UNIVERSITÀ DI CATANIA

Dipartimento di Matematica e Informatica

Laurea Triennale in Informatica

Kevin Speranza

GraphSAGE applicato al dataset MovieLens 100k

Progetto Big Data

Professore: Alfredo Pulvirenti

Academic Year 2024 - 2025

Contents

1 Introduzione **2**

2 Dataset **3**
2.1 MovieLens Dataset 3
2.2 Costruzione della Rete Bipartita 3
2.3 Engineering degli Attributi dei Nodi 4
2.4 Setup training 5

3 Implementazione **6**
3.1 Intro GraphSAGE 6
3.2 Architettura implementata 7
3.3 Esperimenti 8
3.3.1 Setup 8
3.3.2 Generazione embedding 8
3.3.3 Predizione rating 11
3.4 Ulteriori valutazioni 13
3.4.1 Proiezioni t-SNE 13
3.4.2 Similarità film - film 14

4 Conclusioni **15**

Chapter 1

Introduzione

I sistemi di raccomandazione sono oggi fondamentali in molte applicazioni reali, con un impatto evidente nei social network, dove aiutano a suggerire nuovi contenuti o profili da seguire, o negli e-commerce, dove guidano l'utente verso nuovi acquisti potenzialmente rilevanti.

Alla base di questi sistemi vi è la capacità di rappresentare utenti e oggetti all'interno di uno spazio latente, tramite vettori detti *embedding*, che catturano le relazioni implicite tra gli elementi coinvolti.

In questo progetto è stato adottato un approccio basato su Graph Neural Network, in particolare il modello *GraphSAGE*, per generare embedding a partire da un grafo bipartito che collega utenti e film sulla base dei rating espressi. Questi embedding sono stati poi utilizzati per addestrare un modello di regressione con l'obiettivo di predire il rating che un utente potrebbe assegnare a un film non ancora visto.

L'obiettivo finale è valutare la qualità delle rappresentazioni latenti apprese e la loro efficacia nel supportare compiti di previsione e altri scenari simili di raccomandazione.

Chapter 2

Dataset

2.1 MovieLens Dataset

Per questo progetto è stato utilizzato il dataset **MovieLens 100k**. Il dataset contiene informazioni su:

- **Utenti**: identificati da un ID univoco
- **Film**: ciascun film ha un ID, un titolo e un elenco di *generi* associati.
- **Rating**: ogni interazione tra utente e film è rappresentata da un voto tra 1 e 5

2.2 Costruzione della Rete Bipartita

Come mostrato nella figura 2.1, la rete è stata modellata come una **rete bipartita**, ovvero un grafo costituito da due insiemi distinti di nodi, dove gli archi possono connettere solo nodi appartenenti a insiemi diversi. In questo contesto, gli insiemi rappresentano utenti e film, e i collegamenti corrispondono alle interazioni (rating verso il film). Questa struttura riflette fedelmente la natura dei dati e rende possibile l'applicazione di tecniche basate su grafi, come *GraphSAGE*.

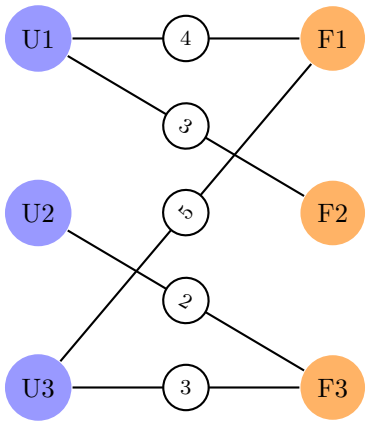


Figure 2.1: Esempio rete bipartita tra utenti (blu) e film (arancione), con pesi sugli archi che rappresentano i rating degli utenti.

2.3 Engineering degli Attributi dei Nodi

Ogni nodo nella rete è arricchito con un vettore di attributi che cattura le sue caratteristiche principali.

