Movie Recommender Systems with PyG

LightGCN-based Recommender Systems

Marco Zhao · Follow

Published in Stanford CS224W: Machine Learning with Graphs · 9 min read · May 13, 2023





By Taiqi Zhao, and Weimin Wan as part of the Stanford CS224W course project. Here we build a graph neural network recommender system on MovieLens 100K Dataset using PyG. We also implement the LightGCN model in the paper "LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation" by He, X. et al. published in 2020 [1].

Google Colaboratory

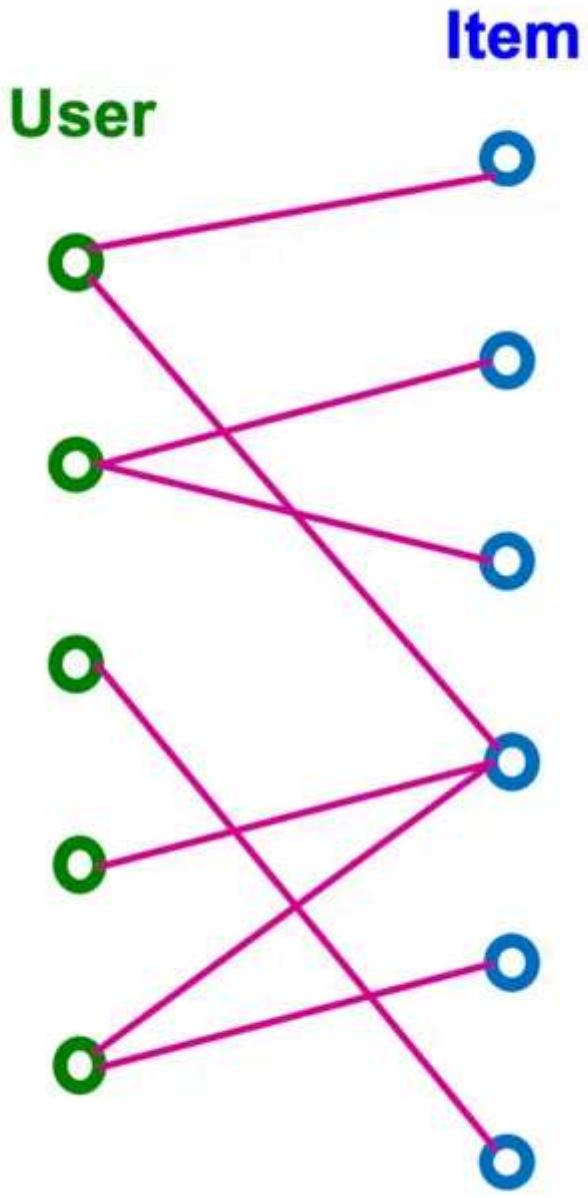
Movie Recommender Systems in PyG

colab.research.google.com

Recommender System Introduction

A recommender system is a type of artificial intelligence algorithm that provides personalized suggestions or recommendations to users based on their preferences, interests, and behavior. These systems are widely used in various industries such as e-commerce, entertainment, social media, and online advertising to help users find relevant products, services, or content.

A recommender system can be naturally modeled as a bipartite graph with two node types: users and items (movies, products, ads, etc.). In the graph, edges connect users and items. They indicate various types of user-item interaction (submit ratings, purchase, click, etc.).



Recommender System as a bipartite graph

A recommender system can utilize the past user-item interaction, and predict new items each user will interact with in the future. This aligns with the ultimate goal of a recommender system to improve user satisfaction and engagement by providing personalized and relevant recommendations.

Why use PyG?

Given the nature of modeling a recommender system as a graph object, it would be very beneficial to utilize graph machine learning techniques to help us make accurate predictions, and this is where PyG comes to help.

PyG, short for PyTorch Geometric, is a Python library for deep learning on irregularly structured input data, such as graphs. It is built on top of PyTorch, a popular deep learning framework, and provides a set of tools and utilities for developing and training neural networks that operate on graph-structured data. PyG includes various graph neural network (GNN) architectures, optimization algorithms, and data handling utilities, making it a powerful tool for a wide range of graph-related tasks, such as node classification, link prediction, graph classification, and more.

A Peek into MovieLens Dataset

Photo by Jakob Owens on Unsplash

Ahoy there, movie buffs! Have you ever wondered what kind of movies your fellow cinephiles like to watch? Well, look no further than the MovieLens dataset! Here we use the MovieLens 100K dataset with 100,000 ratings from 1000 users over 1700 movies.

