UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTO DI INTELLIGENZA ARTIFICIALE

TEXTUAL CLASSIFICATION WITH GCN

CLASSIFICAZIONE TESTUALE MEDIANTE GCN

Docenti: Monica Mordonini - Stefano Cagnoni

Correlatori: Gianfranco Lombardo - Paolo Fornacciari

Studenti: Matteo Gatti - Matricola: 289639, Davide Zanella - Matricola: 293074

ANNO ACCADEMICO 2018-2019

Indice

Elenco delle figure

Elenco delle tabelle

1 Descrizione del problema

Negli ultimi anni, il graph machine learning ha riscosso grande interesse grazie alla sua capacità di modellare nativamente ogni tipo di informazione espressa sotto forma di un grafo (solo per citare alcuni esempi: amicizie ed interessi sui social network).
Il machine learning permette di approcciare un problema di classificazione in molti modi. Un approccio classico viene detto induttivo. In questo caso, nell'addestrare il modello, la rete vede solo ed esclusivamente il training set e ci si aspetta che, ultimata la fase di training, il modello riesca a generalizzare su un dataset che non ha mai visto. L'approccio trasduttivo invece prevede che la rete non veda solo il training set ma anche le features degli altri dati da classificare. Questo approccio assomiglia un po' al modo di pensare umano: l'uomo non impara solo per imitazione ma anche attraverso analogie e ragionamento.
Il nostro progetto consisteva nell'utilizzare le Graph Convolutional Network (GCN) per la classificazione testuale. Ci è stato fornito un dataset di 10000 frasi riguardanti recensioni di film ottenuto attraverso il sito IMDB. Di queste frasi 5000 erano recensioni positive e 5000 negative.
Al dataset abbiamo applicato alcune tecniche di preprocessing. Ad esempio abbiamo eliminato le cosiddette stop-words, ovvero quei termini che, data la loro elevata frequenza in un'idioma, sono poco significativi in una ricerca bibliografica. Un'altra tecnica di preprocessing usata ci ha permesso di non considerare la differenza tra parole che iniziano con una lettera maiuscola o minuscola. Ad esempio la parola "Once" e la parola "once" sono state gestite come se fossero un'unica parola nel nostro modello Bag of Words (BoW).
L'obiettivo che ci siamo posti consisteva nel verificare se le GCN consentissero una migliore accuratezza rispetto ai classificatori tradizionali soprattutto mettendogli a disposizione un training set piccolo.
La GCN, che sarà descritta più nel dettaglio nel capitolo successivo, lavora su un grafo. Abbiamo quindi trasformato ogni frase del nostro dataset in un nodo. Per collegare i vari nodi abbiamo seguito due strade:

- Archi ottenuti mediante similarità sintattica, usando il PoS Tagging.
- Archi ottenuti usando basandosi su similarità semantiche.

In entrambi i casi per stabilire il peso di ogni arco, che indica quanto sono legati i vari nodi, abbiamo applicato una cosine similarity. Il risultato ottenuto è stato poi filtrato in modo da considerare solo gli archi aventi un certo peso. Questo è stato necessario per due motivi:

- Eliminare i collegamenti tra i nodi con una bassa correlazione reciproca.

- Abbassare la grande mole di informazioni da dare in pasto alla GCN per ridurre il tempo di elaborazione.

Per inizializzare la nostra GCN abbiamo quindi considerato solo i nodi connessi, eliminando i nodi con grado zero.
Nei prossimi capitoli saranno spiegate nel dettaglio le scelte fatte e i risultati ottenuti.

2 GCN: cosa sono e come funzionano

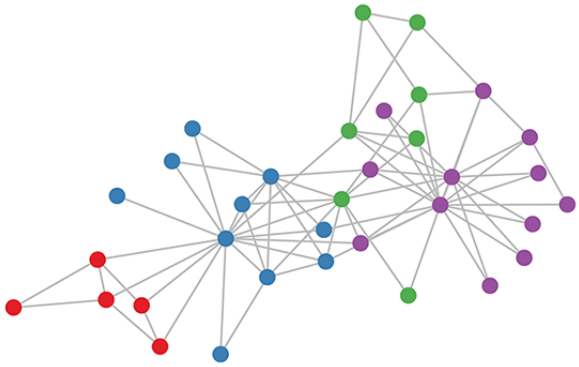

Figura 1: Schema di un grafo

Quando si parla di GCN si fa riferimento ad un'architettura universale che molti modelli di reti neurali basate su grafi hanno in comune. Gli input del modello sono una matrice X di dimensione NxD (N rappresenta il numero di nodi mentre D il numero di features) e una matrice di adiacenza A. Le features contenute nella matrice X caratterizzano ogni nodo. Nel nostro caso ogni nodo corrisponde ad una frase ed è caratterizzato da un ID univoco, dal suo Tf-IDF e dalla sua label.
La matrice di adiacenza A è invece una descrizione del grafo, ogni posizione della matrice di adiacenza specifica la connessione (eventualmente pesata) tra due nodi.

