UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTO DI SISTEMI DISTRIBUITI

DEVELOPMENT OF A MULTIPROCESS TOOL FOR TEXTUAL CLASSIFICATION WITH GCN

SVILUPPO DI UN TOOL MULTIPROCESSO PER LA CLASSIFICAZIONE TESTUALE MEDIANTE GCN

Docente: Agostino Poggi

Correlatore: Gianfranco Lombardo

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

ANNO ACCADEMICO 2018-2019

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 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 nel creare un tool multiprocesso per generare i file di input per una Graph Convolutional Network (GCN) e verificarne il funzionamento con dataset a più classi.
Le operazioni che sono state parallelizzate sono state tre:

- Preprocessing del dataset;
- Calcolo della cosine similarity;
- Calcolo dell'information gain.

Queste operazioni verranno spiegate nel dettaglio nei prossimi capitoli.
Il progetto aveva un duplice obiettivo. In primis verificare che parallelizzando le operazioni su più processi la creazione dei file di input per la GCN fosse più rapida rispetto ad un'esecuzione seriale. Con approccio seriale si fa riferimento al lancio delle varie funzioni una dopo l'altra: tutte sono eseguite dal medesimo thread dello stesso processo. Il secondo obiettivo consisteva nel verificare la bontà dei risultati restituiti dalla

GCN con dataset a più classi confontandone il risultato con un classificatore tradizionale soprattutto nel caso di un training set piccolo. Nei prossimi capitoli saranno spiegate nel dettaglio le scelte fatte e i risultati ottenuti.

Capitolo 2

GCN: cosa sono e come funzionano

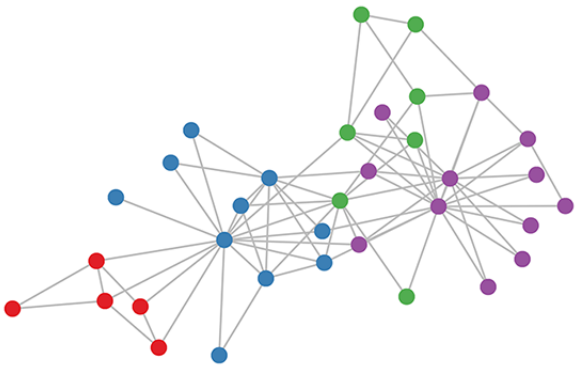

Figura 2.1: Schema di un grafo

Per capire le operazioni che sono state svolte è necessaria un'introduzione sulla GCN per approfondirne il funzionamento. 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.

Una GCN può essere definita dalla seguente equazione:

$$(X) = ReLU(AXW); \tag{2.1}$$

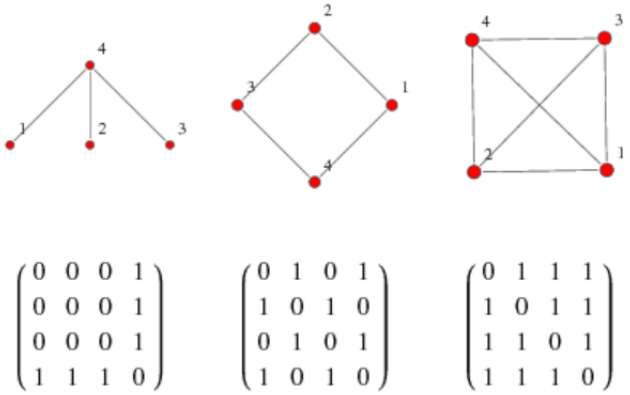

Figura 2.2: Esempi di matrici di adiacenza.

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:

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:

Per i nostri test sono stati usati due filtri che approfondiamo nei due paragrafi successivi.

