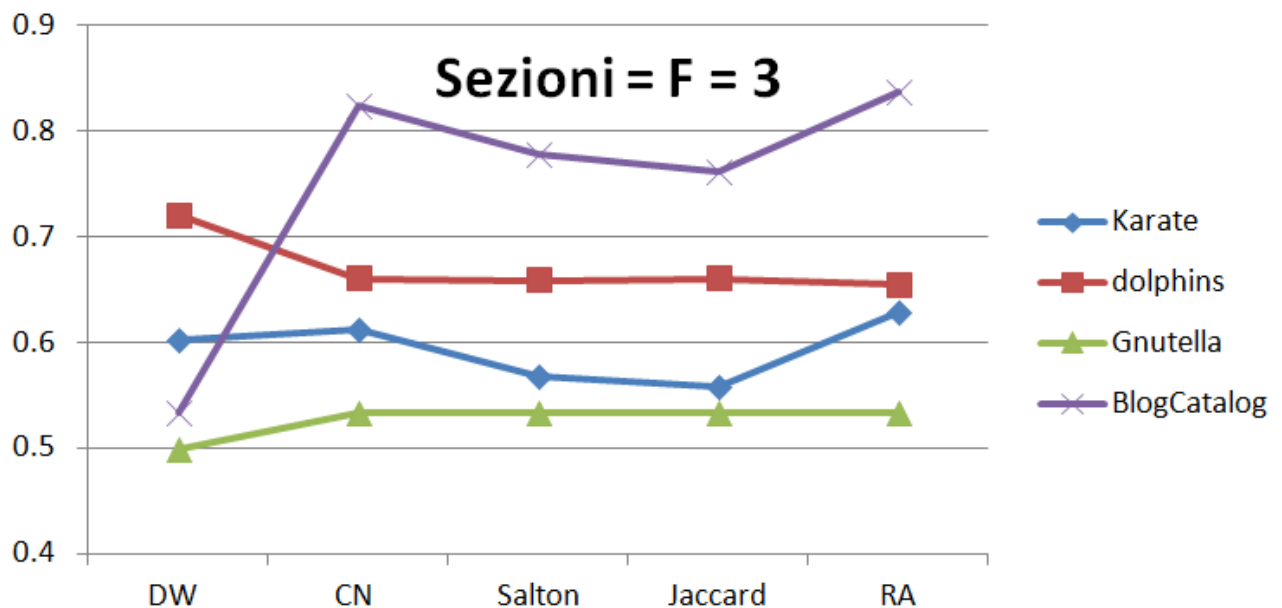


Figura 4.2: Link Prediction con tre sezioni, $F = 3$

Questo algoritmo ha tre possibili variabili su cui si può lavorare:

- Il **parametro F** (dall'inglese fold) viene fissato nella Figura 4.2. Si può notare come ogni grafo abbia un livello su cui si attesta, BlogCatalog grazie al fatto che è estremamente denso ha un valore in media molto alto rispetto ai tre grafi restanti
- Nella Figura 4.3 si lavora su un **unica metrica**. Si utilizza solo la funzione di similarità denominata Common Neighbors, che si basa sui nodi geometricamente vicini agli estremi di un arco all'interno del modello del grafo per valutare la somiglianza. Si può notare come all'aumentare del parametro F , il valore che si associa ad ogni grafo tende ad aumentare. Questo è dovuto ad un modello meglio addestrato perché dispone di più dati in partenza, e quindi si ha una migliore valutazione

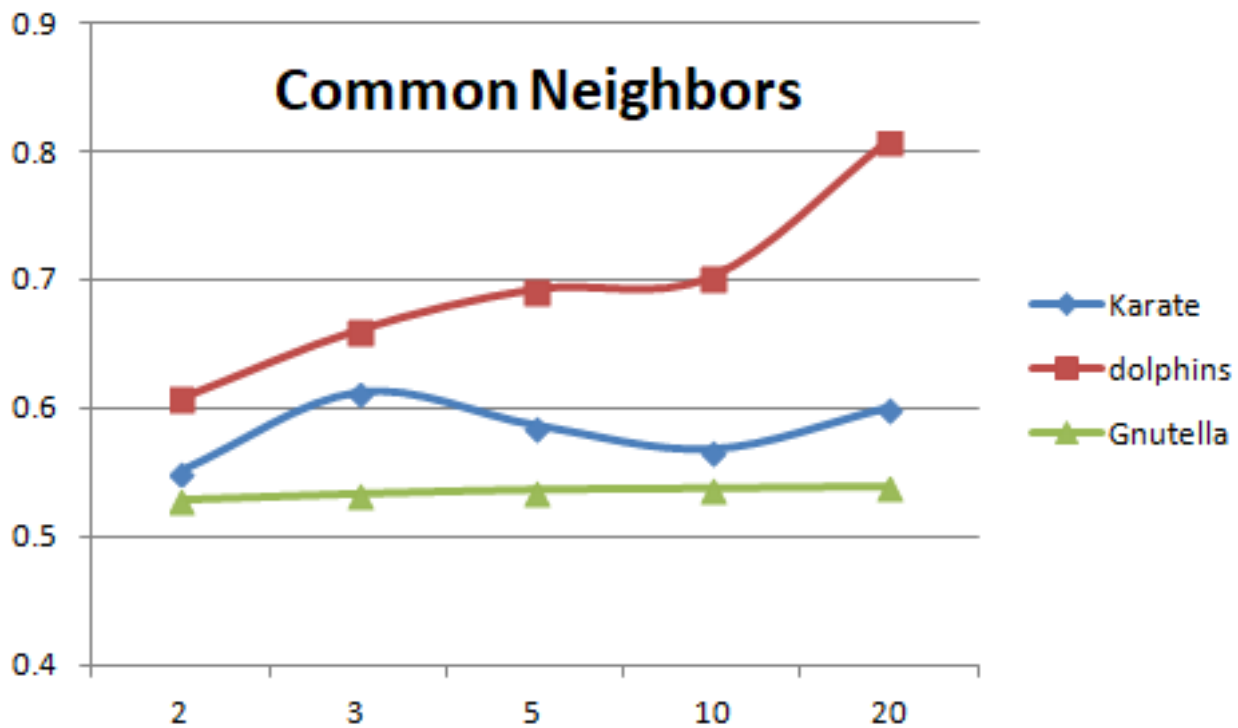


Figura 4.3: Andamento della metrica di Common Neighbors, su tre grafi distinti attraverso differenti percentuali di selezione