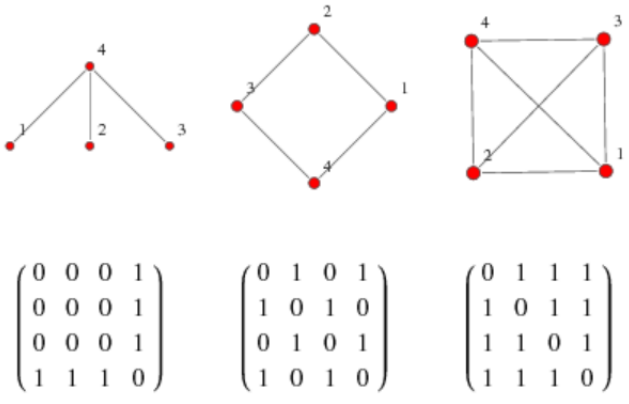

Figura 2: Esempi di matrici di adiacenza.

Una GCN può essere definita dalla seguente equazione:

$$(X) = ReLU(AXW); \tag{1}$$

dove A è proprio la matrice di adiacenza citata prima. Possiamo concepire questa equazione come composta di tre parti in successione:

- Processo il segnale su ciascun nodo in maniera indipendente tramite la moltiplicazione con W.
- La moltiplicazione per la matrice di adiacenza produce una seconda combinazione lineare, fra i segnali di ogni nodo e dei rispettivi vicini. A differenza della prima fase, i coefficienti di questa combinazione sono fissi.
- La terza fase, nuovamente uguale a prima, applica la non-linearità scelta sui vari nodi.

La logica è simile a quella di una rete convolutiva: l'informazione elaborata da ciascun "filtro" dipende solo dagli immediati vicini del nodo, mentre le stesse operazioni (definite dalla matrice W) vengono applicate su tutti i nodi del grafo simultaneamente. Possiamo combinare più strati di questo tipo per ottenere architetture che "diffondono" l'informazione sul grafo stesso tramite ripetute moltiplicazioni per A:

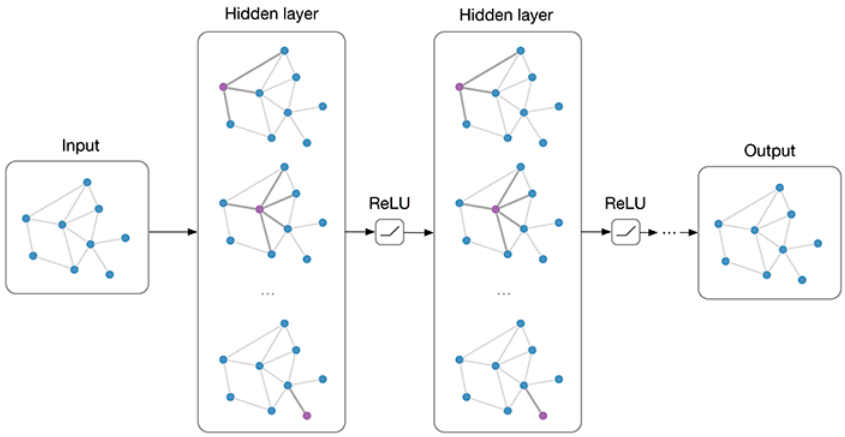

Figura 3: Schema di una GCN

Una rete formata da tre strati di questo tipo, anche senza ottimizzazione, produce una separazione fra i nodi del grafo molto simile alla loro reale categorizzazione:

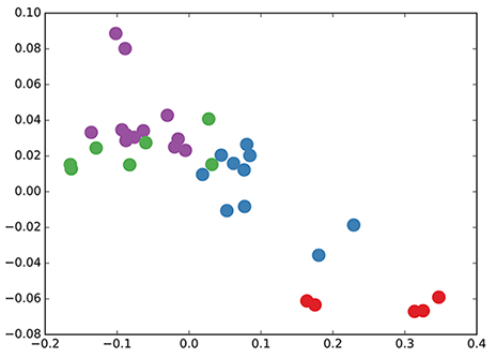

Figura 4: Embedding dei nodi del grafo con una GCN non allenata

3 Generazione del grafo

Il primo problema che è stato affrontato è stato quello della creazione del grafo. I nodi rappresentano ogni frase e sono caratterizzati da tre informazioni: la label (che può essere positiva o negativa), un ID univoco e il Tf-IDF calcolato sul modello BoW. Per quanto riguarda gli archi si è cercato un modo per legare le frasi tra loro. Come detto in precedenza abbiamo seguito due strade diverse:

- PoS Tagging, quindi attraverso una descrizione sintattica della frase.
- Tramite la descrizione semantica della frase.