2.1 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 chia-

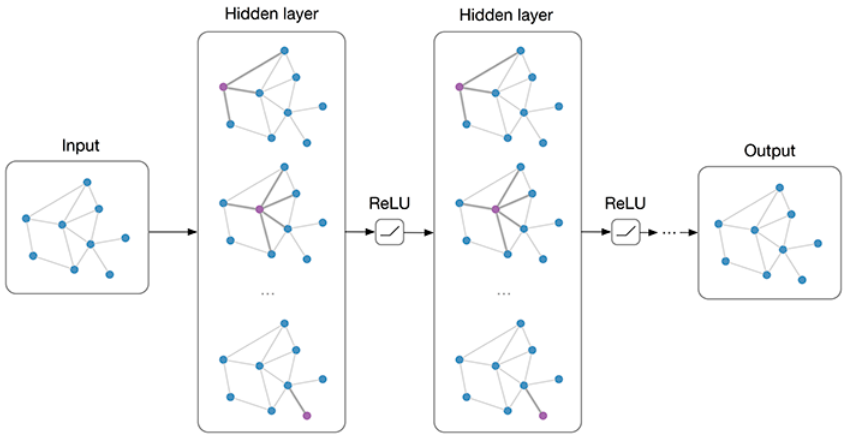

Figura 2.3: Schema di una GCN

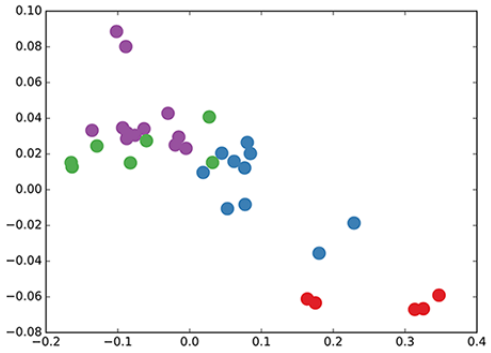

Figura 2.4: Embedding dei nodi del grafo con una GCN non allenata

mato *"Renormalization Trick"*.
Una possibile equazione che descrive la propagazione delle informazioni nella GCN è la seguente:

$$f(H^{(l)},A) = \sigma(AH^{(l)}W^{(l)}); \tag{2.2}$$

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:

$$f(H^{(l)},A) = \sigma(\widehat{D}^{\frac{-1}{2}}\widehat{A}\widehat{D}^{\frac{-1}{2}}H^{(l)}W^{(l)}); \tag{2.3}$$

dove

$$\widehat{A} = A + I \tag{2.4}$$

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.

2.2 Filtro di Chebyshev

In matematica, i polinomi di Čebyšëv, o Chebyshev, sono le componenti di una successione polinomiale che inizia con i seguenti polinomi:
$T_0(x) = 1$
$T_1(x) = x$
$T_2(x) = 2x^2\text{-}1$
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.

Capitolo 3

Introduzione al Multiprocessing in Python

Python è uno dei più popolari linguaggi per l'elaborazione e l'analisi dei dati. L'ecosistema fornisce molte librerie e framework che facilitano i calcoli ad alte prestazioni. Il parallelismo può essere raggiunto seguendo due strade: multithreading e multiprocessing. Le differenze tra i due approcci sono molteplici. Il multithreading non è adatto

PROCESSI	THREAD
I processi non condividono la memoria	I thread condividono la memoria
I processi avviati/cambiati sono costosi	I thread avviati/cambiati sono meno costosi
I processi richiedono molte risorse	I thread richiedono poche risorse(qualche volta sono chiamati processi leggeri)
Non occorre sincronizzazione di memoria	Avete bisogno di usare meccanismi di sincronizzazione per essere sicuri che stiate gestendo correttamente i dati

Figura 3.1: Confronto tra multiprocessing e multithread.

al nostro scopo a causa del lock globale dell'interprete (GIL). In CPython, l'implementazione più popolare di Python, il GIL è un mutex che crea cose thread-safe. Il GIL lo rende semplice da integrare con le librerie esterne che non sono thread-safe e rende il codice non-parallelo più veloce. Tuttavia, ciò ha un costo. A causa di GIL, non è possibile realizzare il parallelismo vero tramite il multithreading. In sostanza, due thread nativi diversi dello stesso processo non possono eseguire il codice Python subito. Utilizzando i processi vengono avviati un certo numero di interpreti Python in parallelo, ognuno dei quali ha uno spazio di memoria privato con il proprio GIL e

ognuno di essi viene eseguito in serie. Questo è il modo più semplice per accelerare un compito a elevato carico per il processore. Per il nostro obiettivo è stata utilizzata la libreria *multiprocessing* che consente di raggiungere il vero parallelismo. La libreria offre un oggetto *Pool* che accetta come argomento il numero dei processi da avviare.
Quando viene lanciato il tool richiede all'utente quanti core del processore vuole utilizzare. Il numero massimo dipende chiaramente dal computer in uso.
Come accennato nell'introduzione le operazioni che sono state suddivise sui più core sono le seguenti tre.