The dataset contains two tables that are relevant to our analysis: movies data and ratings data. The movies table records the title of a movie and its labeled genres. The entries in ratings data are historical ratings by users in real life. Each record contains the ID of the user who submitted the rating, the ID of the rated movie, the rating score, and the timestamp for that rating. The ratings are on a scale of 0 to 5, where a higher rating indicates higher user satisfaction.

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama|Romance
5	Father of the Bride Part II (1995)	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|Thriller

A peak into movies data

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
5	1	70	3.0	964982400
6	1	101	5.0	964980868
7	1	110	4.0	964982176
8	1	151	5.0	964984041
9	1	157	5.0	964984100

A peak into ratings data

Parsing the Movie Review into Graph

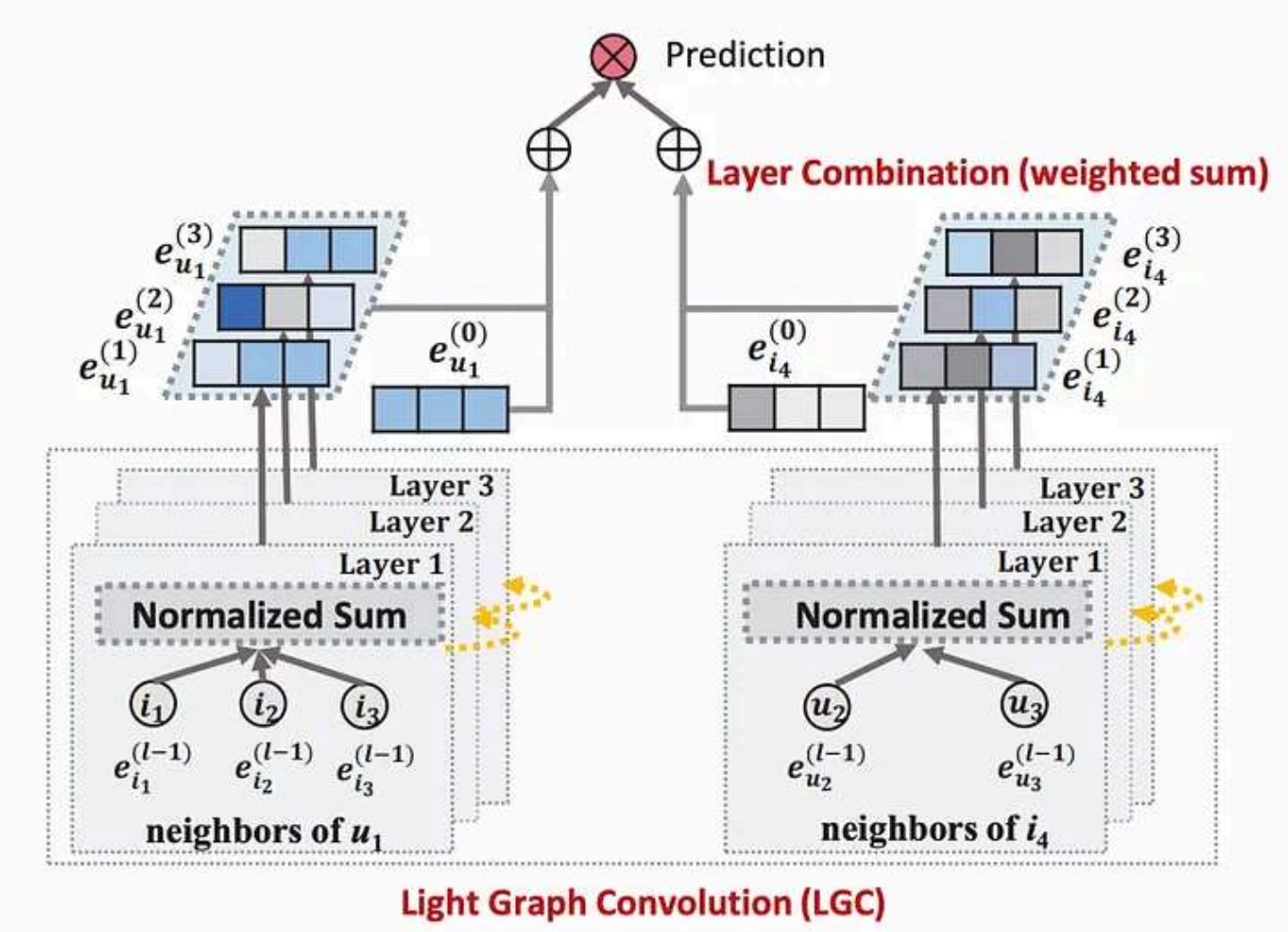
In order to use graph machine learning techniques for link prediction, we will need to construct a connected graph object and then pass it into the PyG library [2]. All that we need is a PyTorch tensor named `edge_index`. It is a tensor of shape (2, num of ratings), where each column stands for a specific rating and the entry on the first row is the user ID, and the entry on the second row is the movie ID. The `edge_index` tensor will then be used for message passing and further propagating.