Analizzeremo meglio queste due strade nei prossimi paragrafi.

3.1 Similarità Sintattica e PoS Tagging

Inizialmente, per poter creare un legame tra le frasi è stata usata una tecnica detta PoS Tagging. Attraverso questa tecnica è possibile fornire una descrizione sintattica della frase, etichettando ogni parola del nostro dataset sulla base della sintassi (nome, verbo, aggettivo, ecc).
In particolare è stata utilizzata la libreria sviluppata dall'Università di Stanford. Questo algoritmo prende come input una frase e restituisce una sequenza di PoS Tag che la descrivono. L'insieme di tutti i PoS Tag utilizzati per descrivere le varie frasi costituisce il dizionario sintattico del nostro dataset. Su ogni sequenza (quindi per ogni frase) è stato calcolato il Tf-IDF. Nel calcolo del Tf-IDF abbiamo considerato anche bigrammi e trigrammi dei PoS Tag in modo da avere un maggior numero di informazioni utili. In seguito abbiamo cercato di dare un peso ad ogni arco. Questo peso equivale al grado di similitudine che c'è tra le due frasi (quindi tra i due nodi del grafo) che vengono uniti dall'arco.

Il peso è stato calcolato mediante la funzione di cosine similarity. Questa funzione prende in ingresso la matrice dei valori del TfIdf e calcola il coseno tra coppie di frasi (le righe della matrice) prese a 2 a 2. L'output sarà un'altra matrice di dimensione NxN (dove N è il numero di nodi) contenente i risultati dell'operazione di cosine similarity. Questa è una matrice simmetrica in quanto il risultato della funzione è uguale per la coppia di frasi i,j e j,i. Sulla diagonale principale ci sono solo dei valori unitari dato che la cosine similarity di una frase con la frase stessa è pari a 1.
Una volta arrivati a questo punto si è filtrato il risultato con una soglia per rimuovere gli archi con un peso poco significativo. In questo modo si è cercato di creare un subset degli archi totali da dare poi in ingresso alla GCN. È stata fissata la soglia a 0.75, ciò ha consentito di avere circa 2.5 milioni di archi e 5012 nodi connessi.

3.2 Similarità semantica

La seconda via che è stata percorsa consisteva nel collegare le frasi sulla base della semantica delle stesse. Questo è stato possibile partendo dal modello Bag of Words. Si sono calcolati i valori di Tf-IDF partendo da tutte le features (parole) delle varie frasi. In questo caso la dimensione della matrice di output era molto elevata: avendo un dataset molto ampio ci si è trovati a gestire una matrice di circa 50 mila colonne, che rappresentano il numero di features totali del nostro dataset. Si è poi provato a usare bigrammi e trigrammi ma in questo caso il numero di colonne del Tf-IDF è salito vertiginosamente (800 mila nel primo caso e oltre 1.7 milioni nel secondo) diventando ingestibile a livello computazionale, pertanto abbiamo abbandonato questa strada optando per i monogrammi.
Per ridurre il numero di features è stato utilizzato un metodo chiamato *information gain*. Grazie a questo metodo infatti si possono selezionare le features più discriminanti che definiscono la positività o la negatività di una frase.
Questo è stato possibile utilizzando una funzione che prende in ingresso la matrice con i valori del Tf-IDF e restituisce un vettore contenente dei valori (non deterministici) che stimano la mutua informazione tra ogni feature e il target (nel nostro caso un vettore di p e n che indica se la frase è positiva o negativa). Una volta ottenuto questo vettore, sono state cancellate dalla matrice tutte quelle colonne (feature) il cui valore di mutua informazione è risultato essere pari a zero. Con questa selezione il numero delle features è sceso da 50 mila a 20 mila.
Utilizzando le 20 mila features rimaste, per ognuna di esse è stata calcolata la cosine similarity con tutte le altre features generando una matrice NxN simmetrica, come visto nel caso sintattico. Anche in questo caso è stata utilizzata una soglia per mantenere solo gli archi aventi un peso significativo. Sono stati dati in ingresso alla GCN i nodi e gli archi ottenuti con soglia 0.6 e 0.7:

- Soglia 0.6: 7058 nodi connessi e circa 600 mila archi

- Soglia 0.7: 4700 nodi connessi e circa 200 mila archi

Per visualizzare nodi e archi così ottenuti è stato utilizzato il software Gephi.