3.1 Preprocessing del dataset - Multiprocesso

Al dataset sono state applicate alcune tecniche di preprocessing. Sono state eliminate 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).
Queste operazioni di preprocessing sono state eseguite lanciando più processi. Il dataset viene suddiviso in tante parti quanti sono i processi che si vogliono lanciare e ad ogni processo è stata fornita una porzione dello stesso in modo da suddividere equamente il carico di lavoro.

3.2 Generazione del grafo e similarità semantica

Il secondo 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 collegare le frasi mediante archi ci si è basati sulla 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.
Per ridurre il numero di features è stato utilizzato un metodo chiamato *information gain*.

3.3 Information Gain - Multiprocesso

Grazie a questo metodo si possono selezionare le features più discriminanti che definiscono la positività o la negatività di una frase. La funzione *mutualinfo* della libreria

sklearn 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. Una volta ottenuto questo vettore, sono state cancellate dalla matrice tutte quelle colonne (feature) il cui valore di mutua informazione è risultato essere molto basso e quindi poco discriminante. Per rendere la funzione multiprocesso è stato necessario dividere la matrice di partenza in tante sottomatrici quanti sono i processi che si vogliono lanciare. Ogni processo ha gestito una parte della computazione e al termine i risultati sono stati riuniti in un unico vettore di output.

3.4 Cosine similarity - Multiprocesso

Per dare un peso agli archi che collegano i vari nodi è stata calcolata la cosine similarity tra ogni feature e tutte le altre. Questa funzione genera una matrice NxN simmetrica. Anche in questo caso è stata utilizzata una soglia per mantenere solo gli archi aventi un peso significativo. 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.

3.5 Il tool

Al termine delle operazioni sopra descritte 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.

Capitolo 4

Prove e risultati ottenuti

In prima istanza sono stati effettuati dei test per verificare che il tool multiprocess garantisse un buon risparmio di tempo rispetto all'esecuzione sequenziale. I risultati di questi test sono riportati nel paragrafo successivo.
Per lanciare la GCN sono stati generati due file: uno contenente informazioni sui nodi e uno sugli archi. Il file dei nodi contiene, per ogni riga, l'ID del nodo, il valore del Tf-IDF e la label del nodo mentre il file degli archi contiene per ogni riga le coppie dei nodi e il peso dell'arco che li collega.
I test sono stati effettuati su due dataset differenti: uno contenente delle notizie e uno di sentimental analysis. Entrambi i dataset sono stati pre-processati così da ottenere un file formattato nel modo corretto per essere dato in input al tool multiprocess precedentemente descritto, inoltre non è garantito che questi siano bilanciati (numero diverso di istanze per ogni classe).
Il dataset delle news inizialmente conteneva notizie riguardanti dodici diverse classi: politica, sport, intrattenimento, business, food & drink, ecc. Per i nostri test sono state usate solo cinque di queste classi, in modo da avere circa 32000 frasi totali. La riduzione del dataset è dovuta al fatto che per elaborare un numero così elevato di frasi il tool multiprocess utilizza un quantitativo enorme di memoria in quanto la dimensione delle matrici generate per il calcolo della cosine similarity e dell'information gain esplode molto rapidamente.
Il secondo dataset è costituito da 13 classi, per un totale di 30000 frasi, non è quindi stato ridurlo.
I test sono stati effettuati utilizzando diverse percentuali per il training set: 5, 10, 30, 50 e 70%. Per quanto riguarda la misurazione dell'accuratezza dei risultati raggiunti è stata utilizzata la funzione f1-score che permette di valutare l'accuratezza nel caso di un dataset sbilanciato. Questa misura tiene in considerazione precisione e recupero del test: la precisione è il numero di veri positivi diviso il numero di tutti i risultati positivi, mentre il recupero è il numero di veri positivi diviso il numero di tutti i test che

numero thread	Dataset sentimental analysis	Dataset news
1	10280 s	12189 s
8	2796 s	3705 s
32	1437 s	1843 s