Vettore Film

Per ciascun film avremo:

- **film_id**: un identificativo univoco per ciascun film.
- **genre_x**: Dati N generi avremo N colonne in **one-hot encoding** dei generi disponibili.
- **median**: il **rating mediano** ricevuto dal film, calcolato come mediana di tutti i rating forniti dagli utenti.

film_id	genre_unknown	genre_...	genre_Western	**median**
1	0	...	1	4.0

Table 2.1: Esempio di vettore per un film.

Vettore Utenti

Per ciascun utente avremo:

- **user_id**: un identificativo univoco per ciascun utente.
- **genre_x**: somma normalizzata dei rating che ha effettuato l'utente verso il genre_x
- **median**: il rating mediano assegnato dall'utente

user_id	genre_unknown	genre_...	genre_Western	**median**
1	0.0450	...	0.0150	4.0

Table 2.2: Esempio di vettore per un utente.

Raffinamento dei pesi

Per dare maggior peso ai generi a cui l'utente è più appassionato, è stato effettuato un raffinamento nelle colonne genre_x.

In particolare, ciascun punteggio è stato confrontato con la media globale del genere corrispondente, ottenendo uno *score relativo* che misura quanto l'interesse dell'utente per un genere si discosta dalla media generale.

Successivamente, i valori ottenuti sono stati normalizzati tramite Min-Max scaling e riscalati per ottenere una distribuzione di preferenza

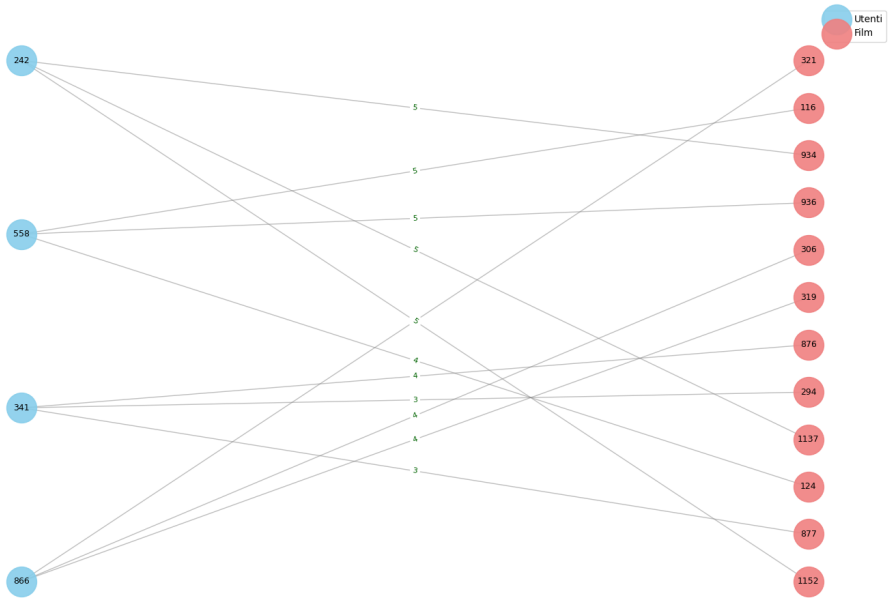


Figure 2.2: Visualizzazione della rete bipartita creata con NetworkX

2.4 Setup training

In un primo momento, la rete è stata costruita utilizzando la libreria `NetworkX`, per rappresentare in modo esplicito la struttura bipartita tra utenti e film. Una volta definita la rete, è stata convertita in un oggetto `PyTorch Geometric`, più adatto per l'addestramento.

Successivamente, è stato effettuato lo **split degli archi** del grafo. Non è stato fatto uno split sui nodi perché l'informazione chiave del modello risiede nelle interazioni tra utenti e film. Suddividere i nodi avrebbe comportato l'esclusione completa di alcuni utenti o film dal grafo, compromettendo la struttura bipartita

La suddivisione degli archi è stata effettuata in modo casuale, con le seguenti caratteristiche:

- train: 80%
- validation: 10%
- test: 10%

Chapter 3

Implementazione

3.1 Intro GraphSAGE

GraphSAGE (*Graph Sample and Aggregate*) è una tecnica di apprendimento di embedding di nodi in grafi che sfrutta un approccio basato sull'aggregazione dei vicini locali.