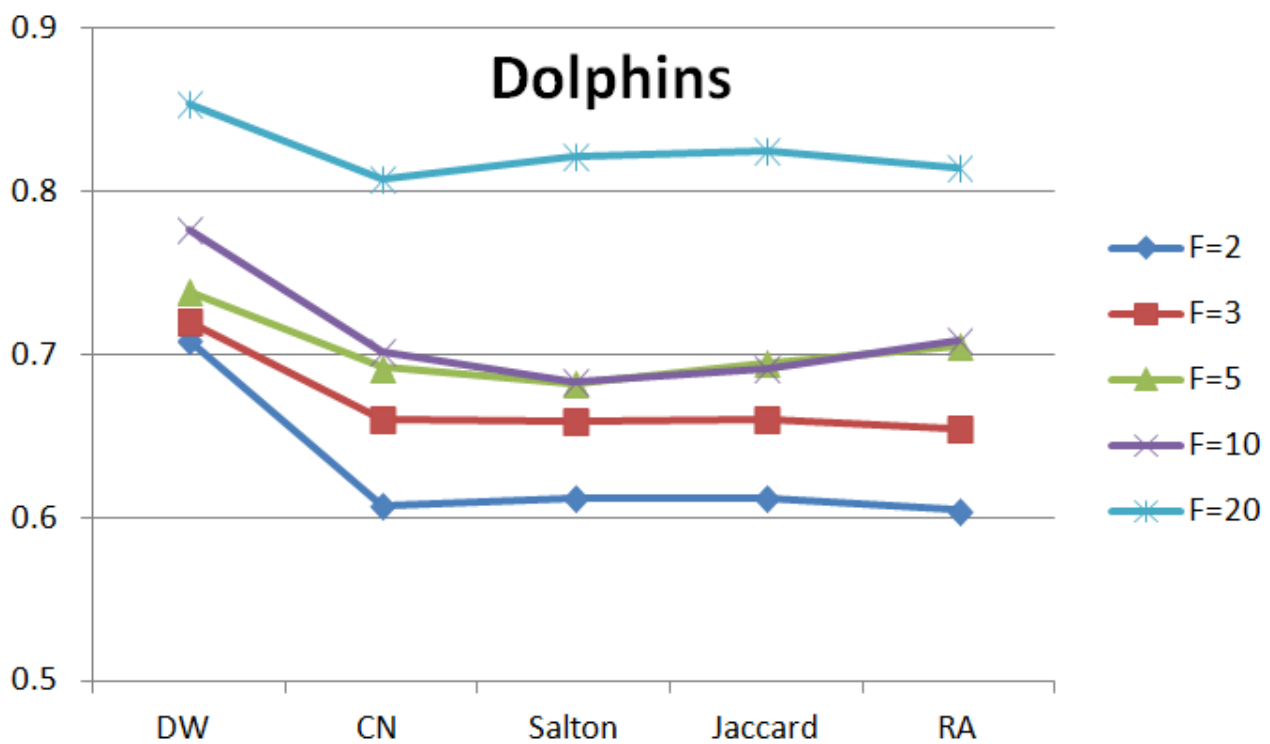


Figura 4.4: Andamento del grafo dolphins, su diverse metriche, attraverso differenti percentuali di selezione

- La Figura 4.4 è forse la più semplice da osservare in quanto è mostrato **un solo grafo**. Come prima si vede che l'influenza del parametro F sul risultato finale è altissima. Si può notare che le prestazioni di DeepWalk sono leggermente più alte rispetto al resto che invece tende ad avere le stesse performance

Si precisa che questi sono degli esempi rappresentativi dell'andamento generale, che però non viene riportato in quanto sarebbe molto complesso da comprendere.

4.4 Classificazione dei nodi

Il problema della classificazione dei nodi/vertici punta a riconoscere, partendo dalla rappresentazione di un elemento, a quale classe questo appartenga. Per definizione, un elemento può appartenere ad una ed una sola classe, è tuttavia possibile pensare a gruppi interni ad altri gruppi o a sovrapposizioni. Per esempio, data la classe A e la classe B si ha che, per come sono state definite, tutti gli elementi di A appartengono anche a B ($A \subseteq B$), di conseguenza $A \cap B = A$. Tuttavia questo vale per l'insiemistica, nei problemi di classificazione, fra A e B, non c'è nessuna relazione e conseguentemente si ha che $A \cap B = \emptyset$.

4.4.1 Dinamiche di funzionamento

Il punto di partenza è costante, dato il grafo lo si visita per generare i cammini e con questi s'invoca l'algoritmo che genera il modello. Il modello viene disordinato e spezzato secondo il parametro di **training ratio** T_r , ossia la percentuale d'addestramento, questo indica la percentuale del modello che dovrà essere usata per l'addestramento, la restante sarà per i test.

