POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA
E DELL'INFORMAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Deep Learning Project Assignment Report and Model Analysis

Political Blog Classification using Graph Convolutional Networks

Docente del corso:
Vito Walter Anelli, Ph.D.

Studenti:
Apollo Beatrice, Cassano Mario

Anno Accademico 2023/2024

Sommario

Sezione 1

Obiettivo del progetto

L'obiettivo del seguente progetto è quello di costruire un modello basato su GCN (Graph Convolutional Network) al fine di eseguire un task di classificazione dei nodi di un grafo, a partire da un dataset fornito. In particolare, il modello dovrà classificare politicamente alcuni blog come "liberali" o "conservatori".

Sezione 2

Il dataset

Il dataset in questione è il "Polblog dataset". Esso è costituito da una rete di collegamenti tra blog riguardanti la politica americana, registrato nel 2005 da Adamic e Glance [1].

2.1 Struttura del dataset

Il dataset contiene 1490 nodi e 19025 archi, con i rispettivi attributi. I nodi rappresentano i blog e gli archi i collegamenti tra loro.

- Attributi dei nodi: "id" (identificativo dei blog), "label" (link dei blog), "value" (può assumere valore 0 o 1, indica se il blog è classificato come liberale o conservativo) e "source" (indica le fonti dalle quali è stato reperito il blog);
- Attributi degli archi: "source" (nodo di partenza) e "target" (nodo di arrivo).

Sezione 3

Preprocessing

Prima di realizzare il modello, sono state effettuate varie operazioni di preprocessing sul dataset di partenza, al fine di adattarlo all'architettura da utilizzare e al task di classificazione.

3.1 Caricamento del dataset

Per leggere il file in formato ".gml" e analizzarne la struttura (Numero di nodi, numero di archi, tipologia degli attributi), è stata utilizzata la libreria "igraph". Successivamente, i nodi e gli archi sono stati salvati come DataFrame Pandas, utilizzando i metodi "get_vertex_dataframe" e "get_edges_dataframe".

3.2 Rimozione dei duplicati

Per rimuovere eventuali righe duplicate nei DataFrame è stato adoperato il metodo "drop" di Pandas.

3.3 Recupero dei testi dei blog (Web Scraping)

Attraverso l'attributo "label" dei nodi, è stato possibile ottenere una lista dei link di tutti i blog presenti all'interno del dataset. Questa lista è stata utilizzata per effettuare richieste HTTP agli elementi delle pagine web, al fine di recuperare i testi principali presenti all'interno dei blog.

3.4 Trasformazione dei testi in embedding

I testi ottenuti sono stati trasformati prima in token, utilizzando il modello BERT Tokenizer pre-addestrato, e poi in embedding, utilizzando il modello BERT base (anch'esso pre-addestrato).

- Ogni token è rappresentato da un embedding, ovvero un tensore di 768 numeri reali;
- Ogni testo è costituito da un numero massimo di 512 token (in alternativa, viene utilizzato uno specifico padding chiamato "attention_mask");
- Infine, il significato del testo è rappresentato dalla media degli embedding di tutti i token che lo compongono.

3.5 Gestione della grande quantità di dati

Per evitare di dover effettuare un download di tutti i testi per ogni modifica del codice, gli embedding di tutti i nodi sono stati salvati in un file ".csv" locale. Il programma ne rileva l'eventuale presenza e lo utilizza per il successivo training.

3.6 Gestione degli attributi dei nodi

Per ognuno degli attributi dei nodi, sono state effettuate scelte differenti. L'attributo "label" è stato escluso dall'addestramento della rete, poiché non ha rilevanza nella classificazione. L'attributo "source", invece, può assumere più valori per la stessa istanza. É stato deciso di trasformarlo in un attributo categorico multiclasse, effettuandone l'encoding attraverso il metodo "MultiLabelBinarizer" della libreria sklearn.

3.7 Aggiunta degli embedding come feature dei nodi

Gli embedding ottenuti precedentemente sono stati poi aggiunti al DataFrame dei nodi, per essere utilizzati come feature aggiuntive dei blog. La dimensione finale del DataFrame dei nodi risulta quindi essere [1490, 777].