Questa metodologia è ideale per il nostro grafo bipartito utente-film, poiché consente di apprendere rappresentazioni che integrano le caratteristiche degli utenti e dei film insieme alla loro struttura di interazione.

Come funziona

Il modello GraphSAGE opera mediante una serie di layer di convoluzione sui grafi che aggiornano iterativamente le rappresentazioni dei nodi combinando le loro caratteristiche con quelle dei vicini. Formalmente, dato un nodo v con rappresentazione (embedding) $h_v^{(k)}$ al layer k, la sua nuova rappresentazione al layer $k+1$ viene calcolata aggregando le informazioni dai nodi nel suo vicinato $N(v)$:

$$h_{N(v)}^{(k)} = \text{AGGREGATE}^{(k)}\left(\{h_u^{(k)} \mid u \in N(v)\}\right)$$

$$h_v^{(k+1)} = \sigma\left(W^{(k)} \cdot \text{CONCAT}\left(h_v^{(k)}, h_{N(v)}^{(k)}\right)\right)$$

dove $\text{AGGREGATE}^{(k)}$ è una funzione di aggregazione, come media, somma o una rete neurale, e σ è una funzione di attivazione non lineare. $W^{(k)}$ è una matrice di pesi appresa durante l'addestramento.

Il numero di layer (L) definisce la profondità del *grafo computazionale*, ovvero il numero di passaggi attraverso cui le informazioni vengono propagate e aggregate nel modello. Con ogni

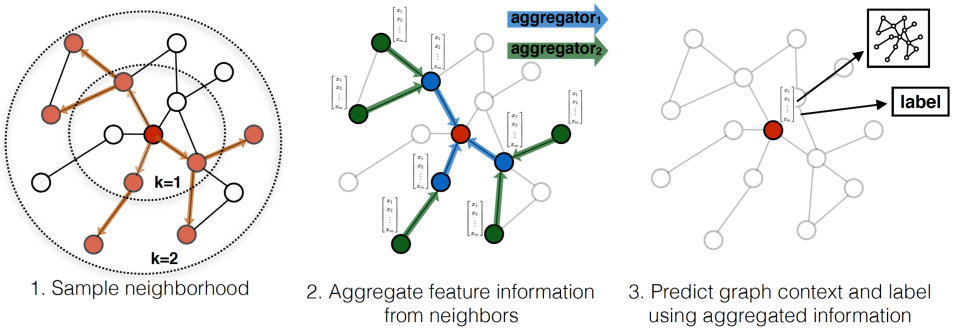


Figure 3.1: Aggregazione in GraphSAGE

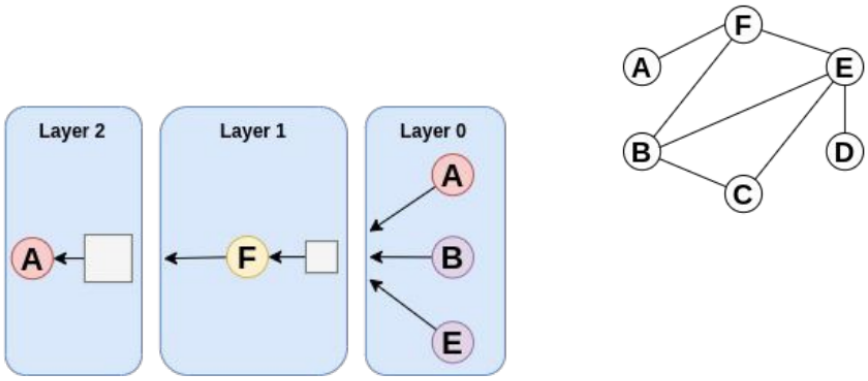


Figure 3.2: Graph convolutional Network

layer aggiuntivo, il nodo riceve informazioni da un livello più esteso del grafo: dopo L layer, la rappresentazione finale di un nodo riflette le informazioni provenienti dai nodi entro L hop di distanza.