In aggiunta si considerano anche le etichette o classi d'appartenenza, quindi per ogni nodo si ha una tripletta contenente:

- ID del nodo che normalmente non viene considerato
- il vettore di numeri reali che lo rappresenta, vengono presi dal modello
- l'identificativo della classe a cui si appartiene

Con questi dati viene addestrata la **funzione di classificazione**, o **classificatore**, il cui scopo sarà predire la classe d'appartenenza dato in input il vettore rappresentativo di un nodo.

In seguito per ogni nodo appartenente all'insieme di test si cerca di predire la sua classe sottoponendo il suo vettore al classificatore. Una volta terminato, si confronta se l'esito corrisponde con la reale classe, nota grazie allo stesso file di etichette utilizzato per l'addestramento.

È importante far notare che il modello viene spezzato in due perché il classificatore deve essere addestrato su dati diversi da quelli che prenderà in seguito in input. Se così non fosse non ci sarebbe nulla da predire in quanto i dati passati vengono ricordati e ed è impossibile sbagliare una risposta.

La valutazione delle predizioni effettuate può avvenire mediante un metodo semplice ossia il **tasso d'errore**, che corrisponde alla Formula 4.4. Si contano quanti errori si son fatti rispetto al totale o in alternativo si usa il metodo opposto ossia l'**accuratezza**, definita nella Formula 4.5, che conta i risultati con esito positivo rispetto al totale.

$$T_{err} = \frac{N(errori)}{N(previsioni)} \quad (4.4)$$

$$Acc = 1 - T_{err} = 1 - \left(\frac{N(errori)}{N(previsioni)} \right) = \frac{N(corretti)}{N(previsioni)} \quad (4.5)$$

Se si vogliono fare delle misurazioni particolari si sfrutta invece la **matrice di confusione**, mostrata in Tabella 4.4, ideata inizialmente per valutare casi con solo due opzioni (Si-positivo / No-negativo). Può essere generalizzata per gestire la valutazione di un classificatore con più di due esiti, tramite due approcci differenti.

- si amplia la matrice per avere invece che due righe e due colonne, K righe e K colonne.
Dove K è il numero di possibili valori ritornati dal classificatore
- si possono creare K matrici di confusione e ogni volta si sceglie una specifica classe come positiva, tutte le altre vengono implose nel valore negativo

		Previsione	
		Sì	No
Risposta	Sì	TP	FN
corretta	No	FP	TN

TP	True Positive
FN	False Negative
FP	False Positive
TN	True Negative

Tabella 4.4: Viene rappresentata una matrice di confusione

Tabella 4.5: Significato delle sigle della matrice di confusione

Da questa matrice possiamo estrarre diverse misurazioni:

- **accuratezza** qui prende una nuova formulazione: $Acc = \frac{TP + TN}{TP + TN + FP + FN}$
- **sensibilità** penalizza le risposte negative sbagliate (FN), di conseguenza è meglio dare per positivo un esito se non è certo.
Definita come: $S = \frac{TP}{TP + FN}$
- **precisione** penalizza le risposte positive sbagliate (FP), di conseguenza è meglio dare per negativo un esito se non è certo.
Definita come: $P = \frac{TP}{TP + FP}$
- **Macro** precisione/sensibilità è la media aritmetica fra più valori di S o di P.
Definita come: $MacroS = \frac{S_1 + \dots + S_n}{n}$
- **Micro** precisione/sensibilità è la media aritmetica delle componenti di S o di P prima di calcolarle, per poi considerarle un'unica S o P.
Definita come: $MicroP = \frac{TP_1 + \dots + TP_n}{(TP_1 + \dots + TP_n) + (FP_1 + \dots + FP_n)}$
- **F1-Score** è la media armonica di: micro/macro/normale sensibilità e precisione.
Definita come: $F1 - Score = \frac{2}{\frac{1}{S} + \frac{1}{P}} = \frac{2 \cdot S \cdot P}{S + P}$

Nel caso mostrato, sensibilità e precisione non possono essere utilizzate in quanto ci sono troppi dati; mentre usare micro e macro non ha senso in quanto le classi sono perfettamente equivalenti, di conseguenza premiarne una a discapito di un'altra non ha significato. Per questo se serve si adotta l'F1-score. Tuttavia questa procedura non è veloce da calcolare, pertanto spesso si adotta il più semplice e veloce metodo dell'accuratezza.

4.4.2 Esempio

ID	x	Previsione	Classe	Corretta
5	9	0	1	0
1	13	0	0	1
2	15	0	0	1
3	27	0	1	0
4	34	1	1	1
8	36	1	0	0
6	48	2	2	1
7	59	2	2	1

Tabella 4.6: Tabella riassuntiva del procedimento di classificazione dei vertici

Tutti i dati dell'esempio sull'algoritmo di classificazione sono riassunti nella Tabella 4.6. Nella prima colonna abbiamo gli ID dei nodi coinvolti. La seconda colonna contiene il vettore rappresentativo di ogni nodo, qui per semplicità è stato ridotto ad un solo numero.

$$C(x) = \begin{cases} 0 & \text{se } x < 30 \\ 1 & \text{se } 30 \leq x \text{ \& } x < 45 \\ 2 & \text{se } 45 \leq x \end{cases} \quad (4.6)$$

Si assume il classificatore definito nella Formula 4.6, che darà come risposte i valori contenuti nella colonna "Previsione".

Mentre nella colonna "Classe" è mostrato il valore autentico del nodo. Infine nell'ultima colonna è indicato se la previsione è risultata corretta o meno.

Per una valutazione mediante accuratezza è sufficiente calcolare la media aritmetica dell'ultima colonna, ossia $Acc = \frac{1+1+1+1+1}{8} = \frac{5}{8} = 0.625$. Mentre per l'F1-score sarebbe necessario calcolare sensibilità e precisione di ognuna delle 3 classi, riassumerli mediante micro o macro, ed infine arrivare ad un valore con F1-score.

