Final Project Report

CSCI 5525 : Advanced Machine Learning

Submitted By :
Alex Besch (besch040@umn.edu)
Mohit Yadav (yadav171@umn.edu)
Nirshal Chandra Sekar (chand861@umn.edu)
Varaha N.S. Menga (menga026@umn.edu)

Movie Recommendation system using GCNs

1. Introduction

Recommender systems are software designed to recommend items or services to individuals based on their past interactions and information about the product. Movie recommendation, as a subset of recommendation systems, is specifically tailored to suggest movies to users that they might enjoy. With the proliferation of streaming services like Netflix and Amazon Prime offering a plethora of items to recommend, it has become crucial to have robust recommender systems capable of keeping users satisfied and sparing them the hassle of sifting through lengthy lists of potentially irrelevant items[9,10]. Recommender systems address this precise challenge. While the field has been explored for some time, recent advancements in machine learning, deep learning, and the availability of data on unprecedented scales have accelerated progress in recommender systems.

In October 2006, Netflix released a dataset containing 100 million anonymous movie ratings and issued a challenge to the data mining, machine learning, and computer science communities to develop systems capable of outperforming its recommendation system, Cinematch[1]. This initiative sparked a race to build increasingly superior recommendation systems. Over time, the field has evolved, yielding new and improved systems annually. With industry giants such as Netflix, Amazon, and YouTube all grappling with the same dilemma—what movie, product, or video to suggest—there is significant impetus from the industry to drive research in this field. In this project, we implemented the LightGCN model introduced by Xiangnan He et al.[2], and drew implementation inspiration from Marco Zhao [3].

The most common approach is to learn embeddings for users and items, and then make recommendations based on the embedding vector[11]. Matrix factorization is a ealy model which did this[12]. Usage of Graph Convolution Network(GCN) for recommendation systems achieved state-of-the-art performance for the task[13]. Despite their success they are rather heavy and burdensome and many operations are directly inherited from GCN without justification[3]. This was the motivation to propose LightGCN for the task which included the most essential component of a GCN i.e. neighbor aggregation. LightCGN utilizes the user-item interaction graph to enhance these embeddings through propagation. Then the embeddings are combined using weighted average to get the final embeddings. This makes LightCGN simpler than GCN.

2. Problem Formulation

While Generative Pre-trained Transformers (GPTs) and deep learning algorithms are currently receiving significant attention in the field of artificial intelligence and machine

learning, one of the challenges we face in this project is to develop an accurate yet efficient recommender algorithm. Traditional algorithms like K-Nearest Neighbors (KNN) offer a simple approach to recommendation: when a user requests a movie recommendation, the algorithm identifies a subset of users most similar to the given user and suggests movies that this subset has enjoyed. Although conceptually straightforward, the computational complexity of this method increases drastically as the dataset size grows, making it impractical for large-scale applications.

To address this scalability issue, we propose the use of Graph Convolutional Networks (GCNs), which can leverage the inherent graph structure of recommender systems to make accurate predictions efficiently. In our project, we compare two different algorithms for movie recommendation: a KNN-based model and a GCN-based model. The KNN model represents the traditional approach, while the GCN model leverages graph neural networks to capture the complex relationships between users and items, potentially offering better performance and scalability.

3. Dataset

For this project we use the smaller version of MovieLens dataset[3] which has 100,000 ratings with 3,600 tag applications applied to 9,000 movies by 600 users. The dataset was created and maintained by GroupLens research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities which specializes in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.

The dataset contains two tables which are used in this project, namely movies data(see Fig. 1) and rating data (see Fig. 2). Movies table contains information about the movies i.e., movie name and genres along with an ID. Whereas the ratings table stores ratings by each user for a particular movie along with a timestamp for when the rating was recorded . Ratings are on a scale of 1 to 5 with 1 being the lowest and 5 being the highest user satisfaction. Users and movies are identified using a movieID and a userID. Timestamp is stored as unix time in seconds. The smaller version of the dataset was used to make the code manageable in the amount of resources and computing time that will be required.

```
movies_df
✓ 0.0s
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

9742 rows × 3 columns

Figure 1. Representative data from Movies Table in Movielens