3.2 Architettura implementata

L'architettura implementata è la seguente:

```
from torch_geometric.nn import SAGEConv
class GraphSAGE(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels,
        num_layers, dropout):
        super().__init__()

        self.convs = torch.nn.ModuleList()

        # Primo layer
        self.convs.append(SAGEConv(in_channels, hidden_channels))

        # Layer intermedi
        for _ in range(num_layers - 2):
            self.convs.append(SAGEConv(hidden_channels, hidden_channels))

        # Ultimo layer
        self.convs.append(SAGEConv(hidden_channels, out_channels))

        self.dropout = dropout

    def forward(self, x, edge_index):
        for i, conv in enumerate(self.convs):
            x = conv(x, edge_index)

            if i < len(self.convs) - 1:
                x = F.relu(x)
                x = F.dropout(x, p=self.dropout, training=self.training)

        return x
```

Algorithm 3.1: Implementazione del modello GraphSAGE

I parametri usati:

- **in_channels**: dimensioni feature di ingresso, in questo caso la dimensione feature dei nodi.
- **num_layers**: numero totale di layer nella rete.
- **hidden_channels**: dimensione degli embedding negli hidden layer.
- **out_channels**: dimensione dell'embedding finale
- **dropout**: probabilità di dropout

Durante la propagazione, ogni layer applica una convoluzione sui nodi del grafo, producendo embedding intermedi che vengono trasformati attraverso la funzione di attivazione non lineare (ReLU) e successivamente regolarizzati tramite dropout.

3.3 Esperimenti

3.3.1 Setup

Gli esperimenti sono stati suddivisi in due fasi principali: la **generazione degli embedding** e la **predizione dei rating**, che verranno descritte nel dettaglio nelle sezioni successive.

Per monitorare l'andamento degli esperimenti, è stato utilizzato **TensorBoard**, uno strumento che fornisce un'interfaccia interattiva per visualizzare metriche.

Infine, l'**ottimizzazione degli iperparametri** è stata implementata tramite la libreria `Optuna`, un framework per l'ottimizzazione automatica basato su strategie di ricerca bayesiana. Questo approccio consente di esplorare in modo efficiente lo spazio degli iperparametri, valutando ogni configurazione in base a una metrica obiettivo. Optuna implementa anche un meccanismo di *pruning*, che interrompe in anticipo le esecuzioni meno promettenti, riducendo i tempi di ricerca e migliorando la qualità complessiva delle soluzioni trovate.

3.3.2 Generazione embedding

Per la generazione degli embedding, l'ottimizzazione è stata guidata dal compito di **link prediction**. In questo modo, il modello viene incentivato a produrre embedding che riflettano la compatibilità tra utenti e film.

Sono stati condotti due esperimenti principali:

- **Exp0**: utilizzando negli utenti solo le feature normalizzate di **genre_x**
- **Exp1**: includendo il raffinamento dei pesi per le colonne **genre_x**

Setup

Per gli iperparametri sono stati esplorati i seguenti intervalli:

- `hidden_channels` $\in \{16, 32, \ldots, 128\}$
- `out_channels` $\in \{16, 32, \ldots, 128\}$
- `num_layers` $\in [1, 5]$
- `dropout` $\in [0.0, 0.8]$

Come ottimizzatore è stato utilizzato `Adam`, con un tasso di apprendimento pari a `0.001`.

Table 3.1: Configurazione ottimale degli iperparametri

Parametro	**Valore**
Hidden Channels	128
Output Channels	32
Numero di Layer	5
Dropout	0.0382
Learning Rate	0.001

Training

Per allenare il modello, si calcolano gli embedding dei nodi. Successivamente, per ogni coppia di nodi reale (`positive` samples) e non (`negative` samples), si combinano i rispettivi embedding attraverso un `dot product`, da cui si ricava uno score scalare che rappresenta la probabilità di connessione tra i due nodi.

Valutazioni

A ogni score viene assegnata un'etichetta binaria: 1 per i `positive`, 0 per quelli `negative`.

Queste etichette vengono utilizzate per calcolare la *binary cross-entropy loss*, che penalizza le previsioni errate, spingendo il modello ad assegnare punteggi alti alle connessioni reali e bassi a quelle assenti.