4.4.3 Applicazioni

I grafi scelti per questa applicazione sono Cora (Figura 4.5) e Citeseer (Figura 4.6) perché fra quelli presentati nella Tabella 4.1 sono gli unici che sono forniti di attributi sui nodi. Questo permette di mostrare come l'utilizzo del grafo bipartito per l'introduzione degli attributi, modifica le prestazioni.

Sull'asse delle ordinate vi sono i valori compresi nell'intervallo $[0,1]$ misurati mediante la tecnica

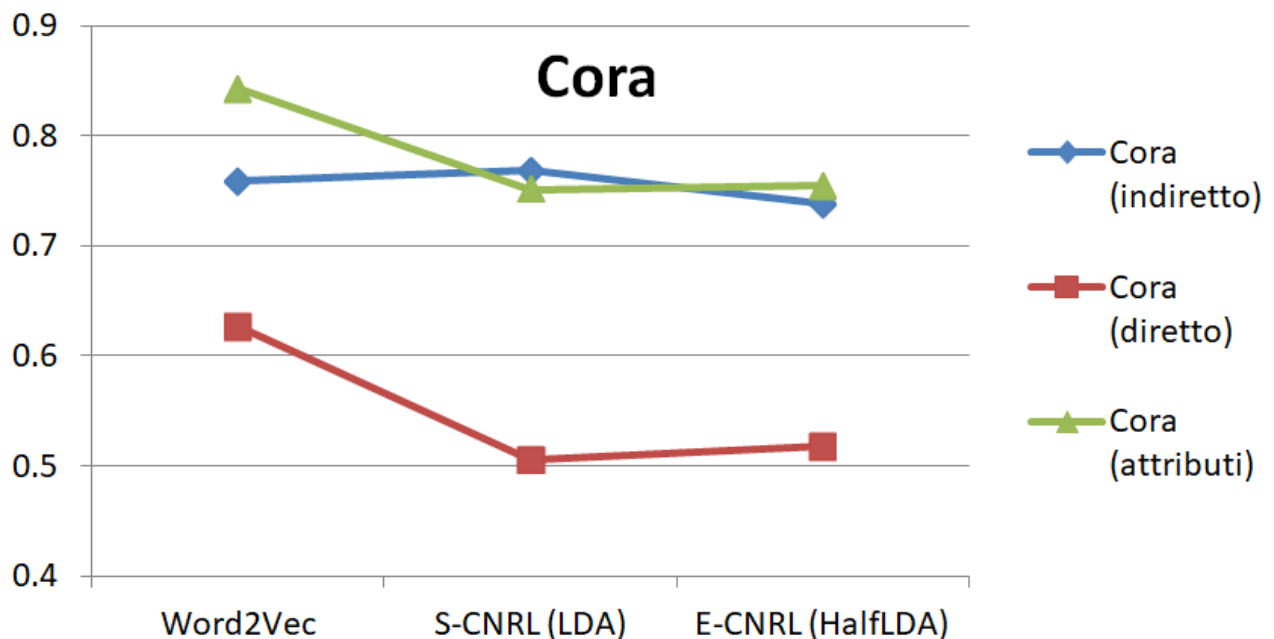


Figura 4.5: Andamento del grafo Cora , con 3 differenti algoritmi di creazione del modello e 3 diverse interpretazioni iniziali, valutato mediante l'accuratezza

dell'accuratezza. Mentre sull'asse delle ascisse vi sono i tre algoritmi utilizzati per creare il modello del grafo partendo dai cammini. Word2Vec è il metodo di controllo, mentre S-CNRL e E-CNRL che lavorano rispettivamente con LDA e HalfLDA, sono i nuovi algoritmi proposti. Entrambi per alcune parti continuano ad usare Word2Vec .

Sia Cora che Citeseer sono grafi originariamente diretti (**misurazioni rosse**), tuttavia si è provato ad interpretarli anche come se fossero indiretti (**campionamento blu**), l'ultimo andamento è dato dall'introduzione degli attributi (**dati verdi**).

Ecco alcuni dettagli che emergono dai due grafici:

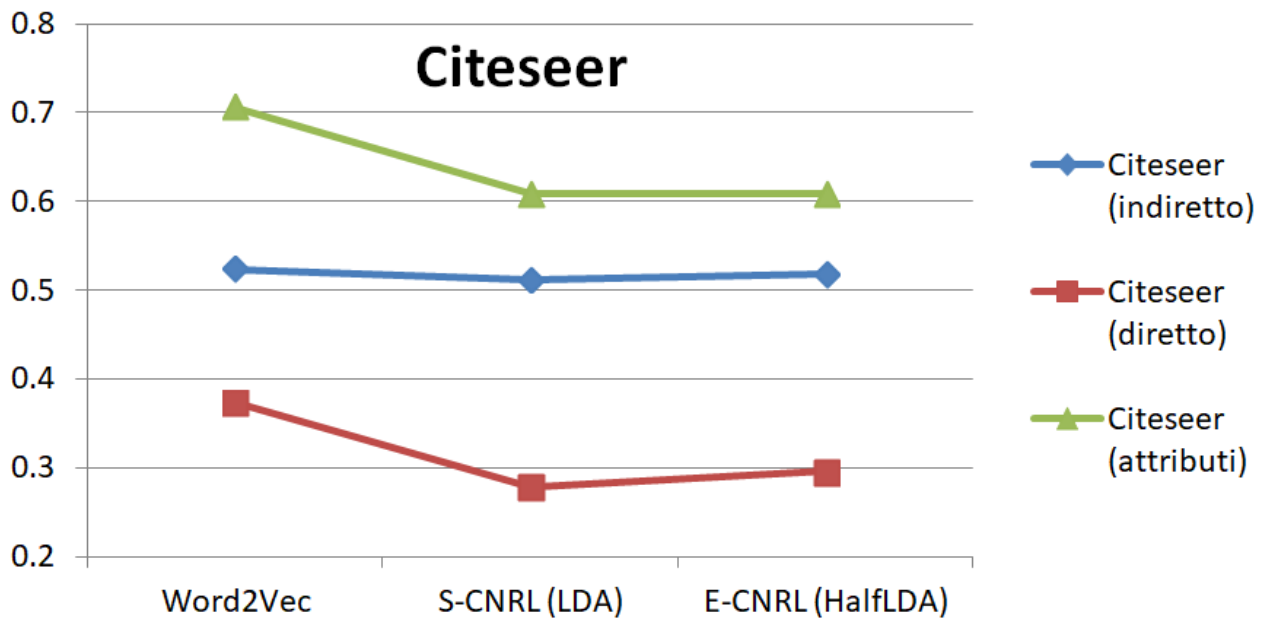


Figura 4.6: Andamento del grafo Citeseer , con 3 differenti algoritmi di creazione del modello e 3 diverse interpretazioni iniziali, valutato mediante l'accuratezza

- Word2Vec mostra prestazioni, anche se di poco, maggiori delle due alternative che invece tendono ad equivalersi
- la media dei valori delle misurazione nel grafico di Cora (Figura 4.5) è 0.696 rispetto a quella di Citeseer (Figura 4.6) che è 0.491, la prima risulta essere maggiore di circa $\frac{2}{10}$ questo è dovuto anche al fatto che Cora abbia una $densita' = \frac{archi}{nodi}$ maggiore rispetto a Citeseer
- un alta densità degli archi permette di creare molte interazioni fra i nodi e quindi il modello meglio rappresenta il grafo. Si noti però che questo non è sempre facile da individuare, qui è presente uno stacco evidente in quanto entrambi i grafi hanno un estremamente bassa densità si dice infatti che sono sparsi
- si è provato ad aumentare la densità, considerando ambo i grafi come indiretti (ogni arco indiretto può essere rappresentato come due archi diretti). Si ha quindi poco meno di un raddoppio della densità, è infatti possibile che un arco avesse già il suo opposto e quindi non risenta del cambiamento.
L'effetto per entrambi è stato un innalzamento del valore medio lungo i 3 algoritmi usati
- con il grafo indiretto l'incremento di S-CNRL e E-CNRL è maggiore rispetto a quello di Word2Vec . Questo può indicare, che Word2Vec lavora molto bene con i grafi orientati a dispetto degli altri due metodi
- l'introduzione degli attributi tramite il grafo bipartito cambia notevolmente la situazione portando ad un incremento di prestazioni. Questi valori vanno confrontati con i dati del grafo orientato in quanto quello è il valore corretto. I punti blu del grafo non-orientato non sono comparabili in quanto servono a mostrare come la densità influisce sul risultato, rappresentano un nuovo grafo diverso dall'originale, e quindi non paragonabile

4.5 Valutazione dell'individuazione di comunità

Questa sezione ripercorre la parte principale del tirocinio, ossia l'integrazione degli attributi e la valutazione di un partizione mediante la modularità. Vengono presentate quattro situazioni ognuna approfondisce un aspetto affrontato.

4.5.1 Confronto numero comunità

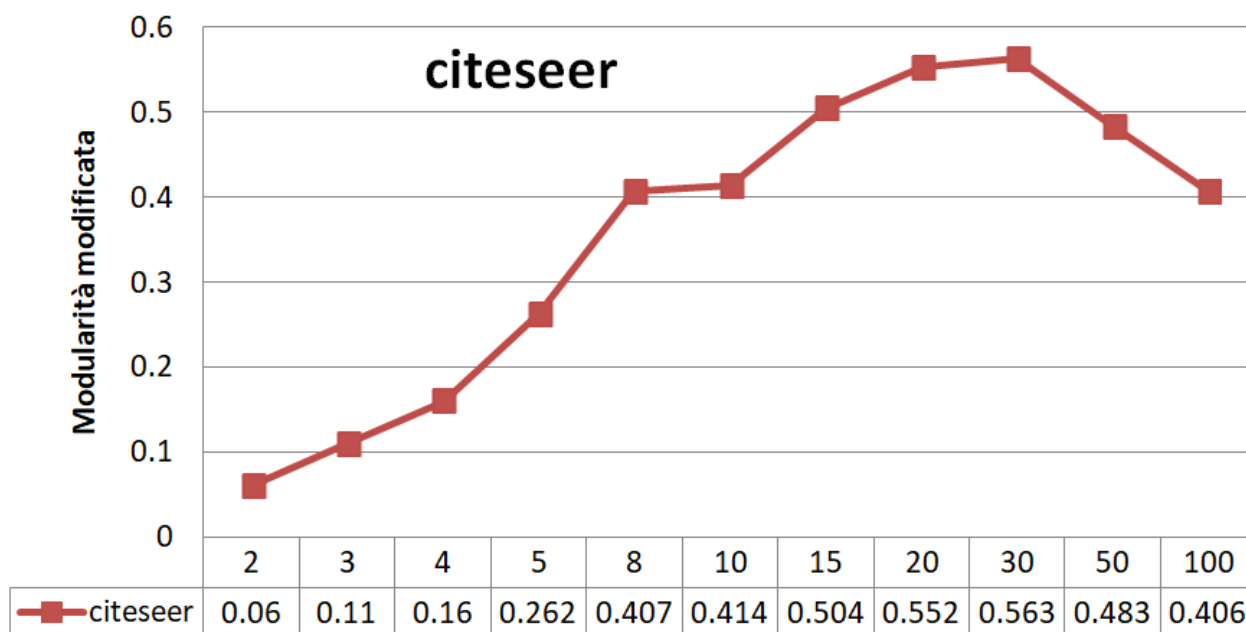


Figura 4.7: Accuratezza rappresentativa al variare del numero delle comunità da generare

In Figura 4.7 è possibile vedere l'andamento dell'algoritmo d'individuazione delle comunità LDA, applicato al grafo Citeseer di cui si considera solo la struttura, la valutazione vien fatta con la modularità modificata .