```
rating_df
✓ 0.0s
```

	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
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

Figure 2. Representative data from Ratings Table in Movielens

4. Pytorch Geometric:

Since our project involves utilizing graph objects, PyTorch Geometric (PyG)[5] proves invaluable for managing graph data structures. PyG, a Python library designed for handling irregularly structured data like graphs, leverages PyTorch, a widely-used deep learning framework. It furnishes an array of resources and functionalities tailored for constructing and training neural networks that operate on graph-based data. Within PyG, one can find diverse architectures for graph neural networks (GNNs), optimization algorithms, and utilities for data manipulation. This amalgamation of features renders PyG a robust tool applicable to an extensive array of graph-centric tasks, including but not limited to node classification, link prediction, and graph classification.

5. Approaches

5.1. K-Nearest Neighbors:

The K-Nearest Neighbor algorithm is a simple non-parametric, instance-based learning algorithm that makes the predictions based on the similarity between the new data point and the existing data points in the dataset. For the Movie recommendation system our KNN model works as follows:

Consider the following table:

Sl/No.	User's Ratings	Movie Genre: [Comedy, Romance, Action]
1	1	[1, 1, 0]
2	3	[1, 0, 0]
3	5	[0, 1, 1]
4	4	[1, 1, 1]

Where, there are four movies, the user's ratings to each of these movies, and the "Movie Genre" vector says which genre that each of these movies belongs to.

First the movie genre vector is normalized. In this KNN model for movie recommendation, we then filter out the movies that the user has rated below a certain threshold, say 4/5. This is because we assume that the user likes the genres associated with the movies they have rated highly. So, in our example we will consider only the movies from rows 3 and 4, since the user has rated them 5 and 4, respectively.

Next, we calculate the average of the genre vectors for these highly rated movies, which would be ([0,1/2,1/2]+[1/3,1/3,1/3])/2 = [1/6,5/12,5/12]. This average genre vector represents the user's preferred genres based on their highly rated movies.

Finally, we find the K nearest movies in our dataset based on the similarity (or distance) between their genre vectors in the dataset and the user's average genre vector. These K most similar movies that the user has not reviewed are then recommended to the user.

For the working model, we have twenty genres - actions, adventure, animation, children, comedy, crime, documentary, drama, fantasy, film-noir, horror, imax, musical, mystery, romance, sci-fi, thriller, war, and western. The model was able to accurately predict 5 movies to any user. If the user did not have a sufficient amount of ratings above the threshold, the model recommended movies at random. Recommending movies to a new user, no matter how aggressive the algorithm is still a very difficult task to overcome.

This model represents a working implementation of a recommender algorithm but has a couple of issues. Since the data points are normalized, movies that conform to more than one genre are likely to be recommended more often than movies strictly adhering to fewer genres. The normalized movie vector will become closer and closer to [0.05, …0.05] as it is classified under more genres. Users will generally watch and enjoy movies from more than one genre, say romance, comedy, action, war depending upon a variety of factors. They may have a higher bias toward certain categories, but it was noticed that people liked a variety of movie genres. Because of this, they tended to be recommended movies that fit more genres. In the real world this would not be a significant issue, however the recommended movies tended to be popular and likely already seen. When filtering out movies the user had not already watched, the recommender algorithm was nearly working at random. Overall the KNN implementation was a great starting point for a proof of concept and to understand the data, but a better implementation was needed.

5.2. Graph Neural Networks

A recommender system can be most intuitively modeled as a bipartite graph with two types of nodes namely users and items (or movies for our movie recommender) and positive interaction(higher rating) are represented as edges between these two types of nodes. A bipartite graph is a graph where the vertices can be divided into two disjoint sets such that all edges connect a vertex in one set to a vertex in another set. There are no edges between vertices in the disjoint sets[4]. As the data can be modeled as a graph it is possible to apply graph machine learning techniques to predict the link between two nodes. The goal of a recommender system is to utilize this past user-item interaction data to predict new items that each user will likely engage with in the future, improving overall user satisfaction and engagement.

5.2.1 Graph Convolution Network (GCN):

Graph Convolutional Networks (GCNs) are a type of neural network designed to operate on graph-structured data. They work by applying convolutional operations on the nodes of the graph, allowing the model to capture the relationships and dependencies between nodes. In a recommender system, users and items can be represented as nodes, and their interactions (e.g., ratings) as edges in the graph.