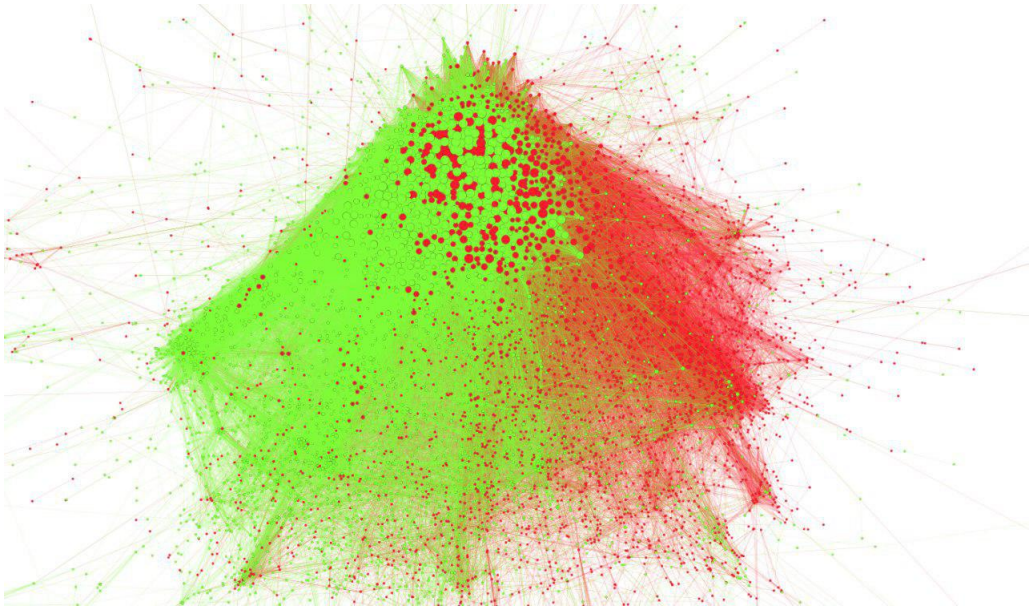

Figura 5: Grafo ottenuto mediante similarità semantica.

3.3 Gephi

Gephi è un software open source per l'analisi e la visualizzazione di grafi che è stato usato per poter valutare se i risultati ottenuti fossero corretti. Per poterlo utilizzare sono stati creati due file: un file di nodi, contenente l'id del nodo, la frase che rappresenta e la label relativa al nodo e un file di archi contenente il nodo di partenza, quello di destinazione e il peso dell'arco, dato dal valore di cosine similarity calcolato tra i due nodi. Una volta generati questi due file è stato creato il grafo e si è controllato che questo avesse una sola componente connessa (una volta eliminati gli eventuali nodi su cui non incide nemmeno un arco).
Applicando in Gephi vari algoritmi e evidenziando i nodi contenenti frasi con label p in verde e quelli con frasi con label n in rosso abbiamo ottenuto i seguenti risultati:

3.4 Considerazioni

Va sottolineato che il dataset utilizzato è costituito da solo due classi (frasi positive o negative). Entrambi i metodi usati per la creazione di un grafo portano ad avere

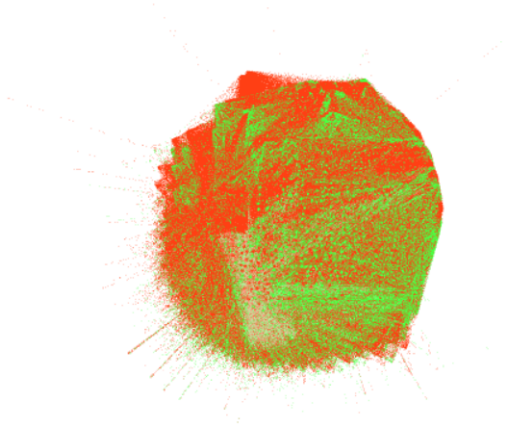

Figura 6: Grafo ottenuto mediante PoS Tagging.

grafi con diversi archi “errati”, ovvero archi in cui frasi positive sono connesse a frasi negative e viceversa. Non è affatto facile definire delle relazioni, delle similarità tra le frasi, si tratta di un open problem di language processing. In questo caso sono state percorse due strade: similarità sintattica e similarità semantica, come si vedrà più avanti la seconda strada è quella che ha portato i risultati migliori.