Next, we divide the complete dataset into training, validation, and testing sets. The partition is made based on the edges in the graph. More specifically, we randomly separate the data into three groups with an 80/10/10 split ratio. This implies that each group will contain a portion of the `edge_index` subset.

During the training phase, we will use mini-batches and select a number of positive and negative edges within each batch[3]. Positive edges refer to observed or training user-item interactions. We aim to punish negative edges during training by allocating them with a greater loss. To achieve this, we will make use of the PyG function called `structured_negative_sampling`, which selects a negative edge for every positive edge in the graph as defined by `edge_index`.

LightGCN Model Architecture

LightGCN is a recommender system model that belongs to the family of graph convolutional networks (GCNs). The primary objective of LightGCN is to provide personalized recommendations by exploiting the user-item interaction graph's structure. Unlike other graph-based recommendation methods that use complex graph neural networks (GNNs), LightGCN omits feature transformation and non-linearity, which requires fewer parameters and has a faster training time. The model's architecture involves performing several layers of graph convolutions to propagate user-item embeddings, which are then used to generate personalized recommendations. LightGCN has demonstrated competitive performance in terms of recommendation quality while being computationally efficient. For more information, you can find the original paper here.



An illustration of LightGCN model architecture [1]

Light Graph Convolution

Light graph convolution is the key part of the LightGCN model. Between each layer, LightGCN uses the following propagation rule for user and item embeddings:

$$e_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} e_i^{(k)} \qquad e_i^{(k+1)} = \sum_{u \in N_i} \frac{1}{\sqrt{|N_i|}\sqrt{|N_u|}} e_u^{(k)}$$

N_u: the set of all neighbors of user u (items liked by u)

N_i: the set of all neighbors of item i (users who liked i)

$e_u^{(k)}$: k-th layer user embedding

$e_i^{(k)}$: k-th layer item embedding

Encoder and Decoder

The LightGCN model's only adjustable parameters are the initial embeddings for each user and item in the 0-th layer. The final embeddings for all users and items are created by merging the embeddings acquired at each propagation layer using the following formula:

$$e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)} \quad e_i = \sum_{k=0}^{K} \alpha_k e_i^{(k)}$$

where ***alpha_k*** is a hyperparameter that weights the contribution of the k-th layer embedding to the final embedding.

The model prediction is then obtained by taking the inner product of the final user and item embeddings.

$$\hat{y}_{ui} = e_u^T e_i$$

In the code snippet below, we walk through an implementation of the LightGCN model:

```
# defines LightGCN model
class LightGCN(MessagePassing):
    """
    LightGCN Model, see reference: https://arxiv.org/abs/2002.02126
    """

    def __init__(self, num_users: int, num_items: int,
                 hidden_dim: int, num_layers: int):
        super().__init__()
        self.num_users = num_users
        self.num_items = num_items
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers

        self.users_emb = nn.Embedding(self.num_users, self.hidden_dim)
        self.items_emb = nn.Embedding(self.num_items, self.hidden_dim)

        nn.init.normal_(self.users_emb.weight, std=0.1)
        nn.init.normal_(self.items_emb.weight, std=0.1)

    def forward(self, edge_index: SparseTensor):
        """
        Forward pass of the LightGCN model. Returns the init and final
        embeddings of the user and item
        """
        edge_index_norm = gcn_norm(edge_index, False)

        # The first layer, concat embeddings
        x0 = torch.cat([self.users_emb.weight, self.items_emb.weight])
        xs = [x0]
        xi = x0

        # pass x to the next layer
        for i in range(self.num_layers):
            xi = self.propagate(edge_index_norm, x=xi)
            xs.append(xi)

        xs = torch.stack(xs, dim=1)
        x_final = torch.mean(xs, dim=1)

        users_emb, items_emb = \
        torch.split(x_final, [self.num_users, self.num_items])

        return users_emb, self.users_emb.weight, items_emb,
                self.items_emb.weight
    def message(self, x: Tensor) -> Tensor:
        return x
```

```
    def propagate(self, edge_index: SparseTensor, x: Tensor) -> Tensor:
        x = self.message_and_aggregate(edge_index, x)
        return x

    def message_and_aggregate(self, edge_index: SparseTensor, x: Tensor):
        return matmul(edge_index, x)
```