3.8 Creazione di X e y

Sono state separate per ogni nodo le feature di input (X) dalla feature target "value" (y).

3.9 Creazione di una struttura a grafo

I dati, sotto forma di tensori, sono stati trasformati in una struttura a grafo, utilizzando l'oggetto "Data" della libreria "torch_geometric". Gli attributi principali dell'oggetto sono:

- "x": La matrice delle feature dei nodi, con dimensione [num_nodi, num_feature_nodi]. A questo attributo è stato assegnato il valore X, che corrisponde al tensore le cui righe rappresentano il numero di nodi (1490) e le colonne il numero di feature che sono state prese in considerazione (777).
- "edge_index": La matrice di adiacenza, che rappresenta la connettività del grafo in formato di coordinate, con dimensione [2, num_edges]. A questo attributo è stato assegnato il valore di "edges", che nel codice rappresenta la trasposta della matrice dei collegamenti tra i nodi.
- "y": Le etichette ("value") dei nodi.

3.10 Divisione in training set, validation set e test set

Successivamente, vengono selezionati randomicamente 900 nodi da utilizzare per il training, il cui 20% viene utilizzato per la validazione. I restanti 590 nodi vengono utilizzati

esclusivamente per il test. Questa divisione è stata effettuata sfruttando gli attributi "train_mask", "val_mask" e "test_mask" dell'oggetto "Data". In particolare, questi attributi richiedono come valori dei tensori booleani, in cui gli indici che hanno valore True sono i nodi da considerare rispettivamente per il training, la validazione e il test.

Sezione 4

Architettura del modello

4.1 Rete GAT

Per il task di classificazione dei nodi è stata implementata una GAT (Graph Attention Network) multi-layer. Una GAT è una tipologia di rete neurale convoluzionale che prende in input un grafo ed effettua operazioni di "message passing" attraverso la convoluzione. La differenza principale tra questo tipo di rete e una semplice GCN (Graph Convolutional Network) è che la GAT sfrutta il calcolo dei coefficienti di attenzione per stabilire l'importanza di un nodo rispetto ad un altro. In particolare, ogni layer riceve in input le feature dei nodi e restituisce in output un set di feature più utile per la classificazione finale. Sia all'interno del GAT layer che tra un layer e l'altro è stato introdotto il dropout. Esso permette, nella fase di training, di eliminare con una certa probabilità (in questo caso pari a 0.2) parte dei coefficienti di attenzione o delle feature considerate. Questa tecnica ha lo scopo di prevenire l'overfitting.

L'architettura realizzata prevede 3 layer di convoluzione con funzione di attivazione ReLU. L'ultimo layer, invece, restituisce le probabilità che il blog sia "liberale" o "conservatore", attraverso l'utilizzo della funzione "log_softmax". Non è stata utilizzata una rete di classificazione fully-connected alla fine, poiché i risultati ottenuti da questa strategia si sono rivelati già soddisfacenti con questa scelta.

4.2 Transformer

Anche se non esplicitamente richiesto dalla traccia, è stato realizzato un modello transformer in un file a parte ("transformer.py") per verificare con quanta accuratezza fosse possibile classificare i blog utilizzando come feature solamente gli embedding dei testi, oltre che la variabile target. A questo scopo, è stato utilizzato il modello pre-addestrato "BertForSequenceClassification" della libreria "HuggingFace". Data la scarsa qualità delle informazioni del dataset (moltissimi link non funzionanti, funzione di web scraping che

rileva tutto il testo nei , spesso poco utile, e dimensione massima di ogni testo pari a 512 token) il risultato ottenuto, dopo la validazione di alcuni parametri, è un'accuratezza di test pari a circa il 60%. Con questo esperimento è possibile confermare che le sole informazioni contenute nel testo (con il dataset in questione e l'algoritmo di web scraping realizzato) non sono abbastanza rilevanti per stabilire se un blog sia liberale o conservatore. Per raggiungere un buon risultato di accuratezza, sono infatti fondamentali le altre feature dei nodi, ma soprattutto la matrice di adiacenza. Nel codice principale il modello BERT viene quindi utilizzato solo per ottenere gli embedding dei testi, da affiancare alle altre feature.