4 Prove e risultati ottenuti

Prima di lanciare la GCN sono stati generati due file che contenessero tutte le informazioni necessarie alla rete neurale per essere correttamente inizializzata. Il file per gli archi contiene per ogni riga le coppie dei nodi e il peso dell'arco che li collega. Il file dei nodi contiene, per ogni riga, l'ID del nodo, il valore del Tf-IDF e la label del nodo. È stata implementata un'apposita funzione per creare la matrice di adiacenza A e il vettore delle label partendo dai file menzionati in precedenza.
Il codice originale delle GCN proposto da Kipf utilizzava sempre gli stessi set di train, validation e test. In particolare venivano sempre selezionati gli stessi nodi per il training, per il test set e per il validation set. Questo però è un errore nell'ambito del machine learning pertanto è stata implementata una nuova funzione per la selezione casuale degli elementi all'interno del dataset. Ad ogni inizializzazione della GCN i nodi (e i relativi archi) usati come train vengono scelti casualmente. I nodi e gli archi rimanenti vengono suddivisi, sempre in modo casuale, tra test set e validation set. Ovviamente sono stati inseriti i controlli necessari per evitare che il medesimo nodo comparisse, ad esempio, sia nel training set che nel test set.
I test sono stati effettuati utilizzando diverse percentuali per il training set: 5, 10, 20, 40 e 70 percento.
Prima di iniziare a discutere dei test che sono stati effettuati vediamo una breve introduzione sui due tipi di filtri usati per effettuare le diverse prove: il filtro Localpool e il filtro di Chebyshev.

4.1 Filtro Localpool

Il filtro Localpool è stato introdotto nell'articolo *"Semi-Supervised Classification with Graph Convolutional Networks"* di Thomas Kipf e Max Welling dove è anche chiamato *"Renormalization Trick"*.
Una possibile equazione che descrive la propagazione delle informazioni nella GCN è la seguente:

$$f(H^{(l)},A) = \sigma(AH^{(l)}W^{(l)}); \tag{2}$$

Questa legge ha due limiti evidenti. La moltiplicazione di H per A significa che, per ogni nodo, vengono sommate tutte le features dei nodi vicini ma non il nodo stesso (a meno che ci siano dei self-loops che però non sono state introdotte nel grafo).
Il secondo limite è dato dal fatto che A non è normalizzata e il prodotto per A va a modificare completamente il vettore delle features. Per risolvere questo problema si moltiplica A per D^{-1}, dove D è una matrice diagonale. Così facendo si ottiene però la media delle features dei nodi vicini. Una possibile miglioria consiste nel moltiplicare a destra e a sinistra A per $D^{\frac{-1}{2}}$. Così facendo si opera una normalizzazione simmetrica (che non equivale più alla semplice media dei nodi vicini).

Si giunge quindi alla seguente formula che è stata enunciata anche nel paper di Kipf e Welling:

$$f(H^{(l)},A) = \sigma(\widehat{D}^{\frac{-1}{2}}\widehat{A}\widehat{D}^{\frac{-1}{2}}H^{(l)}W^{(l)}); \tag{3}$$

dove

$$\widehat{A} = A + I \tag{4}$$

e $\widehat{D}$ è la matrice diagonale con un numero di colonne pari al grado dei nodi del grafo. E' questo il cosiddetto *"Renormalization Trick"* che è necessario per evitare che la continua applicazione della legge costitutiva della GCN porti a instabilità numeriche.

4.2 Filtro di Chebyshev

In matematica, i polinomi di Čebyšëv, o Chebyshev, sono le componenti di una successione polinomiale che inizia con i seguenti polinomi:
T_0(x) = 1
T_1(x) = x
T_2(x) = 2x^2-1
Il filtro di Chbyshev si basa sull'omonimo polinomio di espansione. È una tecnica che è stata introdotta a causa dell'elevato costo computazionale del filtraggio di un segnale nel dominio di Fourier. Questa espansione polinomiale è usata tradizionalmente nella Graph Signal Processing per approssimare i kernel.