Le prestazioni vengono poi misurate tramite due metriche:

- **ROC AUC score**: misura quanto bene il modello riesce a distinguere tra le due classi (`positive` / `negative` samples), valutando la capacità di assegnare punteggi più alti ai collegamenti veri rispetto a quelli falsi.
- **Average Precision** (`average_precision_score`): La precisione indica in media la capacità di predirre link reali.

Nelle sezioni seguenti sono riportati i risultati più rilevanti per ogni esperimento.

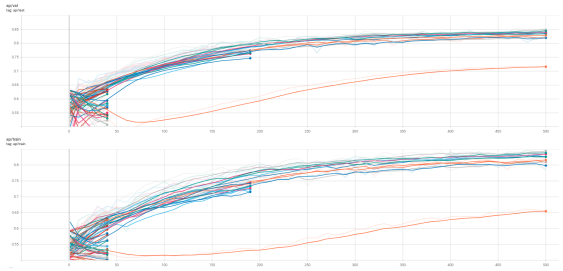
exp1

Sono stati eseguiti **50 iterazioni** di tuning degli iperparametri. Come si osserva nella Figura 3.3, molte configurazioni non superano neanche le 50 epoche a causa dello *early stopping* automatico operato da Optuna.

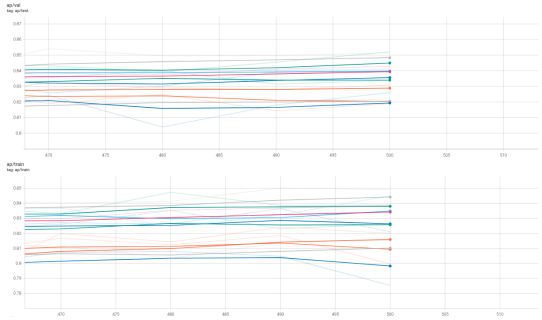
Le metriche osservate nelle migliori configurazioni risultano le seguenti:

- **Average Precision**: [0.82 - 0.85]
- **ROC AUC**: [0.83 - 0.86]
- **Binary Cross Entropy Loss**: [0.56 - 0.64]

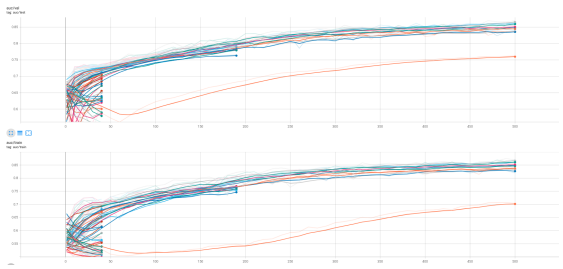
In tutte le metriche analizzate, si è osservato che la curva di colore grigio ha mostrato performance superiori rispetto alle altre. I parametri sono riportati nella Tabella 3.1. Le metriche ottenute sul dataset di test, sono riportate in Tabella 3.2.