Tabella 4.1: Risultati dei test sul tempo di esecuzione del tool in secondi

sarebbero dovuti risultare positivi (ovvero veri positivi più falsi negativi). L'equazione che viene utilizzata da questa funzione è la seguente, rappresenta la media armonica di precisione e recupero:

$$F1 = 2\frac{p*r}{p+r}$$

La funzione implementata nella libreria *scikit learn* di Python fornisce diverse opzioni per calcolare questa media: i test sono stati eseguiti sia con f1-score micro che con f1-score macro.
Nelle sezioni sottostanti si presentano i risultati raggiunti sui due dataset considerati usando sia i filtri Localpool e Chebyshev propri della GCN, sia il classificatore Naive-Bayes, usato come metro di paragone. I risultati di quest'ultimo 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 sono stati riportati i risultati ottenuti calcolando la media di 20 iterazioni.

4.1 Test sul tempo di esecuzione del tool multiprocess

Sono stati eseguiti dei test per verificare l'effettivo risparmio di tempo nella generazione dei file di input per la GCN usando il tool multiprocess. I test sono stati effettuati con 1, 8 e 32 (massimo numero di processi eseguibili in parallelo sul cluster in uso) processi in esecuzione in parallelo. Questi risultati sono ovviamente legati a diverse variabili: dipendono, ad esempio, dalla dimensione del dataset e dalla potenza computazionale della macchina su cui viene eseguito il tool.
Dalla Tabella 4.1 si può notare come effettivamente ci sia un evidente risparmio di tempo nell'esecuzione con 8 e 32 processi rispetto all'esecuzione sequenziale.

4.2 Test dataset news

In questo paragrafo illustreremo i risultati che sono stati ottenuti operando sul dataset delle news.
Si può notare dalle tabelle, una relativa al filtro Localpool (Tabella 4.2), e una relativa

bim	micro	macro
5	0,767	0,763
10	0,795	0,791
30	0,807	0,803
50	0,815	0,813
70	0,818	0,814

Tabella 4.2: Risultati ottenuti sul dataset news usando il filtro Localpool

bim	micro	macro
5	0,760	0,756
10	0,773	0,770
30	0,793	0,788
50	0,809	0,805
70	0,798	0,795

Tabella 4.3: Risultati ottenuti sul dataset news usando il filtro di Chebyshev

al filtro di Chebyshev (Tabella 4.3), che la differenza tra questi due filtri (dal punto di vista dei risultati) è rilevante ma non eccessiva. Il filtro Localpool si comporta meglio rispetto al filtro di Chebyshev ed è anche più veloce. Mediamente un'epoca del filtro Localpool viene calcolata in 3 secondi mentre con Chebyshev in 6 secondi.
Questo risultato è interessante perchè nel caso di un dataset con due sole classi, come quello considerato in [1], i risultati sono stati decisamente migliori con filtro di Chebyshev.

Con il classificatore di Naive-Bayes (vedi Tabella 4.4) sono stati confermati i risultati ottenuti in [1]. Con percentuali di training molto basse (5 e 10%) la GCN si è comportata decisamente meglio, arrivando a ottenere un'accuratezza migliore anche del 10%, sia su micro che su macro, considerando il paragone tra il filtro Localpool (il migliore tra i due della GCN) e Naive-Bayes.

Nella Figura 4.1 si può osservare un grafico riassuntivo dei risultati raggiunti coi

bim	micro	macro
5	0,686	0,670
10	0,767	0,774
30	0,820	0,818
50	0,838	0,834
70	0,845	0,842

Tabella 4.4: Risultati ottenuti sul dataset news utilizzando il classificatore Naive-Bayes

due filtri della GCN e con Naive-Bayes utilizzando il parametro micro per il calcolo della media.