4.3 Test con similarità sintattica

Le prime prove sono state effettuate utilizzando un grafo costruito usando il PoS Tagging. Prima di effettuare le prove sono stati selezionati gli archi la cui cosine similarity fosse superiore a 0.75. Questa soglia non è stata scelta in modo casuale ma ragionando prima sull'istogramma visibile nella Figura 7. Questo grafico mette in evidenza qual è il numero di archi con un peso che sta tra i due estremi di ogni intervallo sull'asse delle ascisse. Va ricordato che nell'ultimo intervallo cadono anche tutti quei valori che hanno peso uno, ovvero il risultato della cosine similarity tra una frase e sé stessa. Questo significa che 10000 valori, che è il numero delle frasi del dataset, vanno scartati in quanto non ci interessano loop di un nodo con sé stesso.
La scelta della soglia è stato un compromesso: si è cercato di mantenere un buon numero di nodi e archi e allo stesso tempo di rimuovere i nodi non connessi e gli archi con un peso troppo basso.

Utilizzando questa soglia sono rimasti circa 2,4 milioni di archi e 5012 nodi connessi. Il file dato in input alla GCN conteneva i dati riguardanti solamente questi 5012 nodi. I risultati ottenuti con le percentuali di train menzionate in precedenza e usando il filtro Localpool sono riassunti nella tabella 1. Il numero di epoche impostato per

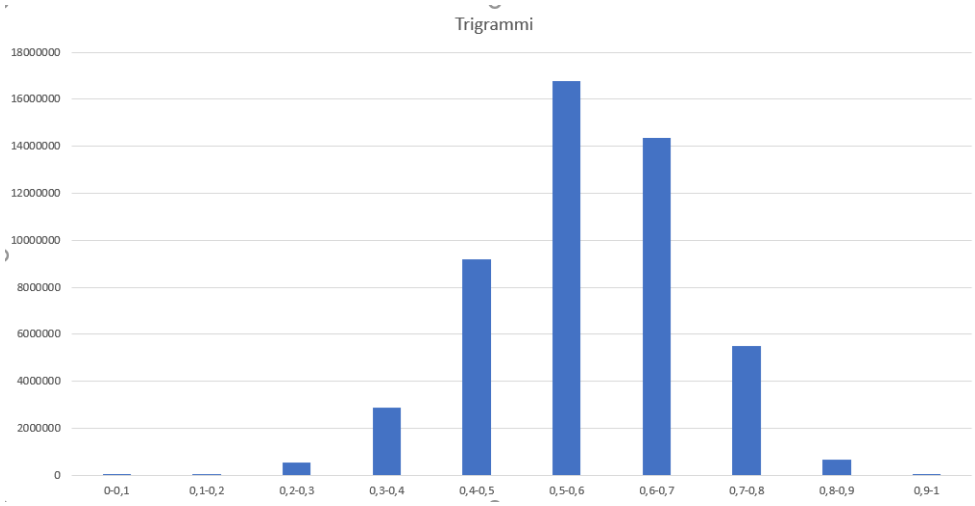

Figura 7: Distribuzione dei pesi considerando trigrammi.

percentuale train	5	10	20	40	70
numero epoche	60	153	20	15	23
accuracy	0.57	0.55	0.53	0.53	0.528

Tabella 1: Risultati PoS Tagging e filtro Localpool.

ogni prova è stato fissato a 300.
Con l'obiettivo di una buona generalizzazione è molto difficile decidere quando è il momento di bloccare il training. C'è il rischio di overfitting se non si ferma l'addestramento al punto giusto. E' stato quindi impostato l'early stopping per bloccare il training nel caso si verificasse overfitting. Il numero di epoche riportato in tabella è quello in cui si è entrati in early stopping. Il tempo di esecuzione per ogni epoca è pari a circa 3 minuti, soprattutto a causa dell'ingente numero di archi.

Utilizzando invece Chebyshev al posto di Localpool i tempi di esecuzione diventano molto più elevati (circa 10 minuti a epoca) ma si ottengono risultati nettamente migliori come si vede in tabella 2.

Una volta ottenuti questi risultati è stato fatto un ulteriore passo avanti. Per cercare di ridurre i tempi di esecuzione e per dare in pasto alla GCN dei dati ancor più

percentuale train	5	10	20	40	70
numero epoche	36	45	42	53	66
accuracy	0.806	0.821	0.838	0.868	0.876

Tabella 2: Risultati PoS Tagging e filtro Chebyshev.

percentuale train	5	10	20	40	70
numero epoche	36	43	32	41	39
accuracy	0.51	0.54	0.57	0.52	0.55

Tabella 3: Risultati PoS Tagging e filtro Localpool

percentuale train	5	10	20	40	70
numero epoche	91	173	251	300	300
accuracy	0.71	0.78	0.843	0.847	0.85

Tabella 4: Risultati PoS Tagging e filtro Chebyshev

significativi, in modo da ottenere un'accuratezza migliore, si è deciso di fare uso della tecnica di "*information gain*". Con questa tecnica ogni nodo viene descritto solamente dalle feature più discriminanti. Usando il filtro Localpool si sono ottenuti i risultati visibili in tabella 3.