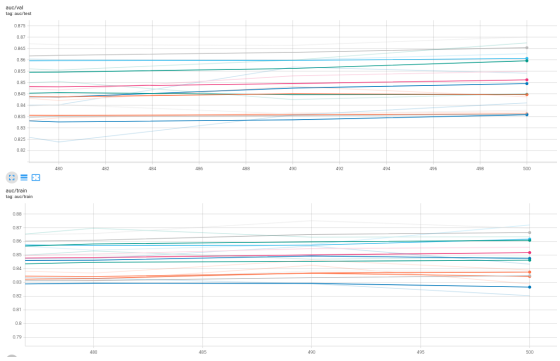
(a) Average Precision - Tutte le run



(b) Average Precision - Migliori run

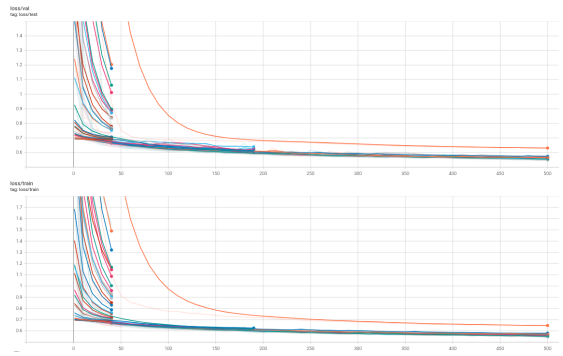


(c) ROC AUC - Tutte le run

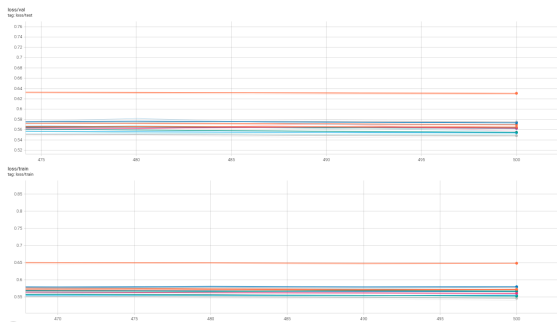


(d) ROC AUC - Migliori run

Figure 3.3: Metriche durante i trial: *Average Precision* (sopra) e *ROC AUC* (sotto), sia per tutte le configurazioni che per le migliori.



(a) Loss - Tutte le run



(b) Loss - Migliori run

Figure 3.4: (Binary Cross Entropy) durante training (sotto) e validation (sopra)

Table 3.2: Metriche per la migliore run sul dataset di test

Metrica	**Valore**
Loss (Binary Cross Entropy)	0.656
Accuracy	0.825
Precision	0.770

exp2

Si nota un netto miglioramento con il raffinamento.

Configurazione ottimale degli iperparametri

- `Hidden Channels`: 64
- `Output Channels`: 16
- `Numero di Layer`: 4
- `Dropout`: 0.000048
- `Learning Rate`: 0.001

Prestazioni sul test set

- **Loss (Binary Cross Entropy)**: 0.554
- **Accuracy**: 0.862
- **Precision**: 0.848

3.3.3 Predizione rating

Una volta fissato il miglior modello di embedding, si ottiene uno spazio di rappresentazione vettoriale in cui è possibile effettuare ulteriori analisi, come la predizione dei rating.

Per effettuare delle raccomandazioni è stato allenato un regressore. Gli esperimenti condotti sono stati:

- **exp0**: Costruzione di una rete neurale utilizzando PyTorch.
- **exp1**: Utilizzo di XGBoost, libreria che implementa algoritmo basato su alberi decisionali.

Analogamente alla predizione del rating, sono stati utilizzati TensorBoard e Optuna.

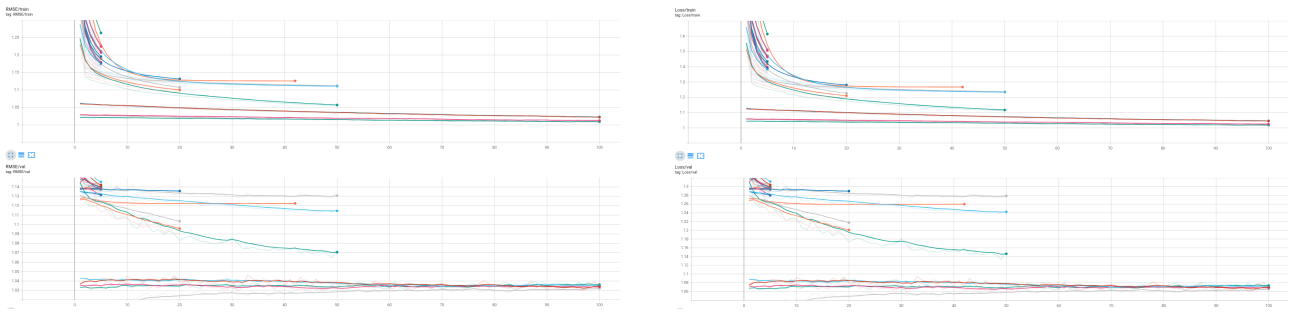
Essendo un problema di regressione, per la valutazione delle prestazioni è stato adottato il Root Mean Square Error (RMSE), che misura la deviazione quadratica media tra i valori predetti e quelli reali.

exp0

La rete implementata è una semplice **MLP** (Multi-Layer Perceptron) che prende in input la concatenazione degli embedding utente e film. L'input attraversa un numero variabile di `hidden_layer`, ciascuno di dimensione `hidden_dim`. Ogni layer applica una trasformazione lineare, seguita da una funzione di attivazione ReLU e da un'operazione di dropout per la regolarizzazione.