L'unico parametro che viene modificato è mostrato sull'asse delle ascisse ed è il numero di comunità che si vanno a ricercare all'interno del grafo. Ecco cosa s'osserva:

- la progressione del numero di comunità da ricercare è stata scelta per mostrare la rapida crescita della modularità quando piccoli numeri sono coinvolti
- il valore calcolato su 2 comunità è estremamente basso, perché difficilmente riesce a rappresentare a dovere una vasta selezione di nodi
- la modularità cresce con l'aumentare delle comunità, che premiano sempre più le peculiarità del grafo
- c'è un massimo compreso fra 30 e 50 dove questa crescita s'interrompe
- gli ultimi due dati 50 e 100 decrescono, questo è noto come il **problema dell'overfitting**, a causa di un addestramento eccessivo, il risultato raggiunto non è più in linea con le aspettative. Via via che i gruppi di nodi si stringono perdono di rappresentazione portando così ad un decremento della modularità. Si osserva questo fenomeno anche in Figura 4.8

4.5.2 Metriche di valutazione

Nella Tabella 4.7 vengono mostrati i due grafi di Cora e Citeseer , valutati mediante tre differenti metriche di calcolo della modularità.

Ambo i grafi sono stati caricati senza considerare gli attributi dei nodi, ma solo la loro struttura. La creazione del modello è avvenuta con S-CNRL e il suo algoritmo LDA, tramite il quale sono state ricavate le 10 comunità che vengono valutate in Tabella.

Non ha senso confrontare i tre differenti metodi di calcolo della modularità, in quanto premiano aspetti differenti. Inoltre ognuno si posiziona su un diverso ordine di grandezza. Spicca soprattutto `mod_overlap` che presenta tutti i valori molto vicini a 0, questo perché, come riporta la Tabella 4.1, ambo i grafi sono estremamente sparsi e, a causa di ciò, `mod_overlap` ha il valore assoluto della modularità calcolata

vicino a 0.

Tutti i metodi assegnano a Cora il valore più alto, e questo fa capire che, se una partizione è nettamente migliore di un'altra, allora verrà premiata indipendentemente dalla metrica di valutazione usata.

La modularità modificata è stata scelta come metrica ufficiale degli esperimenti, non solo per il fatto che è quella usata nell'articolo di CNRL, ma anche perché presenta dei valori facilmente analizzabili e comprensibili. `mod_overlap` funziona bene ma è sempre difficile comprendere i risultati a cui s'arriva rischiando incomprensioni.

	<code>mod_withMax</code>	<code>mod_overlap</code>	modularità modificata
Cora	$2.04 \cdot 10^{-1}$	$1.21 \cdot 10^{-5}$	$5.31 \cdot 10^{-1}$
Citeseer	$8.42 \cdot 10^{-2}$	$-3.55 \cdot 10^{-6}$	$3.67 \cdot 10^{-1}$

Tabella 4.7: Confronto fra i metodi di modularità spiegati, partendo dal grafo normale con 10 comunità

4.5.3 Confronto metodi d'elaborazione

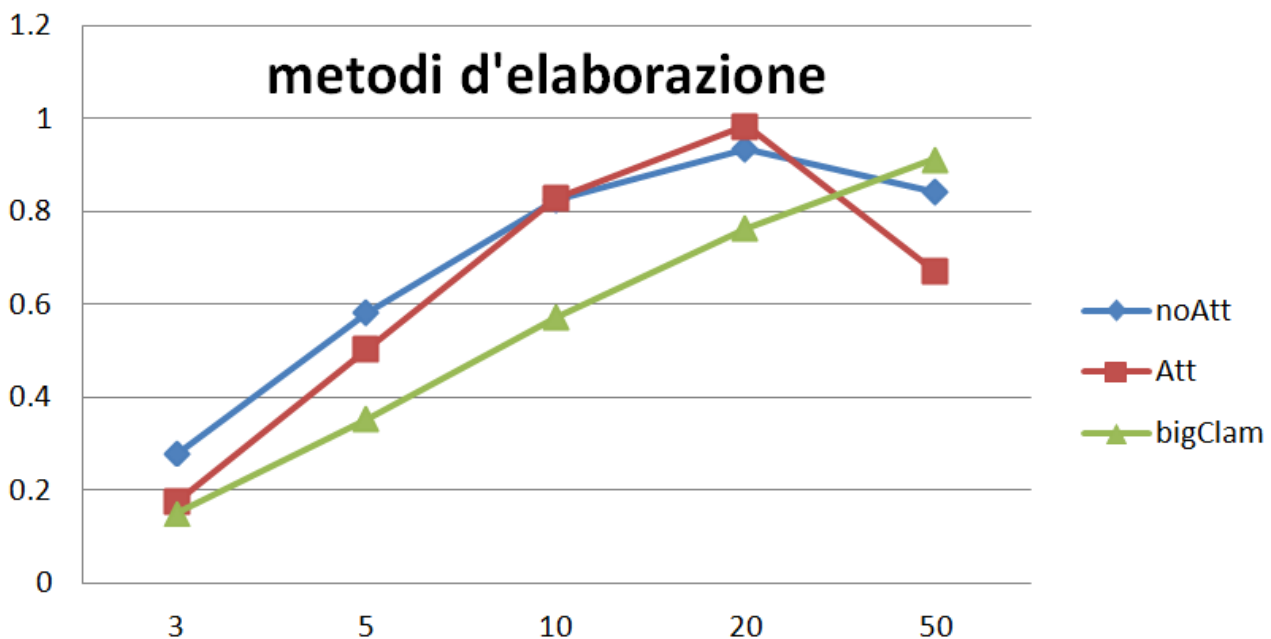


Figura 4.8: Prestazioni dei diversi procedimenti d'elaborazione del modello al variare del numero di comunità da ricercare (grafo Cora)

In Figura 4.8 vengono confrontati tre differenti metodi d'elaborazione dei grafi, il cui scopo è trovare le comunità.