Con il filtro Chebyshev ancora una volta si sono allungati i tempi di esecuzione rispetto all'uso di Localpool ma si sono ottenuti risultati migliori visibili nella tabella 4.

4.4 Test con similarità semantica

A questo punto si è pensato di verificare se le accuratezze raggiunte potessero migliorare creando archi su base semantica piuttosto che sintattica. Per questo tipo di legami è stata ottenuta la distribuzione visibile in figura 8.

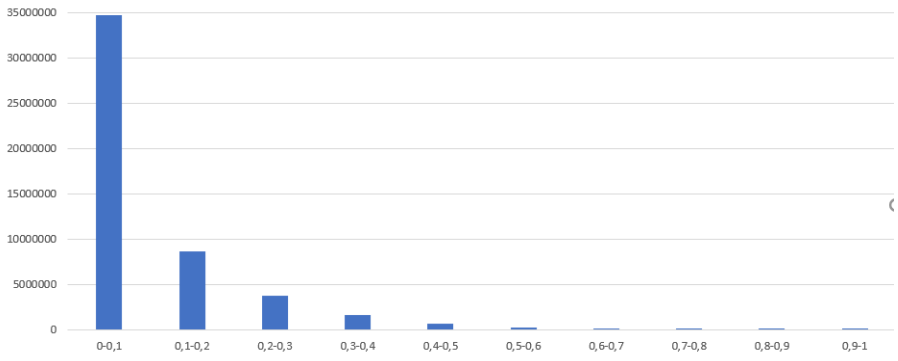

Figura 8: Distribuzione dei pesi considerando monogrammi.

Ancora una volta sono state imposte delle soglie per considerare solo archi con un peso significativo. In particolare sono stati svolti i test con soglie pari a 0.6 e

percentuale train	5	10	20	40	70
numero epoche	200	200	200	200	200
accuracy	0.72	0.755	0.754	0.782	0.754

Tabella 5: Risultati archi semantici e filtro Localpool

percentuale train	5	10	20	40	70
numero epoche	193	271	285	300	205
accuracy	0.738	0.775	0.807	0.844	0.834

Tabella 6: Risultati archi semantici e filtro Chebyshev

0.7. Nel primo caso è stato ottenuto un grafo con circa 195 mila archi e 7058 nodi connessi, nel secondo invece il numero di archi è sceso a circa 70 mila con 4717 nodi connessi. Nonostante l'uso di soglie minori rispetto ai test con PoS tagging, i grafi ottenuti presentavano un ottimo numero di nodi connessi e un buon numero di archi. Questo porta a pensare che utilizzando archi basati sulla semantica della frase sia meglio che crearli usando informazioni sintattiche. I nodi sono stati descritti ancora con le 20 mila feature estratte mediante la tecnica di information gain.
Sono state utilizzate le medesime percentuali per training set, validation set e test set citate in precedenza ed i filtri Localpool e Chebyshev. In tabella 5 si possono vedere i risultati ottenuti utilizzando il filtro Localpool con soglia settata sul valore 0.6.

Utilizzando invece il filtro Chebyshev, sempre con soglia 0.6, sono stati ottenuti i risultati visibili in tabella 6.

Con soglia 0.7 invece i risultati ottenuti con Localpool sono visibili in tabella 7 mentre quelli ottenuti con filtro Chebyshev e soglia 0.7 sono visibili in tabella 8.