However, traditional GCNs can be computationally expensive for large-scale graphs, as they need to compute the convolution operation for every node in the graph.

5.2.3 Light GCN

The primary objective of our project is to provide personalized recommendations by exploiting the structure of the user-item interaction graph. Unlike other graph-based recommendation methods that use complex GNNs, LightGCN, introduced by Xiangnan He et al.[2] simplifies the architecture by omitting feature transformation and non-linearity operations, resulting in fewer parameters and faster training times while maintaining competitive recommendation quality.

The LightGCN model involves performing several layers of graph convolutions to propagate user and item embeddings. These embeddings are then used to generate personalized recommendations through a decoder component. The key components of LightGCN are:

1. Light Graph Convolution: This is the core operation of LightGCN, where user and item embeddings are propagated through multiple layers of graph convolutions using a specific propagation rule.
2. Encoder and Decoder: The initial user and item embeddings are the only adjustable parameters in LightGCN. The final embeddings are obtained by combining the embeddings from each propagation layer using a weighted average. The model prediction is then calculated by taking the inner product of the final user and item embeddings.

6. Methodology:

We first start by processing the data to construct a connected graph object which can be passed to the PyG library. We use the data with rating above and equal to 4 as a positive interaction i.e., a metric that a person actually likes that movie. We create the edge_index pytorch tensor of shape (2, num_of_ratings), where each column has a specific rating, with entries in the first row being userID and second row being movieID. The edge index is then divided into training, validation and testing datasets in 80/10/10 proportion randomly. These now contain part of the edge index data.

We implement a LightGCN with three propagation layers. The user and movie embedding were taken of size 24 and were initialized with normal xavier initialization. These parameters were tweaked after experimenting with the model.

$$\sigma = \sqrt{\frac{2}{n_{inputs} + n_{outputs}}}$$

Where:

- σ is the standard deviation for normal Xavier initialization.
- n_{inputs} is the number of inputs in the input layer.
- $n_{outputs}$ is the number of outputs in the output layer.

LightCGN uses the embedding in all the layers and not just the last layer, on a simulator line we used the weighted average of layer embeddings with weights as 0.3, 0.3, 0.4.

For the loss function we used the Bayesian Personalized Ranking (BPR) loss function [4], which is a pairwise ranking loss that compares positive (observed) and negative (unobserved) user-item interactions. Observed interactions are the user-item pairs where the user had interacted with or rated the item, such as user rating a movie. For these pairs, the model aims to predict a higher score, as they represent the user's positive preference. Likewise, unobserved interactions are the user-item pairs that the user has not interacted with or rated that item. For these pairs, the model aims to predict a lower score, as they are treated as negative examples or items the user is not interested in. The BPR loss function compares the predicted scores for a positive sample and a negative sample for a given user. The loss function aims to maximize the difference between the scores for the positive and negative samples, essentially learning to rank the observed (positive) interactions higher than the unobserved (negative) interactions.

The loss function equation is shown below:

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

Where,

- M is the number of users,
- Nu is the observed items,
- $\hat{y}_{ui}$ is the predicted score for the observed items,
- $\hat{y}_{uj}$ is the predicted score for the unobserved items,
- $\lambda\left|\left|E^{(0)}\right|\right|^2$ is the regularization term.

By minimizing this loss, the LightGCN model learns to assign higher scores to observed (positive) user-item interactions and lower scores to unobserved (negative) interactions, enabling personalized item ranking and recommendation for each user.

The model was trained with an adam optimization algorithm[8] using 1e-3 as learning rate. We used an exponential learning rate scheduler with a decay rate of 0.9.

A recommender model cannot be trained and tested like a normal machine learning classification algorithm. Since the movie is recommending items, there is not a clear way to classify the recommendation as accurate or not. To evaluate the LightGCN recommender model, the Recall@K function was used along with a precision function.

Recall@K measures the proportion of top-K recommendations that the user has already enjoyed. Recall@K receives the edge_index_data and K value as a parameter. The *edge_index_data* is fed into the recommender model, and a list of movie recommendations is returned ($Y_{user,\ movie}$), each ranked with a higher or lower recommendation. The *torch.topk()* function is then used to separate a subset of the highest recommended movies. Next, the recommended movies are compared with a list of *positive_items* which are movies the user has already rated highly ($X_{user,\ movies}$). The Recall@K function then finds the number of movies that are both recommended by the model, and the user has rated highly. The figure 3 below shows an illustration of this evaluation metric.

