Project Final Report

Serena Ulammandakh, Eric Sun
Yale Univeristy
CPSC 483
tselmeg.ulammandakh@yale.edu, eric.sun.ecs92@yale.edu

1 Introduction

In the digital landscape, recommendation systems are pivotal in guiding users through a vast sea of choices, enhancing user experience through personalized content. This personalization is crucial for user engagement and targeted marketing. Our project delves into these systems, emphasizing precision and user-centric personalization.

Traditional recommender systems, often reliant on basic collaborative or content-based filtering, struggle with complex feature dynamics and face escalating computational costs with increasing data volume. Our project introduces a sophisticated approach to movie recommendation, integrating a broad spectrum of features to unravel the intricate relationships between users and movies and employing biased random-walks for computational efficiency.

Utilizing the MovieLens dataset, with 100K ratings from 1K users on 1.7K movies, we extend beyond standard collaborative filtering. Our heterogenous graph structure incorporates user nodes based on ratings and personal attributes, and movie nodes characterized by ratings, genres, genome-tags, and titles. Initially, we enriched our model with IMDb dataset features like cast and relational movie data but faced computational challenges when scaling to the larger MovieLens-25m dataset.

To overcome this, we developed two novel learning methods. The first, inspired by the PinSage Graph Neural Network, incorporates a modified random-walk algorithm and advanced preprocessing of textual data. The second approach leverages the node2vec algorithm to refine the similarity metrics between movies and users within our graphs, adding bias to the random-walks to focus on immediate and secondary neighbors.

Our project aspires to push the boundaries in movie recommender systems, combining a richer feature set with advanced graph neural networks. We aim not just to enhance computational efficiency but also to provide a novel framework for probing deep into the complexities of user-item relationships in a condensed feature space, thus redefining the standards for personalized content recommendation.

2 Related Works

2.1 Traditional Methods

Earlier recommender systems predominantly utilized collaborative filtering, which capitalized on user-item co-occurrences to predict preferences. Despite its straightforward approach and effectiveness with ample data, this method grapples with the cold-start problem for new users or items. Markov chain-based methods, notably the Factorized Personalized Markov Chains (FPMC), emerged to better capture sequential user behavior, improving next-item recommendations. These models, however, are limited in grasping longer-term behavioral patterns and face scalability issues in sparse data environments.

2.2 Deep Learning-based Methods

The integration of deep learning into recommender systems has been a significant step forward. Methods like collaborative filtering, despite being popular, suffer from issues like matrix sparsity and scalability. Deep learning techniques, applied successfully in fields like natural language processing and image classification, have been adapted to recommender systems for more precise representations. Examples include DLCRS, which combines deep learning with collaborative filtering, providing enhanced accuracy but often overlooking the structural nuances of user-item interactions.

2.3 Content-based/Collaborative Filtering combined GNN

A recent trend in hybrid recommender systems is the application of graph neural networks (GNNs) to marry content-based and collaborative filtering methods. Ying et al. (2019) introduced a GNN model that effectively navigates

user-item interaction patterns while integrating content features. Similarly, Fan et al. (2020) developed a model that combines semantic content with user-item interaction graphs using GNNs. These approaches significantly improve handling of sparse data, a common limitation in traditional collaborative systems, but face computational complexity challenges.

2.4 PinSage and Random-Walk Algorithms

Innovations like PinSage and random walk methodologies have marked notable advancements in recommender systems. PinSage, which utilizes random walks and graph convolutional networks, adeptly manages complex item graphs and surpasses traditional collaborative filtering in capturing intricate interactions. Random walk techniques, such as node2vec, provide a diverse and contextually relevant exploration of item networks. These methods address scalability and complexity of interaction patterns but require careful parameter tuning and computational resources, particularly for real-time applications.

3 Method

3.1 Method 1: PinSage Modification Implementation

Our first approach to building a recommender system involves constructing a heterogeneous graph from the movieLens-1m dataset and IMDb dataset, utilizing PinSage's Graph Neural Network model, and leveraging libraries such as DGL (Deep Graph Library) and Torchtext to make necessary modifications.

3.1.1 Graph Construction

We created our heterogeneous graph using a custom class, PandasGraphBuilder, which creates a heterogeneous graph from multiple different pandas dataframes (imported movieLens and IMDb datasets). This process involves adding entities (users and movies) and binary relations (e.g., user-movie interactions) to the graph. For instance, the graph connects users to movies they have watched, forming a bipartite structure. Each node and edge in the graph are assigned features based on the dataframe columns, with categorical and numerical data handled appropriately.