4.5 Naive Bayes

Per confrontare i risultati ottenuti con un classificatore classico sono stati svolti dei test usando il classificatore Naive-Bayes, appositamente progettato per la classificazione testuale, sia con i nodi restanti settando la soglia a 0.6 sia con soglia 0.7 ottenendo i risultati in tabella 9 e in tabella 10.

percentuale train	5	10	20	40	70
numero epoche	221	200	200	200	200
accuracy	0.73	0.726	0.768	0.769	0.768

Tabella 7: Risultati archi semantici e filtro Localpool

percentuale train	5	10	20	40	70
numero epoche	130	136	176	176	183
accuracy	0.75	0.78	0.80	0.82	0.83

Tabella 8: Risultati archi semantici e filtro Chebyshev

percentuale train	5	10	20	40	70
accuracy	0.759	0.794	0.845	0.87	0.87

Tabella 9: Risultati Naive-Bayes soglia 0.6

5 Conclusioni

Prima di iniziare questo progetto si è valutata la possibilità di utilizzare le Graph Convolutional Network come possibile classificatore. Per farlo è stato necessario creare un grafo partendo dalle frasi, dove ognuna di queste rappresentava un nodo. È stato utilizzato sempre lo stesso dataset modificato però, come spiegato nei capitoli precedenti, attraverso varie tecniche (dal pre-processing all'information gain e considerando solo gli archi ottenuti imponendo una certa soglia alla cosine similarity).
Gli archi sono stati creati prima su base sintattica. L'accuratezza raggiunta in questo caso differisce di molto al variare del filtro utilizzato: con il filtro Localpool i risultati sono stati scarsi mentre con il filtro Chebyshev si è raggiunta un'accuratezza anche dell'87% con una percentuale del 40% e 70% in training. Una differenza significativa che va evidenziata consiste nel tempo di esecuzione, infatti con Chebyshev la GCN impiega 10-15 minuti a epoca contro i 2-3 minuti impiegati utilizzando il filtro Localpool.
Va detto che i risultati ottenuti mediante similarità sintattica, soprattutto nel caso di Localpool, non sono stati soddisfacenti. Invece i risultati ottenuti con Chebyshev sono buoni, ma i tempi d'esecuzione sono decisamente elevati.
Con archi creati basandosi basandosi sulla similarità semantica tra le frasi i risultati sono nettamente migliorati. In particolare con il filtro Localpool si è passati da un'accuratezza media del 55% ad un'accuratezza media del 75%. Invece con Chebyshev i risultati si sono mantenuti circa sulla stessa accuratezza raggiunta con gli archi costruiti col PoS Tagging.
In questo caso si sono ottenuti risultati soddisfacenti anche usando Localpool mentre con Chebyshev i risultati sono rimasti sullo stesso livello del PoS tagging ma con tempi

percentuale train	5	10	20	40	70
accuracy	0.59	0.68	0.77	0.82	0.87

Tabella 10: Risultati Naive-Bayes soglia 0.7

di computazione nettamente inferiori (grazie al minor numero di archi complessivo).
Come ultimo banco di prova è stato utilizzato il classificatore di Naive-Bayes i cui risultati però sono molto variabili e possono cambiare anche del 10% o 15% ad ogni iterazione, questo perchè i nodi vengono assegnati in modo completamente casuale al classificatore. Nelle tabelle 9 e 10 sono stati riportati i risultati ottenuti calcolando la media di 10 iterazioni.
In conclusione possiamo dire che il modello semantico è migliore rispetto a quello basato su PoS tagging perchè, con un tempo di esecuzione inferiore, si ottiene un'accuratezza uguale o migliore. Il confronto tra GCN e Naive Bayes ci permette di constatare che il classificatore induttivo è migliore nel caso di training set elevato. Con un training set piccolo però la GCN si comporta meglio e questo è quello che volevamo verificare in principio.
Il nostro obiettivo, infatti, consisteva nel verificare se, in un caso reale, in cui si hanno pochi dati classificati, fosse possibile ottenere una accuratezza migliore rispetto ad un approccio induttivo.
Un possibile sviluppo futuro per verificare l'effettiva qualità di questo approccio basato su GCN consiste nel testarlo con un problema avente più classi (Es: emotion detection) per avere un grafo che si scompone meglio e in cui si creano un maggior numero di comunità.
In una situazione con molte classi un classificatore come Naive Bayes fa molta fatica rispetto ad un caso binario (label positiva o negativa) e quindi la demarcazione nei risultati potrebbe essere ancora più ampia.

Riferimenti bibliografici

[1] P. V. Michaël Defferrard, Xavier Bresson. *Convolutional Neural Networks on Graphswith Fast Localized Spectral Filtering*. https://arxiv.org/pdf/1606.09375.pdf, 2016.

[2] M. W. Thomas Kipf. *Graph Convolutional Networks*. http://tkipf.github.io/graph-convolutional-networks/, Sept. 2016.

[3] M. W. Thomas Kipf. *Semi-Supervised Classification with Graph Convolutional Networks*. https://arxiv.org/pdf/1609.02907.pdf, 2017.