Iper-parametri esplorati

- `num_hidden` $\in [1, 5]$
- `hidden_dim` $\in \{16, 128\}$
- `dropout` $\in [0.0, 0.8]$



RMSE Loss

Figure 3.5: Metriche regressore

Prestazioni validation set

- **Root Mean Squared Error (RMSE)**: [1.009 – 1.022]

Migliori iper-parametri trovati

- `num_hidden` = 4
- `hidden_dim` = 80
- `dropout` = 0.05

Prestazioni test set

- **Root Mean Squared Error (RMSE)**: **1.0081**

Il miglior risultato rappresenta un errore ancora piuttosto elevato in termini assoluti. Sebbene il modello riesca a sfruttare le info degli embedding il margine di miglioramento rimane ampio.

exp1

A fronte delle prestazioni limitate ottenute con la rete MLP, è stato sperimentato l'uso di **XGBoost**, una libreria basata su *gradient boosting* per alberi decisionali. XGBoost costruisce il modello in modo incrementale, combinando più alberi deboli (poco profondi) che correggono iterativamente gli errori residui dei precedenti, ottimizzando una funzione obiettivo — in questo caso RMSE.

Iper-parametri esplorati

- `n_estimators`: numero totale di alberi nel modello; $\in [50, 150]$
- `max_depth`: profondità massima di ciascun albero; $\in [3, 7]$
- `learning_rate`: $\in [0.01, 0.3]$

Prestazioni validation set

- **Root Mean Squared Error (RMSE)**: [0.96 – 1.10]

Migliori iper-parametri trovati

- `n_estimators` = 136
- `max_depth` = 5
- `learning_rate` = 0.2859

Prestazioni test set

- **Root Mean Squared Error (RMSE): 0.9574**

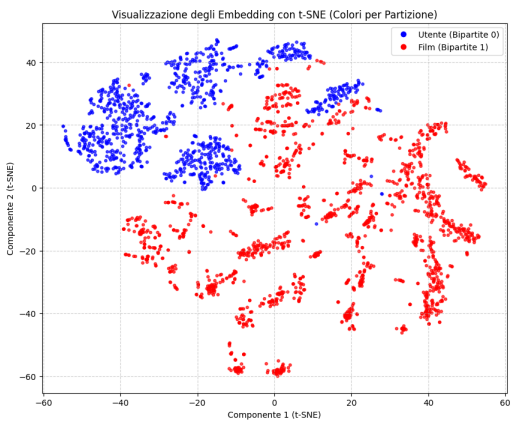
Il risultato ottenuto con XGBoost è sensibilmente migliore rispetto alla MLP. Sembra più robusto nel modellare questo tipo di embedding. Dall'altro lato, la MLP si è rivelata particolarmente sensibile alla scelta degli iperparametri, rendendo l'ottimizzazione più instabile e limitando la capacità del modello di generalizzare correttamente.

3.4 Ulteriori valutazioni

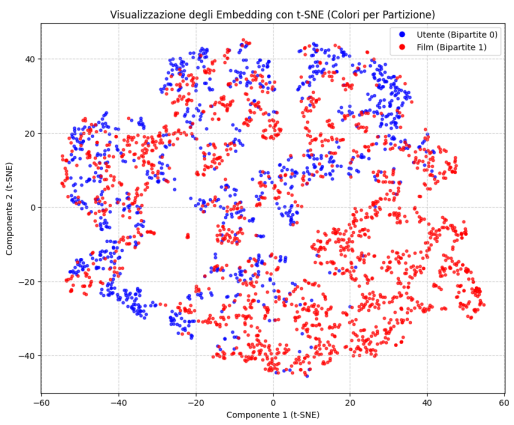
3.4.1 Proiezioni t-SNE

La proiezione t-SNE consente di visualizzare in modo intuitivo la distribuzione degli embedding in uno spazio bidimensionale, mettendo in evidenza come il modello organizza i nodi e se riesce a costruire una rappresentazione coerente delle relazioni apprese. Questa tecnica aiuta a valutare la qualità dello spazio latente generato, in particolare la capacità del modello di raggruppare entità simili e separare quelle diverse.