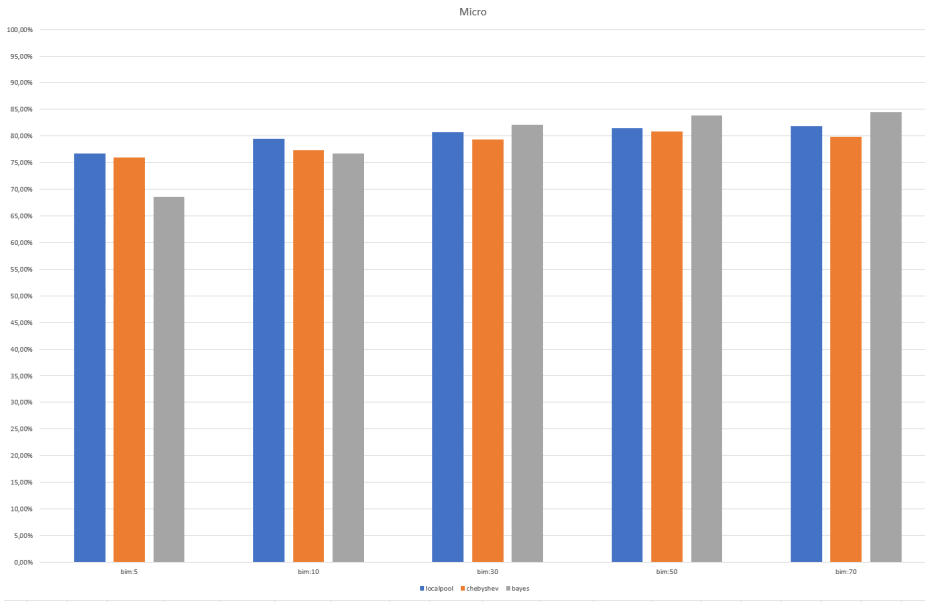

Figura 4.1: Plot delle percentuali raggiunte con micro - Dataset news

Infine nella Figura 4.2 è stato riportato lo stesso tipo di grafico solo considerando i risultati raggiunti coi due filtri della GCN e con Naive-Bayes utilizzando il parametro macro per il calcolo della media.

4.3 Test dataset sentimental analysis

I test effettuati su questo dataset hanno evidenziato una maggiore difficoltà da parte della GCN a raggiungere buone accuratezze. Inoltre in questo caso, anche con percentuali di training molto basse, il classificatore Naive-Bayes (vedi Tabella 4.7) ha sempre raggiunto migliori accuratezze rispetto ai filtri utilizzati nella GCN (vedi Tabelle 4.5 e 4.6).

Da notare inoltre che l'accuratezza raggiunta dalla GCN con entrambi i filtri, sia con l'opzione micro che con macro, è pressoché la stessa per tutte le percentuali di training. I risultati sono riportati nelle Tabelle 4.5 e 4.6.

I risultati ottenuti con questo dataset sono scarsi sia usando il classificatore Naive-Bayes sia con i due filtri Localpool e Chebyshev. Questo risultato può essere dovuto principalmente a due fattori:

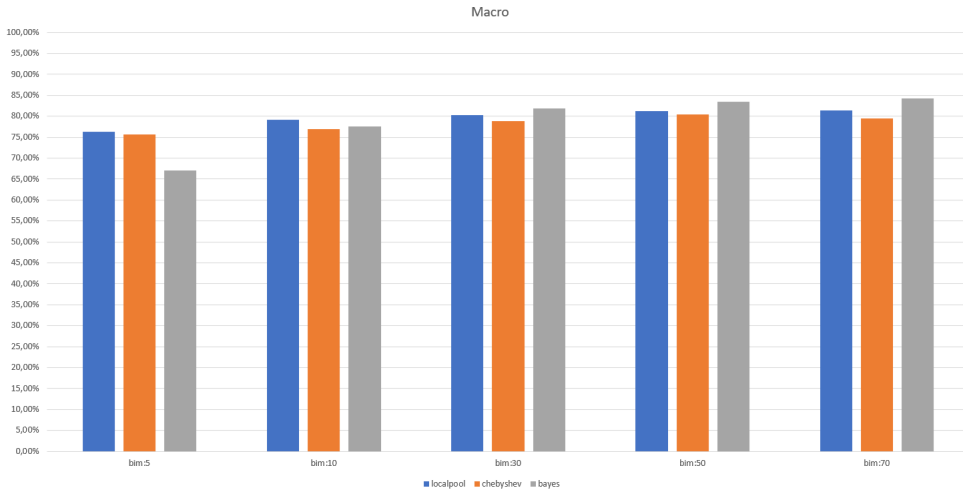

Figura 4.2: Plot delle percentuali raggiunte con macro - Dataset news

bim	micro	macro
5	0.251	0.098
10	0.270	0.094
30	0.275	0.097
50	0.283	0.106
70	0.283	0.094

Tabella 4.5: Risultati ottenuti sul dataset di sentimental analysis usando il filtro Localpool

- il numero di classi, nettamente maggiore rispetto al caso precedente, il che rende più difficile il lavoro della GCN;
- le caratteristiche intrinseche del dataset. Le frasi sono infatti molto brevi perchè derivano da post pubblicati sui social network e includono link, tag, abbreviazioni etc che non aiutano nella classificazione di una frase.

Nella figura 4.3 è mostrato un confronto dei risultati relativi ottenuti con l'opzione micro usando il dataset di sentimental analysis con i filtri Localpool, Chebyshev e con il classificatore Naive-Bayes. Nella figura 4.4 è possibile osservare i medesimi risultati relativi però al caso f1-score macro. O

bim	micro	macro
5	0.256	0.098
10	0.260	0.099
30	0.281	0.109
50	0.276	0.101
70	0.290	0.111

Tabella 4.6: Risultati ottenuti sul dataset di sentimental analysis usando il filtro di Chebyshev

bim	micro	macro
5	0.378	0.168
10	0.384	0.193
30	0.396	0.204
50	0.405	0.216
70	0.416	0.223

Tabella 4.7: Risultati ottenuti sul dataset di sentimental analysis usando il classificatore Naive-Bayes

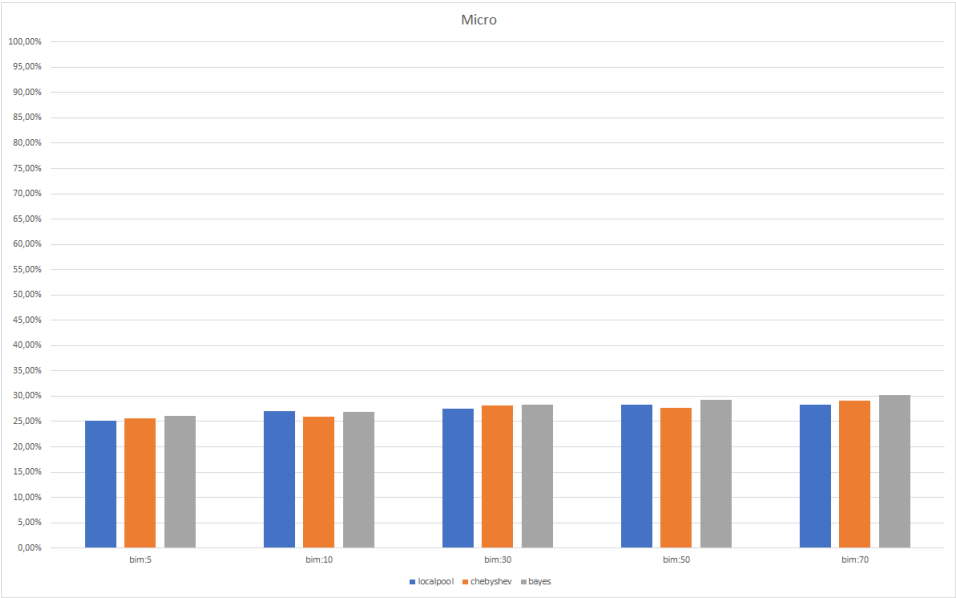

Figura 4.3: Plot delle percentuali ottenute con micro - Dataset sentimental analysis