Viene presentato il grafo Cora su cui si è applicata una progressione da 3 a 50 comunità, valutate tramite la modularità modificata.

I tre metodi utilizzati sono:

- **noAtt**: si considera in partenza unicamente la struttura del grafo, dai cui si ricavano i cammini e vengono elaborati con LDA
- **Att**: oltre ai cammini derivanti dalla struttura si crea anche il grafo bipartito degli attributi, il tutto elaborato sempre mediante LDA
- **bigClam**: è il metodo tratto dall'articolo "Overlapping community detection at scale: a non-negative matrix factorization approach"[22], implementato dall'università di Stanford[19]. È stato scelto come algoritmo di controllo per verificare l'andamento delle prestazioni di S-CNRL

Dalla figura possiamo capire:

- come avviene in Figura 4.7, con l'aumentare del numero di comunità si ha il problema dell'overfitting.
Per Citeseer il problema si rivela quando si cercano 50 comunità e così è per Cora, questo sembra non valere per bigClam anche se è inevitabile che accada con numeri molto alti di comunità da individuare
- durante la crescita le prestazioni di bigClam restano costantemente inferiori
- l'apice è raggiunto mediante gli attributi con 20 comunità ad un valore di $Q = 0.986$. È importante notare che non tutti gli apici dei vari metodi si posizionano a 20 comunità. Al contrario, BigClam sembra aver questo massimo oltre le 50 comunità
- valutare il grafo mediante e senza gli attributi sembra non influire troppo, anche se così non dovrebbe essere (caso meglio analizzato in Tabella 4.8)

4.5.4 Confronto criteri di valutazione

Cora		valutazione	
		struttura	ipergrafo
elaborazione	noAtt	$8.24 \cdot 10^{-1}$	$9.83 \cdot 10^{-3}$
	Att	$8.29 \cdot 10^{-1}$	$2.33 \cdot 10^{-2}$

Citeseer		valutazione	
		struttura	ipergrafo
elaborazione	noAtt	$4.14 \cdot 10^{-1}$	$8.86 \cdot 10^{-6}$
	Att	$4.14 \cdot 10^{-1}$	$2.30 \cdot 10^{-2}$

Tabella 4.8: Misurazione delle comunità, generate attraverso due differenti metodi d'elaborazione, analizzate su due tipologie di valutazione

Nella Tabella 4.8 vengono mostrati entrambi i grafi di Cora e Citeseer, su ognuno sono state fatte quattro differenti valutazioni.

Sulle righe vi sono i metodi d'elaborazione utilizzati per trovare le comunità:

- **noAtt** indica che del grafo è stata utilizzata solamente la struttura per la generazione dei cammini
- **Att** indica invece che oltre ai dati della struttura vengono utilizzati anche gli attributi introdotti tramite il grafo bipartito

Sulle colonne sono invece riportati i due metodi per la valutazione della partizione generata:

- **struttura** indica che il grafo utilizzato per la valutazione, tramite modularità modificata, è quello originale, che contiene solo la struttura
- **ipergrafo** il grafo per la valutazione con modularità modificata contiene anche gli archi degli attributi, non è possibile utilizzare un grafo bipartito in quanto contiene nodi (l'insieme N_2) non appartenenti al grafo originario, si opta quindi per l'ipergrafo.
Quello qui utilizzato, rispetto a quello mostrato in Figura 2.4, mantiene anche gli archi originali e per non creare incoerenze considera tutti gli archi come diretti, ciò porta a più d'un raddoppio del numero d'archi, partendo da un numero base già elevatissimo

Dalla Figura 4.8 si può vedere che con 10 comunità la valutazione di Cora, elaborato con e senza attributi, è molto simile. Si è scelto d'utilizzare le 10 comunità per osservare le variazioni quando si adotta una valutazione attraverso l'ipergrafo.

Ecco cosa si può notare:

- per la scelta del numero di comunità i dati di Cora, valutati sulla struttura, sono pressoché uguali
- per Citeseer accade lo stesso, qui i due valori sono identici.
Il dato di $4.14 \cdot 10^{-1}$ è già apparso in Tabella 4.7

- se la valutazione avviene sull'ipergrafo sia nel caso di Cora che di Citeseer il valore più alto viene generato con l'elaborazione effettuata sugli attributi.
È in linea con le aspettative, perché i cammini aggiunti dal grafo bipartito vanno a descrivere una sezione che l'elaborazione, utilizzando solo la struttura, non può conoscere e quindi non vi si può adattare
- tutte le valutazioni sull'ipergrafo sono così vicine allo 0 perché l'Espressione 3.7, parte della Formula 3.5, della modularità modificata, in presenza di un grafo molto denso, ed è questo il caso, abbassa notevolmente il valore della misurazione

Nonostante i dati mostrati in Tabella 4.8 siano perfettamente in linea con le aspettative, non è possibile affermare che quanto mostrato in questa sottosezione sia sempre vero in ogni possibile situazione, in quanto i risultati restituiti dalla modularità modificata presentano una non trascurabile variabilità.