Di seguito sono riportate le proiezioni relative agli esperimenti `exp0` ed `exp1` della link prediction. Nel caso di `exp0`, si osserva una netta divisione tra utenti e film, segno che senza il raffinamento le rappresentazioni tendono a rimanere più isolate e meno strutturate. Al contrario, in `exp1`, contribuisce a una maggiore coerenza nello spazio latente: utenti e film risultano più mescolati.



t-SNE embedding - exp0



t-SNE embedding - exp1

Figure 3.6: Proiezioni t-SNE degli embedding nei due esperimenti

3.4.2 Similarità film - film

Dalla rappresentazione degli embedding dei film potremmo già ottenere una misura di similarità basata principalmente sulle colonne in one-hot encoding. Tuttavia, se vogliamo catturare similarità più complesse, che tengano conto anche delle interazioni tra utenti possiamo utilizzare gli embedding.

Di seguito sono riportati due esempi di similarità calcolata tramite la similarità coseno.

Esempio 1: Film di partenza - *Toy Story (1995)*

- Fugitive, The (1993)
- Die Hard (1988)
- Mission: Impossible (1996)
- In the Line of Fire (1993)
- Best Men (1997)

Esempio 2: Film di partenza - *Star Wars (1977)*

- Independence Day (1996)
- Aliens (1986)
- Return of the Jedi (1983)
- Menace II Society (1993)
- City of Lost Children, The (1995)

Osservazioni Le raccomandazioni ottenute mostrano alcune incongruenze evidenti dal punto di vista tematico o di genere. Ad esempio, per *Toy Story*, la lista include titoli d'azione e thriller che difficilmente verrebbero associati a un film di animazione per famiglie.

Allo stesso modo, per *Star Wars*, accanto a sequel o film di fantascienza come *Return of the Jedi* e *Aliens*, compaiono pellicole molto diverse come *Menace II Society* o *City of Lost Children*.

Queste anomalie possono derivare dal fatto che la similarità è calcolata su embedding prodotti da un modello GraphSAGE addestrato su interazioni piuttosto che caratteristiche esplicite di contenuto.

Di conseguenza, gli embedding catturano pattern complessi di preferenze degli utenti che sono molto varie.

Chapter 4

Conclusioni

In questo progetto è stato esplorato un approccio basato su Graph Neural Network per la raccomandazione di film, partendo da un grafo bipartito che collega utenti e titoli attraverso i rating. Utilizzando il modello GraphSAGE, sono stati generati embedding per ciascun nodo, successivamente impiegati come input per i regressori con l'obiettivo di predire il valore del rating verso i film.

Le valutazioni hanno mostrato che, sebbene le performance nella fase di link prediction siano risultate buone, la predizione dei rating tramite regressore non ha raggiunto risultati altrettanto convincenti. Questo può essere dovuto al fatto che gli embedding appresi non sono stati ottimizzati direttamente per la regressione, ma per la semplice previsione di connessioni. Inoltre, la struttura sbilanciata del dataset, caratterizzata dal fenomeno della *long tail* (dove molti film hanno pochissime valutazioni), può aver limitato la capacità del modello di generalizzare su elementi poco rappresentati.

Tra i possibili sviluppi futuri si potrebbe considerare l'integrazione di una componente temporale nelle feature, al fine di modellare l'evoluzione delle preferenze nel tempo. Un'altra direzione è l'adozione di un approccio end-to-end in cui regressore ed embedder vengano ottimizzati congiuntamente. Inoltre, sarebbe utile estendere gli esperimenti a dataset più estesi e ricchi, come MovieLens 1M. Questi miglioramenti potrebbero contribuire a potenziare sia la qualità delle rappresentazioni latenti sia l'accuratezza nella predizione finale dei rating.