Movie Recommender System Report

Ashera Dyussenova
BS20-DS
a.dyusenova@innopolis.university

December 3, 2023

1 Introduction

In the face of information overload online, recommendation systems have become crucial for providing users with personalized content suggestions. These systems analyze user preferences and behaviors to predict and offer tailored items or content, making it easier for individuals to navigate vast online choices. By alleviating information overload, recommendation systems enhance user experience by saving time and facilitating content discovery across various platforms.

Graph Neural Networks (GNNs) have emerged as powerful tools in the realm of recommendation systems due to their inherent capability to model complex relationships and dependencies present in user-item interaction data. Unlike traditional recommendation methods, GNNs can effectively capture and leverage the intricate graph structures inherent in user-item interaction networks. In recommendation systems, users and items can be naturally represented as nodes in a graph, and the edges between them denote interactions.

The utility of GNNs in this context lies in their ability to learn and propagate information across the graph, enabling them to capture latent patterns and user-item associations. GNNs excel at incorporating collaborative filtering principles by aggregating information from a user's neighbors or items with similar characteristics. This fosters a more nuanced understanding of user preferences and helps overcome data sparsity issues, making GNNs particularly beneficial in scenarios with limited or sparse interaction data.[1]

2 Data analysis

In this work i used MovieLens 100k dataset.[2] **General information about the dataset:**

- It consists of 100,000 ratings from 943 users on 1682 movies
- Ratings are ranged from 1 to 5
- Each user has rated at least 20 movies
- It contains simple demographic info for the users (age, gender, occupation, zip code)

Dataset consists of three main set: user, item and data. User dataset contains demographic information about user itself: age, gender, occupation and zip-code. Item dataset is about movies: title, realise date and genre. I decided to focus on the main data, where given 4 main columns from which it would be possible to extract correlation and construct graph: user, movie, raiting and timestamp.

2.1 Exploratory Data Analysis

Main dataset containing user id, movie id, raiting and timestamp are in integer format, so there is no need to convert it. Also i decided to add some demographical about user. Gender and occupation transformed and encoded to the integer values, and after that all columns normilized including age and zipcode.

Dataset contains 1682 movies and 943 unique users. User_id ranged from 1 to 943 and movie_id range from 1 to 1682, so we can highlight that all movies present and id will not cause out off bound issues. After data analyzis i prepare data to construct graph.

2.2 Preproccessing

After initial data preparation, to create graph it is important to understand the way how to represent nodes and edges in the graph.

COO coordinate format

It is one of the ways to store sparse matrix in efficient way is to use COO coordinate graph representation. It consist only of three list: first and second lists represents edge pair and last the value of the edge. Example:

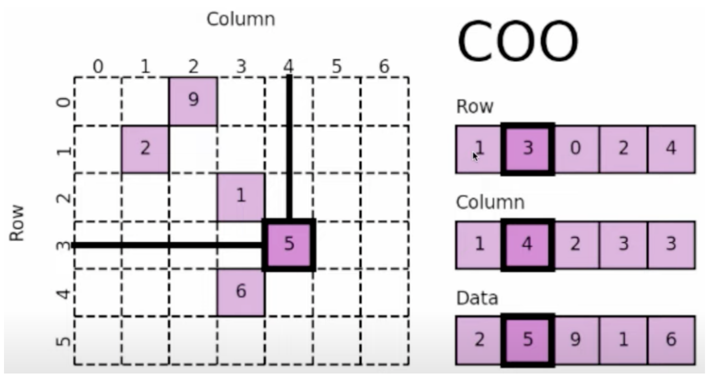

Figure 1: COO graph representation

Bipartite Graph

A bipartite graph is a graph whose vertices can be divided into two disjoint sets such that every edge connects a vertex from one set to a vertex in the other set. In other words, there are no edges that connect vertices within the same set.

Most of the recommendation systems constructed on the bipartite graph. In the structure of the graph we have users on the right side and items(movies) on the other side.

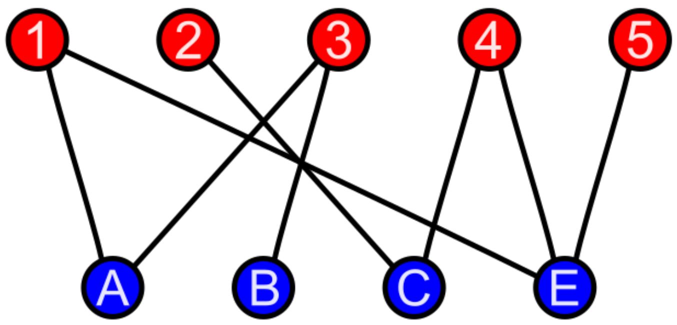

Figure 2: Bipartite Graph

How to construct the graph?

- Use 3.5 as the threshold for positive rating. So, any interaction above rating 3.5 is considered an edge.