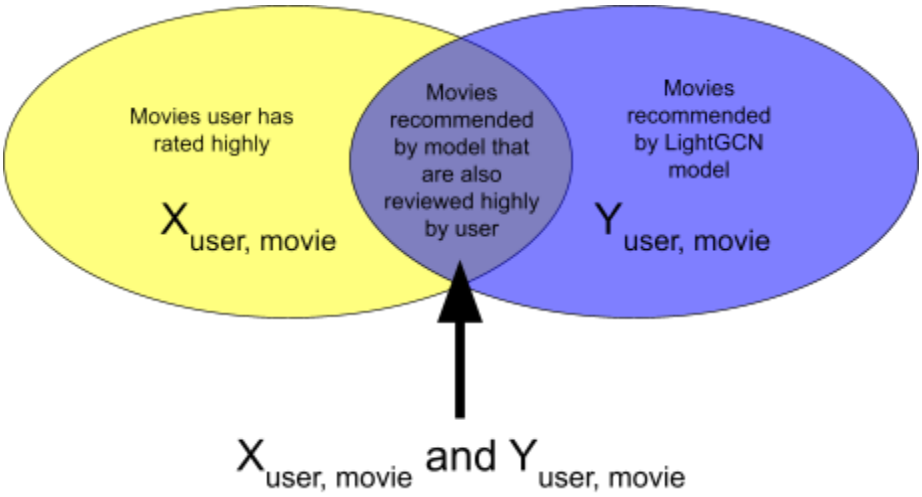

Figure 3: Graphical representation of how Recall@K is implemented.

The final equation Recall@K uses to evaluate the accuracy of the model is:

$$Recall@K = \frac{length(X_{user,movie} \cap Y_{user,movie})}{length(X_{user,movie})}$$

The precision of the model is found in a very similar manner to recall. The precision measures the proportion of model predictions that are both liked by the user, and recommended by the model. The formula for precision is shown below, where K is the parameter passed to *torch.topk()* to return the best K movie recommendations:

$$Precision@K = \frac{length(X_{user,movie} \cap Y_{user,movie})}{K}$$

7. Results:

The model was trained for 50 epochs with a batch size of 256. The training and validation losses are shown in Figure 4.

Figure 4: Training and validation losses achieved during testing.

We can see from Figure 4 that the model was successfully trained since the losses decreased during training. The validation loss gets stagnant after 40 epochs. However, a decreasing loss over time does not indicate that the model is performing particularly well. Upon testing the outputted model, the following figure is created.

Test Loss was : 0.27824077010154724

Metric	Train	Val	Test
Recall@20	0.122715	0.109944	0.105631
Precision@20	0.222085	0.0295455	0.0288288
Recall@50	0.218625	0.20678	0.189869
Precision@50	0.176617	0.0224825	0.0225946

Figure 5: Model performance data

Figure 5 shows the resulting model achieved a recall@20 of 0.12 while this is not comparable to the top model available, it still performs decently given the small dataset and limited computing resources.

8. Conclusion and Future Work:

Our implementation of LightCGN was able to learn about the complex nature of user-item interaction, but there is a lot of room for improvement. Future work for the project could include implementing a model on the full-scale Movielens dataset, with a more complex model to capture more intricate information within the dataset. Additionally, this implementation doesn't consider auxiliary information that, in hindsight, would make sense to include in the model. This could encompass information such as the lead actors in the movie, details about the director, release year, and genre. Moreover, the public's liking or disliking of a movie has a temporal influence as well; for instance, 'The Shawshank Redemption' was not initially a hit upon release but has since maintained the top-rated spot on IMdB. However, the temporal component was not considered in this model.

Despite the discussed drawbacks, LightCGN presented a novel approach to creating a recommender algorithm that could glean much more insight from the dataset compared to a simple KNN. In this project, we implemented both a simple KNN and LightCGN to provide recommendations for users. While the KNN model represents a basic process for creating a recommender, it lacks the complexity needed to fully comprehend the intricate interactions within this type of dataset. LightCGN, on the other hand, has the potential to glean much more information from the dataset edges and become a significantly improved model for the application. Furthermore, it accomplishes this while remaining a relatively lightweight model considering the complexity of GCNs.

References:

1. Bennett, J., Lanning, S., Netflix, & Netflix. (2006). The Netflix Prize. In Netflix. Netflix. https://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/The-Netflix-Prize-Bennett.pdf
2. He, X., Deng, K., Wang, X., Li, Y., Zhang, Y. and Wang, M., 2020, July. Lightgcn: Simplifying and powering graph convolutional network for recommendation. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval (pp. 639-648).
3. Zhao, M. (2023). Movie Recommender Systems with PyG. [online] Stanford CS224W GraphML Tutorials. Available at: https://medium.com/stanford-cs224w/movie-recommender-systems-with-pyg-37da71f405a4 [Accessed 10 May 2024].
4. Rendle, S., Freudenthaler, C., Gantner, Z. and Schmidt-Thieme, L., 2012. BPR: Bayesian personalized ranking from implicit feedback. arXiv preprint arXiv:1205.2618.
5. Constales, D., Yablonsky, G.S., D'hooge, D.R., Thybaut, J.W. and Marin, G.B., 2016. Advanced data analysis and modeling in chemical engineering. Elsevier.
6. https://www.pyg.org/
7. "MovieLens." GroupLens, 26 Apr. 2019, grouplens.org/datasets/movielens/.
8. Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
9. Covington, P., Adams, J. and Sargin, E., 2016, September. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems (pp. 191-198).
10. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L. and Leskovec, J., 2018, July. Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 974-983).
11. Cheng, Z., Ding, Y., Zhu, L. and Kankanhalli, M., 2018, April. Aspect-aware latent factor model: Rating prediction with ratings and reviews. In Proceedings of the 2018 world wide web conference (pp. 639-648).
12. Koren, Y., Bell, R. and Volinsky, C., 2009. Matrix factorization techniques for recommender systems. Computer, 42(8), pp.30-37.
13. Wang, X., He, X., Wang, M., Feng, F. and Chua, T.S., 2019, July. Neural graph collaborative filtering. In Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval (pp. 165-174).