Sezione 5

Training e validazione del modello

Una volta implementata l'architettura del modello, è stato necessario trovare la combinazione di iperparametri in grado di restituire la migliore accuratezza possibile sul set di validazione. In particolare, è stata implementata una grid search from scratch che considera i seguenti iperparametri:

- Hidden_dim: la dimensione del set di feature in output da ogni layer. Può assumere valore 4, 8, oppure 16;
- Heads: il numero di teste di attenzione parallele. Può assumere come valore 8, 12, oppure 16;
- Learning rate/Learning rate iniziale (per gli ottimizzatori adattivi): può assumere valore 0.001, oppure 0.01;
- Optimizer: il tipo di ottimizzatore da utilizzare. Sono stati considerati Adam, RMSProp e Adadelta.

Una volta istanziati tutti i modelli possibili, ottenuti attraverso le varie combinazioni, essi sono stati allenati per 200 epoche calcolando l'accuratezza sul validation set. In questo caso, il modello migliore è risultato essere quello composto dalla seguente combinazione di iperparametri:

- Hidden_dim: 4;

- LR (iniziale): 0.001;
- Heads: 12;
- Optimizer: Adam.

Questo modello ha dato come risultato un'accuratezza sul validation set pari al 96%.

Sezione 6

Test del modello

Il modello selezionato è stato testato sul test set, raggiungendo un'accuratezza del 94,9% (554 nodi classificati correttamente su 590) ed una test loss pari a 0.22. È doveroso precisare che questi risultati sono stati ottenuti su questo particolare test set (definito mediante permutazioni casuali) e che, a fronte di significativi cambiamenti di quest'ultimo, i risultati potrebbero differire con quelli ottenuti.

L'andamento della loss del miglior modello ottenuto raggiunge valori molto bassi già a metà addestramento (100 epoche), come mostrato in Figura 1.