3 Model Implementation

In my work i implement Light Graph Convolution Network (LightGCN) Self-Supervised Learning to construct recommendation system.[3]

Self-supervised learning on graphs is a type of machine learning paradigm where the model learns from the graph structure itself without relying on external labels or annotations. Graphs are mathematical structures that represent relationships between entities, with nodes representing entities and edges representing connections or interactions between them. Self-supervised learning aims to leverage the inherent structure of graphs to train models in a semi-supervised or unsupervised manner.

Signals come from graph themselves. In terms of link prediction we predict if two nodes are connected.

The LightGCN model is designed for collaborative filtering in recommendation systems. It operates on a user-item interaction graph, where nodes represent users and items, and edges indicate user-item interactions. The model leverages graph convolutional networks (GCN) for message passing between users and items.

The key idea behind LightGCN is the simplification of the graph convolution operation. Instead of using complex weight matrices for different layers, LightGCN employs a straightforward propagation rule, where the embedding of a user or item in the next layer is computed as the weighted sum of embeddings of its neighbors in the current layer. The weights are normalized by the square root of the degrees of the nodes involved in the interaction.

The model consists of multiple layers, each performing the described message passing operation. The trainable parameters are the initial embeddings for users and items in the 0-th layer. The final embeddings for users and items are obtained by combining the embeddings from all layers with trainable weights. The model prediction for the interaction strength between a user and an item is computed as the inner product of their final embeddings.

The matrix form of LightGCN involves multi-scale diffusion, where embeddings are diffused across multiple hop scales. The final embedding is a combination of embeddings obtained at different scales, each weighted by a hyperparameter. The adjacency matrix is symmetrically normalized to facilitate the diffusion process.

The LightGCN model was implemented using PyTorch, utilizing the matrix form for efficient computation. The model is defined as a subclass of the PyTorch ‘MessagePassing‘ class, with the ‘forward‘ method implementing the message passing process across multiple layers. The model parameters are initialized, and the message passing is performed through the graph layers to obtain the final embeddings for users and items. The resulting embeddings can be used for making recommendations in a collaborative filtering setting.

4 Model Advantages and Disadvantages

4.1 Advantages

1. **Scalability:** LightGCN exhibits scalability due to its simplified model structure. The absence of complex weight matrices and the use of a straightforward propagation rule make it computationally efficient, allowing for training on large-scale datasets.

2. **Simplicity and Interpretability:** The model's simplicity is an advantage, making it easier to understand and interpret. The absence of intricate weight matrices reduces the risk of overfitting and enhances model interpretability, crucial for real-world applications.

3. **Implicit Feedback Handling:** LightGCN is adept at handling implicit feedback in recommendation systems. By focusing on user-item interactions without relying on explicit ratings, the model can effectively capture user preferences in scenarios where users do not provide explicit feedback.

4.2 Disadvantages

1. **Limited Expressiveness:** The simplification of the graph convolution operation in LightGCN, while contributing to its efficiency, may result in limited expressiveness. The model may struggle to capture more complex relationships and intricate patterns present in user-item interaction data compared to more sophisticated graph neural network architectures.

2. **Cold Start Problem:** LightGCN may face challenges in scenarios where there is limited initial information about users or items (cold start problem). Since the model heavily relies on user-item interactions, it may struggle to make accurate recommendations for new users or items with minimal interaction history.

3. **Dependency on Hyperparameters:** Like many machine learning models, the performance of LightGCN is sensitive to hyperparameters. Selecting appropriate hyperparameter values, such as the learning rate and regularization strength, is crucial for achieving optimal performance, and tuning these parameters may require careful experimentation.

4. **Lack of Content Information:** LightGCN primarily focuses on collaborative filtering and may not effectively incorporate additional content information about users or items. Models that integrate both collaborative and content-based information might outperform LightGCN in scenarios where rich content information is available.

Understanding these advantages and disadvantages is essential for informed decision-making when choosing LightGCN or any other recommendation system model for a specific application. Fine-tuning and customization based on the characteristics of the dataset and user requirements may be necessary to address limitations and enhance performance.

5 Training Process

The training configuration involves iterating 10,000 times with a batch size of 1024 for each of the 10 epochs. The learning rate is set to 1e-3, and evaluations are conducted every 200 iterations to monitor performance. A learning rate decay is implemented every 200 iterations to adapt model parameters. The evaluation metrics focus on the top 20 recommendations (K=20), providing a specific assessment. Regularization is applied with a lambda value of 1e-6 to prevent overfitting. These parameters collectively aim to strike a balance between convergence, computational efficiency, and model generalization during recommendation system training.

What loss function to use?