Loss Function

The model's training objective in LightGCN is based on a pairwise Bayesian Personalized Ranking (BPR) loss. When selecting the surrogate losses, they should be differentiable and should align well with the original training objective. Binary loss is a widely-used loss function in classification tasks, where the model takes in an input and has to classify it into one of two pre-set categories. The binary loss is defined as the summation of the loss evolving all positive terms and the loss for all negative terms, respectively. However, in the binary loss, the scores of all positive edges are pushed higher than those of all negative edges. This would unnecessarily penalize model predictions even if the training metric is perfect. The reason for this is binary loss is **non-personalized** so that all the positive edges are set to have higher scores than negative edges. The loss function is considered across all users at once, instead of taking each individual into account. That's why we need this binary loss to be personalized.

Thus, the BPR loss function is defined as follows:

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

where y_ui is the predicted score of a positive sample, and y_uj is the predicted score of a negative sample. The BPR loss is first calculated for a specific user u, then we sum up across all users to generate a system-level BPR loss. A regularization term is also introduced and the lambda value controls the regularization strength.

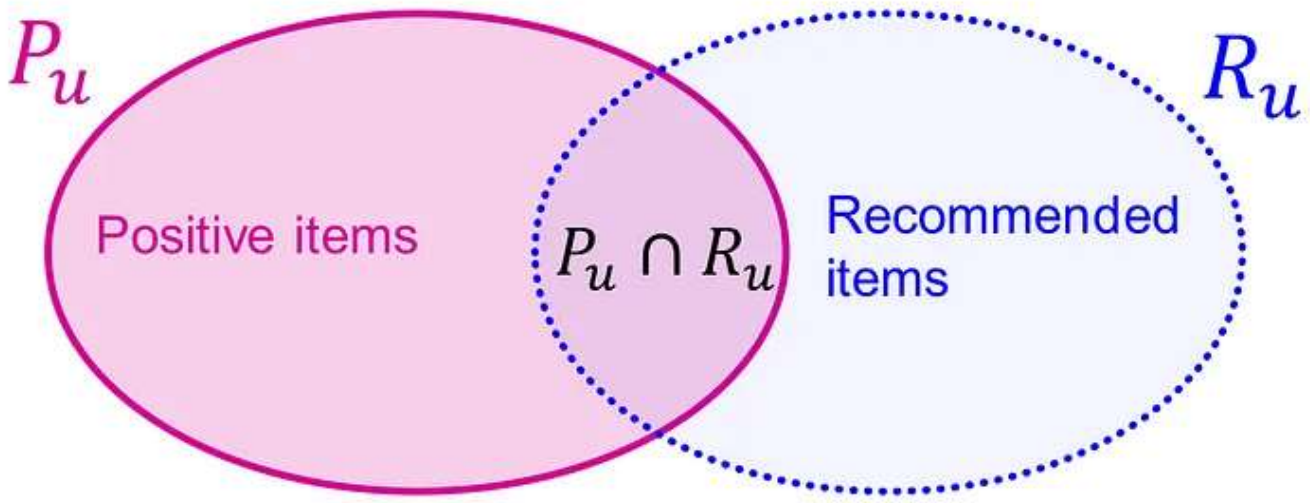
In the code snippet below, we walk through an implementation of the BPR loss:

```
def bpr_loss(users_emb, user_emb_0, pos_emb,
             pos_emb_0, neg_emb, neg_emb_0, lambda_val):
    """
    Calculate the Bayesian Personalzied Ranking loss.

    Parameters:
    users_emb (torch.Tensor): The final output of user embedding
    user_emb_0 (torch.Tensor): The initial user embedding
    pos_emb (torch.Tensor):  The final positive item embedding
    pos_emb_0 (torch.Tensor): The initial item embedding
    neg_emb (torch.Tensor): The final negtive item embedding
    neg_emb_0 (torch.Tensor): The inital negtive item embedding
    lambda_val (float): L2 regulatization strength

    Returns:
    loss (float): The BPR loss
    """
    pos_scores = torch.sum(users_emb * pos_emb, dim=1)
    neg_scores = torch.sum(users_emb * neg_emb, dim=1)
    losses = -torch.log(torch.sigmoid(pos_scores - neg_scores))
    loss = torch.mean(losses) + lambda_val * \
    (torch.norm(users_emb_0) + torch.norm(pos_emb_0) + torch.norm(neg_emb_0))

    return loss
```