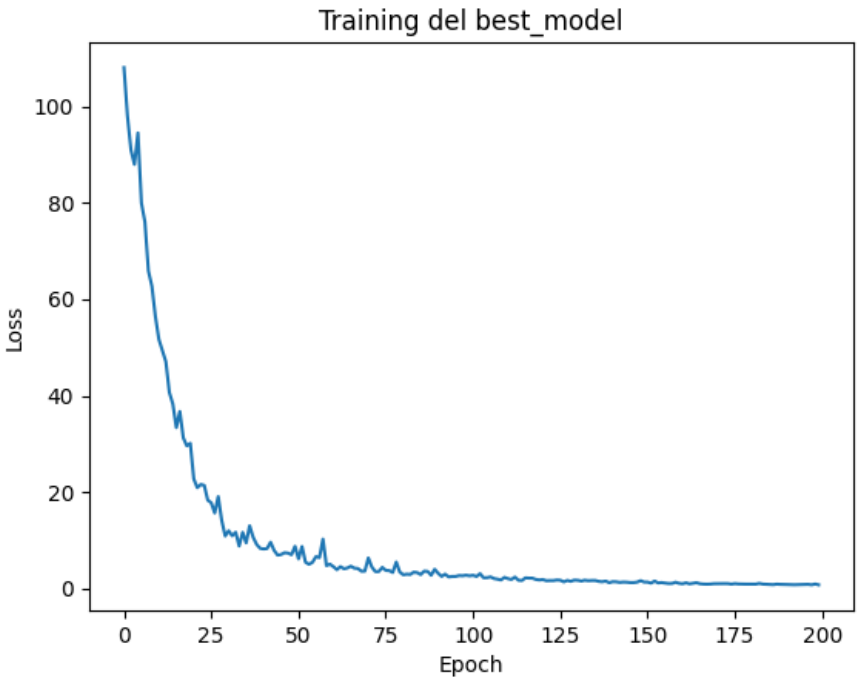

Figura 1 - Training del miglior modello

Sezione 7

Conclusioni e sviluppi futuri

7.1 Analisi delle prestazioni del modello

La principale metrica utilizzata per valutare le prestazioni del modello è l'accuratezza, calcolata come la percentuale di nodi classificati correttamente sul numero totale di nodi. Per il task in oggetto, infatti, non c'è una priorità di classificazione corretta rispetto ad una delle due classi, in quanto hanno entrambe la stessa importanza. È possibile affermare che i risultati ottenuti sono del tutto soddisfacenti; tuttavia, come dimostrato dall'esperimento effettuato con il modello "BertForSequenceClassification", alcuni elementi del dataset sono stati più utili di altri per il task di classificazione. In particolare, il testo dei blog si è rivelato poco utile, mentre le feature iniziali e i collegamenti tra i nodi sono stati determinanti per il risultato ottenuto. Infatti, l'accuratezza sul test set, senza gli embedding dei testi, è pari all'86%. Anche i parametri utilizzati sono stati determinanti per i risultati ottenuti: in particolare, la gestione del learning rate e del numero di epoche, rivelano che tecniche come Adadelta hanno bisogno di più epoche per raggiungere un livello adeguatamente basso di loss. Inoltre, come mostra la Figura 2, il suo andamento nel tempo è fortemente instabile.

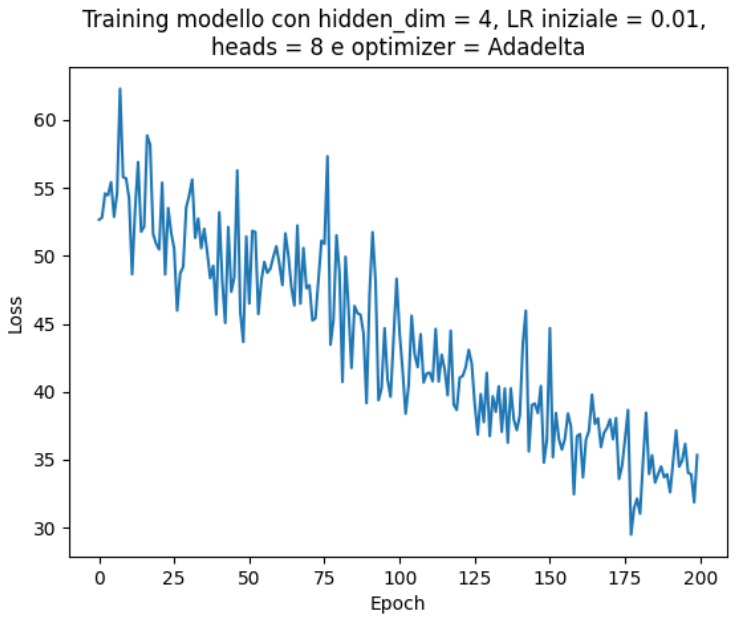

Figura 2 - Andamento della loss in funzione delle epoche utilizzando Adadelta

7.2 Difficoltà riscontrate

I problemi riscontrati riguardano principalmente il dataset, in quanto la maggior parte dei link dei blog si è rivelato non funzionante o obsoleto (solo 500 download su 1490 sono andati a buon fine). Questo ha portato a dover considerare, per la maggior parte dei nodi, l'embedding della frase "Testo non disponibile!". La conseguenza di ciò è che anche la prestazione del modello ne ha risentito. Inoltre, per alcuni link funzionanti, i testi dei blog non erano disponibili; dunque, fornivano un testo senza informazioni rilevanti per la classificazione.

7.3 Sviluppi futuri

Tra i possibili miglioramenti possiamo elencare:

1. Un algoritmo di web scraping più sofisticato, che permetta di analizzare una maggiore quantità e qualità (in termini di rilevanza per la classificazione) di testo utile;

2. Una generazione degli embedding più accurata (ad esempio, utilizzando il modello BERT nella versione Large);

3. Una validazione del modello che comprenda anche le epoche utilizzate durante l'addestramento e la probabilità di innescare il dropout, differenziando quello che riguarda i coefficienti di attenzione da quello che riguarda le feature dei nodi.

Bibliografia

[1] L. A. Adamic and N. Glance, "*The political blogosphere and the 2004 US Election*", in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005).