- Since it is self-supervised learning we can not relay on rating labels to compute loss function. So we can not use RMSE.
- Then, i use Bayesian Personalized Ranking (BPR) loss, a pairwise objective which encourages the predictions of positive sample to be higher than negative samples for each user.

$$L_{BPR} = -\sum_{u=1}^{M} \sum_{i \in \mathbb{N}_u} \sum_{j \notin \mathbb{N}_u} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda \|E^{(0))}\|^2 \tag{1}$$

$\hat{y}_{ui}$: predicted score of positive sample
$\hat{y}_{uj}$: predicted score of negative sample
λ : hyperparameter which controls L2 regularization strength

The BPR loss function is defined based on pairs of observed and unobserved interactions. Given a user and a positive item that the user has interacted with, along with a negative item that the user has not interacted with, the BPR loss penalizes the model if the predicted score for the positive item is not higher than the predicted score for the negative item.

The BPR loss function includes a regularization term to prevent overfitting. This term penalizes the magnitude of the model parameters, helping to control the complexity of the model.

6 Evaluation Metrics

In evaluation of model i used the following metrics

$$\text{Recall} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Precision} = \frac{TP}{TP + FN} \tag{3}$$

Recall@k and Precision@k is just applying only the topK recommended items and then for the overall Recall@k and Precision@k, it's just averaged by the number of users

Dicounted Cumulative Gain (DCG) at rank position p is defined as:

$$\text{DCG}_\text{p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{4}$$

p: a particular rank position
$rel_i \in \{0, 1\}$: graded relevance of the result at position i

Idealised Dicounted Cumulative Gain (IDCG), namely the maximum possible DCG, at rank position p is defined as:

$$\text{IDCG}_\text{p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{5}$$

$|REL_p|$: list of items ordered by their relevance up to position p

Normalized Dicounted Cumulative Gain (NDCG) at rank position p is defined as:

$$\text{nDCG}_\text{p} = \frac{\text{DCG}_p}{\text{nDCG}_p} \tag{6}$$

Specifically, we use the metrics recall@K, precision@K, and NDCG@K. @K indicates that these metrics are computed on the top K recommendations.

7 Results

Training error and Validation error loss slowly goes down, validation error little bit smoother because i sample, every 200 iterations, so it is less data point.

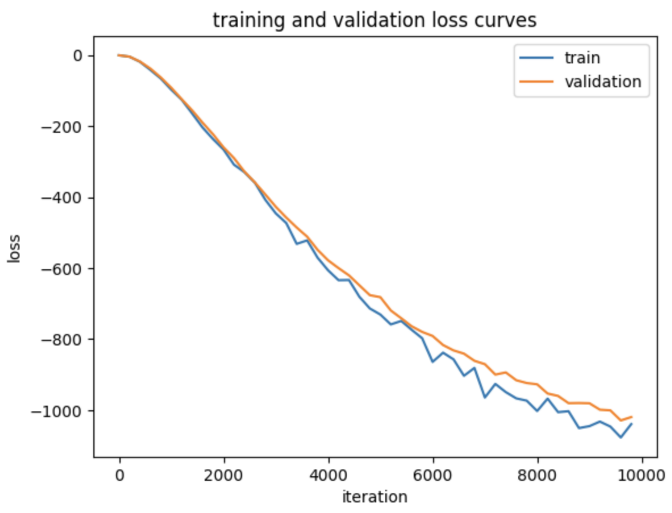

Figure 3:

The Recall at K curve reveals an interesting trend in the model's performance with increasing values of K. Initially, there is a substantial rise in recall, reaching around 14%. However, as K continues to increase, the curve exhibits a noticeable deceleration and eventually levels off. This plateau suggests that the model's ability to identify relevant items within the top K recommendations becomes more stable beyond the 14% recall mark. The sustained recall rate indicates a certain limit in the system's effectiveness, implying that additional recommendations beyond this point may not significantly enhance the recall metric. To improve and better understand the model's behavior in the higher K ranges, further analysis and fine-tuning may be required.

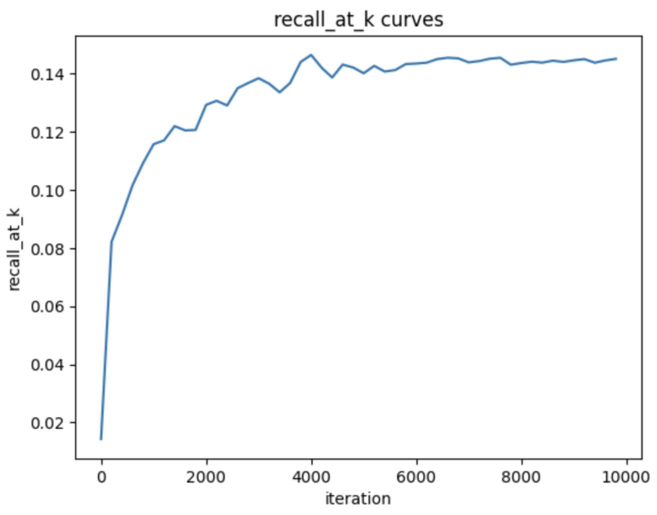

Figure 4:

References

[1] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. Graph neural networks for recommender system. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 1623–1625, 2022.

[2] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015.

[3] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.