Evaluation Metrics

The evaluation metric we will use is **Recall@K**, which is defined as the proportion of relevant items that are recommended to a user among the top-K items recommended by the algorithm[4].

For each user *u*, let P_u be a set of positive items the user will interact with in the future. Let R_u be a set of items recommended by the model, in the top-K recommendation, |R_u| = K. Items that users have already interacted with are excluded.



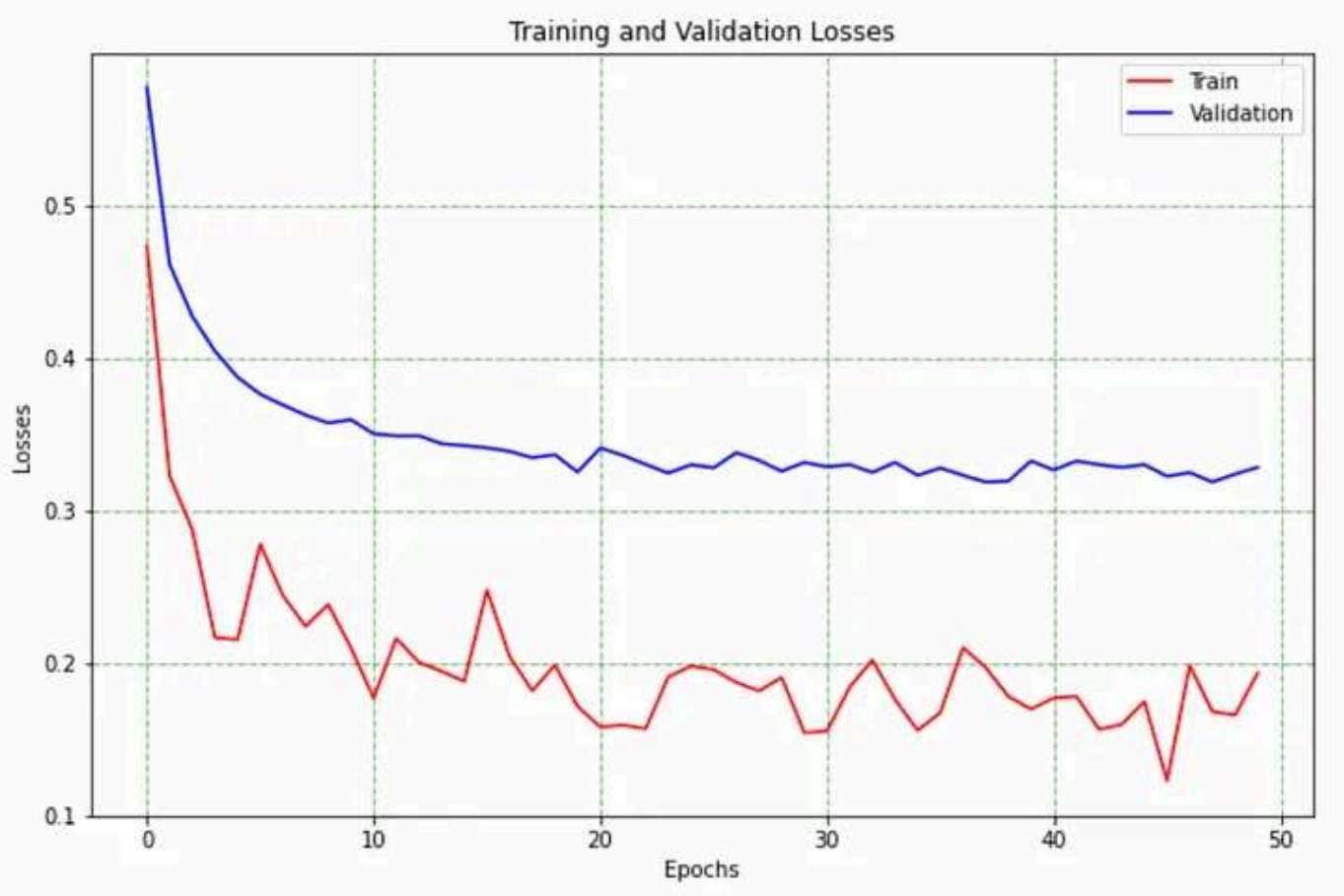
Recall@K for user *u* is |P_u ∩ R_u| / |P_u|. The final Recall@K is computed by averaging the recall values across all users.