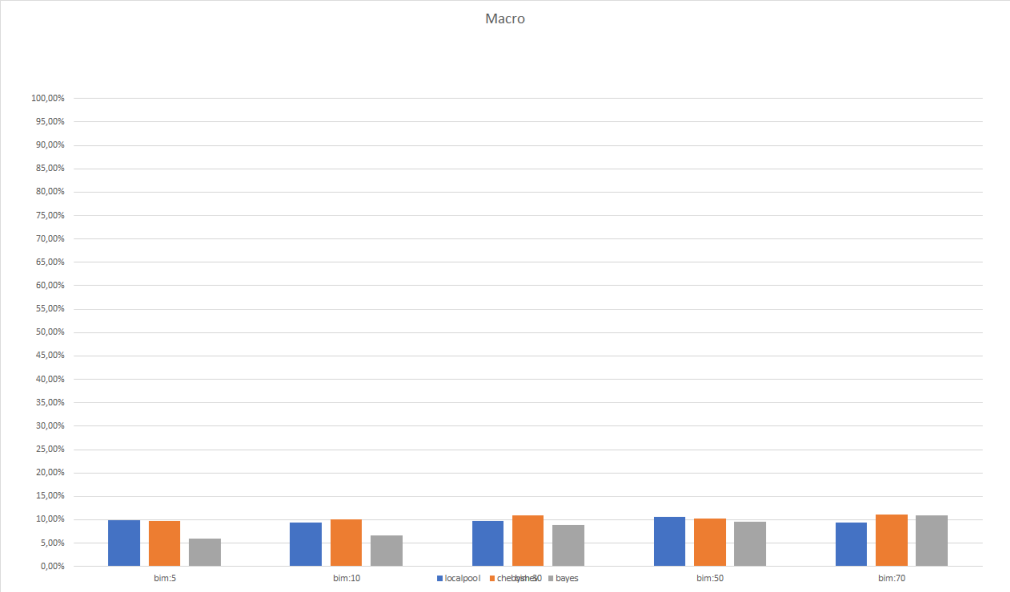

Figura 4.4: Plot delle percentuali ottenute con macro - Dataset sentimental analysis

Capitolo 5

Conclusioni

Il primo obiettivo che si voleva raggiungere con questo progetto consisteva nel rendere multiprocesso e dinamico il tool che permette di creare i file di input per la GCN. Come mostrano i risultati illustrati nel capitolo precedente, i vantaggi che derivano dall'uso di più processi in parallelo sono evidenti. Il risparmio in termini di tempo è netto: con 32 processi i file vengono generati in circa un decimo del tempo rispetto all'esecuzione sequenziale (monoprocesso). Anche usando solo 8 core, presenti sulla stragrande maggioranza dei portatili moderni, il risparmio in termini di tempo è evidente: il tool impiega circa un quarto del tempo rispetto all'esecuzione sequenziale (in questo caso però c'è da tenere in considerazione il fattore relativo alla memoria necessaria per memorizzare le matrici di cosine similarity e information gain).
Per quanto riguarda i test con la Graph Convolutional Network è stato necessario, innanzitutto, creare un grafo partendo dalle frasi, dove ognuna di queste rappresentava un nodo. I risultati ottenuti con il dataset delle notizie sono stati soddisfacenti. La GCN, nel caso d'interesse, ovvero con un training set piccolo, è riuscita a ottenere accuratezze nettamente migliori rispetto a Naive Bayes. Ad esempio con un training set del 5% Naive-Bayes restituisce un'accuratezza del 67% (f1-score macro) contro il 76% della GCN (con filtro Localpool).
I risultati ottenuti con il dataset di sentiment analysis non sono stati soddisfacenti in termini assoluti anche se la GCN non sfigura nel confronto: l'accuratezza raggiunta da Naive-Bayes è solo leggermente migliore.
Sicuramente le accuratezze raggiunte sono migliorabili. Con un tuning accurato delle soglie di cosine similarity e information gain e ulteriori test si potrebbero ottenere risultati ancora migliori. Va sottolineato che, in generale, non è affatto facile definire delle relazioni, delle similarità tra le frasi, si tratta di un open problem di language processing.

Bibliografia

[1] D. Z. Matteo Gatti. *TEXTUAL CLASSIFICATION WITH GCN*, 2019.

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

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

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