5 Conclusioni

Seguono alcune considerazioni personali sul lavoro affrontato e su come è stato svolto.

Conclusioni tecniche

Vorrei evidenziare le potenzialità dell'uso degli attributi nella creazione dei modelli.

Durante il tirocinio sono stati indagati solo alcuni dei metodi esistenti per l'introduzione di questi dati aggiuntivi. Ne esistono molti altri, ognuno con i suoi vantaggi e svantaggi. Non so se gli approcci da noi utilizzati sono i migliori o quelli con più potenzialità, ma comunque abbiamo riscontrato che lavorano bene e hanno portato ad un miglioramento.

Uno dei vantaggi principali è che, basandosi su delle meccaniche semplici, è potenzialmente fattibile integrarli con altri metodi, portando ad ulteriori miglioramenti. Uno svantaggio evidente è il tempo d'esecuzione. Il codice iniziale di CNRL è molto veloce, l'introduzione delle meccaniche proposte in questa tesi hanno appesantito molto il costo computazionale. Questo è ancor più evidente quando, come nel nostro caso, si lavora con grafi estremamente sparsi che però dispongono di molti attributi. Probabilmente questo approccio dovrà essere ancora e ancora sviluppato, ma non è escluso che potrà dare un suo valido contributo.

Conclusioni personali

Affacciarsi per la prima volta a questo campo è molto complesso.

È possibile, anche se estremamente difficoltoso, comprendere tutte le meccaniche degli algoritmi che si vanno ad utilizzare. Essendo un progetto molto vasto personalmente ho preferito appoggiarmi a svariate parti di codice già ideate, scritte e testate.

I dataset su cui si va ad operare sono troppo ampi e questo comporta che non se ne possa tener traccia. Gli errori sono complicati da individuare poiché non è possibile eseguire manualmente le operazioni svolte dal codice, nonostante si conoscano nel dettaglio come lavorano gli algoritmi. È fondamentale non sottovalutare la mole del progetto su cui si lavora e non tralasciare nessun dettaglio finché è ancora possibile individuarlo, testarlo e correggerlo.

La complessità maggiore che personalmente ho incontrato sta nel settaggio dei parametri.

Visto la moltitudine di possibilità di cui si dispone, risulta complesso abbracciarle tutte, per comprendere come impostare i parametri che regolano la gestione dei dati all'interno del codice. Spesso alcuni settaggi sono disponibili solo in alcune situazioni o dipendono da delle impostazioni. Un'esecuzione avvenuta con parametri errati non rispecchia le aspettative, e porta a dover rivedere il tutto.

In conclusione, le tante sfaccettature di questo argomento lo rendono complesso e difficile, è quindi necessario prestare sempre estrema attenzione al metodo di lavoro.

Bibliografia

- [1] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. Journal of machine Learning research, 3(Jan):993–1022, 2003.
- [2] Gobinda G Chowdhury. Introduction to modern information retrieval. Facet publishing, 2010.
- [3] Cunchao Tu; Hao Wang; Xiangkai Zeng; Zhiyuan Liu; Maosong Sun. simplified CNRL, 2017.
- [4] D. Lusseau; K. Schneider; O. J. Boisseau; P. Haase; E. Slooten; S. M. Dawson. social network of dolphins. <http://www.cise.ufl.edu/research/sparse/matrices/Newman/dolphins>, 2003.
- [5] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864. ACM, 2016.
- [6] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. Radiology, 143(1):29–36, 1982.
- [7] Anna Lázár, Dániel Abel, and Tamás Vicsek. Modularity measure of networks with overlapping communities. EPL (Europhysics Letters), 90(1):18001, 2010.
- [8] Huan Liu Lei Tang. Blogcatalog data. http://leitanet.net/social_dimension.html.
- [9] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. ACM Transactions on Knowledge Discovery from Data (TKDD), 1(1):2, 2007.
- [10] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. arXiv preprint arXiv:1705.04969, 2017.
- [11] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. Physica A: statistical mechanics and its applications, 390(6):1150–1170, 2011.
- [12] Qing Lu and Lise Getoor. Link-based classification. In Proceedings of the 20th International Conference on Machine Learning (ICML-03), pages 496–503, 2003.
- [13] David Meyer. How exactly does word2vec work? 2016.
- [14] Mark EJ Newman. Clustering and preferential attachment in growing networks. Physical review E, 64(2):025102, 2001.
- [15] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 701–710. ACM, 2014.
- [16] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. arXiv preprint cs/0209028, 2002.
- [17] Ryan Rossi and Nesreen Ahmed. Network repository. <http://networkrepository.com>, 2013.

- [18] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. AI magazine, 29(3):93, 2008.
- [19] Stanford Network Analysis Platform. snap-dev-64. <https://github.com/snap-stanford/snap-dev-64/tree/master/examples/bigclam>.
- [20] Cunchao Tu, Hao Wang, Xiangkai Zeng, Zhiyuan Liu, and Maosong Sun. Community-enhanced network representation learning for network analysis. arXiv preprint arXiv:1611.06645, 2016.
- [21] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. Network representation learning with rich text information. In IJCAI, pages 2111–2117, 2015.
- [22] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In Proceedings of the sixth ACM international conference on Web search and data mining, pages 587–596. ACM, 2013.
- [23] Wayne Zachary. Karate club. <https://www.cise.ufl.edu/research/sparse/matrices/Newman/karate.html>, 1977.
- [24] Hongyi Zhang, Irwin King, and Michael R Lyu. Incorporating implicit link preference into overlapping community detection. In AAAI, pages 396–402, 2015.
- [25] Zhiya zhiyzuo. python-modularity-maximization. <https://github.com/zhiyzuo/python-modularity-maximization>.
- [26] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. The European Physical Journal B, 71(4):623–630, 2009.