Training the Model

The training configuration for this model is as follows:

```
# model configurations
config = {
    'batch_size': 256,
    'num_epoch': 50,
    'epoch_size': 200,
    'lr': 1e-3,
    'lr_decay': 0.9,
    'topK': 20,
    'lambda': 1e-6,
    'hidden_dim': 32,
    'num_layer': 3,
}
```

In the graph below, we have shown the losses on training and validation set in each epoch:



Here is the model performance on the test set:

Train Loss: *0.3142*

Recall@K: *0.1344*

Precision@K: *0.0348*

Make Recommendations for New Users

Are you tired of endlessly scrolling through Netflix, only to find yourself re-watching the same old movies? Do you wish you could discover new and exciting films that are tailored specifically to your tastes? Well, my friend, you're in luck! Come and try out our new recommender systems! 😎

Photo by Tyson Moultrie on Unsplash

In this section, we'll be putting our recommender model to the test to help you find the perfect movie for your next movie night. Using the power of the LightGCN model and user data, we'll suggest new and exciting films that are sure to satisfy your cinematic cravings. So sit back, relax, and get ready to discover your new favorite movie!

Let's say our friend Charlie is a huge Marvel fan, and he also has a fondness for mystery, sci-fi, and thriller movies. Below is his top10 movie list:

Title	Genres
The Matrix (1999)	Action|Sci-Fi|Thriller
The Shawshank Redemption (1994)	Crime|Drama
Fight Club (1999)	Action|Crime|Drama|Thriller
The Silence of the Lambs (1991)	Crime|Horror|Thriller
Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller
Guardians of the Galaxy (2014)	Action|Adventure|Sci-Fi
Seven (a.k.a Se7en) (1995)	Mystery|Thriller
The Avengers (2012)	Action|Adventure|Sci-Fi|IMAX
Inception (2010)	Action|Crime|Drama|Mystery|Sci-Fi|Thriller
The Prestige (2006)	Drama|Mystery|Sci-Fi|Thriller

Top10 Movies for Charlie

Let's further take a look at what movies our recommendation system would pick for him:

Title	Genres
Hush (1998)	Thriller
Forrest Gump (1994)	Comedy|Drama|Romance|War
Rosemary's Baby (1968)	Drama|Horror|Thriller
Amen. (2002)	Drama
Donnie Brasco (1997)	Crime|Drama
There Will Be Blood (2007)	Drama|Western
The Dark Knight (2008)	Action|Crime|Drama|IMAX
The Lord of the Rings (2002)	Adventure|Fantasy
Ocean's Eleven (2001)	Crime|Thriller
The Truman Show (1998)	Comedy|Drama|Sci-Fi

Top10 Recommendations for Charlie

Charlie must be happy about a bunch of new thrillers and sci-fi movies! And there is also another superhero movie for him to watch: The Dark Knight🦇🦇🦇

References

[1] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020). LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. *ArXiv*. /abs/2002.02126

[2] Tutorial: Link Prediction on MovieLens, PyTorch Geometric

[3] Ding, Z., Quan, W., He, X., Li, Y., & Jin, D. (2019). Reinforced Negative Sampling for Recommendation with Exposure Data. In IJCAI. 2230–2236.

[4] Leskovec, J., (2022). GNNs for Recommender Systems, CS224W Machine Learning with Graphs

Graph Neural Networks | Recommendation System

Written by Marco Zhao

Follow

3 Followers · Writer for Stanford CS224W: Machine Learning with Graphs