3.1.2 Feature Processing

The main implementation changes we made to PinSage are to how the features of nodes (users and movies) are processed and converted from pandas series to PyTorch tensors, facilitating the integration of tabular data with textual features. We obtained our textual features for movies (titles, tags, etc) from IMDb dataset and created a new custom dataset class, CustomDataset, which tokenizes and numericalizes this text using pre-defined vocabularies and embeddings. We did this by utilizing FastText's pre-trained embeddings, and to address Out-of-Vocabulary words, we assigning unique vectors for unknown terms and zero vectors for those not in the FastText vocabulary.

We believed that this enhanced text processing approach would integrate more complex item textual features and offer a more nuanced and accurate recommendation system than the original PinSAGE's, which focused mainly on structural graph features.

3.1.3 Model Architecture

The core of our recommender system, the custom PinSAGEModel, is an intricate fusion of a LinearProjector and a GraphSAGE neural network, SAGENet. The LinearProjector efficiently handles the initial transformation of diverse node features, including both categorical and numerical data, and is particularly adept at processing textual information using Bag-of-Words with pre-trained embeddings. Following this, SAGENet, structured with multiple layers of WeightedSAGEConv, takes over to learn deep node embeddings. These layers aggregate neighboring information, taking into account edge weights, which enables the model to capture the varying intensities of user-item interactions. The embeddings learned through SAGENet are then utilized by the ItemToItemScorer module, which computes item-to-item similarity scores.

3.1.4 Sampling and Training

To train the model, we employ a neighbor sampling technique (NeighborSampler) and an item-to-item batch sampler (ItemToItemBatchSampler). The NeighborSampler class, which is used to sample neighborhood nodes, is inspired by the PinSAGE algorithm. PinSAGE itself is an extension of random walk techniques, combining them with graph convolutional networks. While this method does not use a traditional random walk, this method does utilize random walks for sampling neighbors in the graph. The ItemToItemBatchSampler class is used for generating batches of item pairs (positive and negative samples) for training the model. This sampler selects random nodes (items) and then performs sampling to generate item pairs. Training involves optimizing the model to predict user-item interactions, with a loss function computed based on the difference in scores between observed and negative interactions.

3.1.5 Testing and Evaluation

The primary metric of evaluation is precision at K (HIT@10), which assesses the accuracy of the top-K recommendations. The evaluation process, involves generating recommendations using the LatestNNRecommender class, which prioritizes users' most recent interactions. Recommendations are made by ranking items based on their interaction scores, and the system's effectiveness is measured by comparing these top-K recommendations against a validation set. We ran our training/testing steps three separate times to create different precision scores and calculated the std error for our results.

3.2 Method 2: Contextualized Embedding-based Recommendation via Skip-Gram Graph Neural Network

For this method, our recommender system's methodology is inspired by an existing GitHub project that implemented a graph-based recommendation system using TensorFlow. We adapted and extended this approach using PyTorch, introducing significant adaptations and enhancements along with expanded dataset compatibility and improved embedding processes.

3.2.1 Data Preparation

Initially, the data is prepared by loading movie information and user ratings into pandas DataFrames, a process similar to the initial GitHub project. Unique identifiers are assigned to each movie and rating, and ratings are converted to floating point values.

3.2.2 Graph Construction

We construct an undirected graph where nodes represent movies. Edges between movies are created based on user ratings, in such a way that an edge is formed between two movies if both are rated above the minimum threshold we set by the same user. subject to certain conditions. Edge weights are calculated based on the pointwise mutual information (PMI) between movies, considering the frequency of co-occurences in user ratings.

$$\texttt{PMI}(x, y) = \log(\texttt{xy}) - \log(x) - \log(y) + \log(D) \tag{1}$$

3.2.3 Random Walks

Adapting the concept of random walks from the original GitHub project, random walks are performed on the graph to generate sequences of movies. These walks are biased by parameters controlling the likelihood of revisiting nodes and exploring local versus distant neighborhoods. The return parameter p influences the probability of revisiting the immediate previous node. The in-out parameter q differentiates between inward and outward nodes in the walk.

The `next_step` function calculates the next node in a walk by adjusting the probabilities of moving to neighboring nodes, taking into account the edge weights and parameters p and q. The `random_walk` function conducts multiple walks over the graph, leveraging the `next_step` function to create sequences of movies. Before the training could start, the most crucial part of this methodology centered around finding the right values to create a representative sequence, making sure to not explore too far or not linger in the node's local area. We aimed to captured the structural essence of the graph by finetuning these parameters and analyzing the number of edges and nodes to the average number of degrees in the graph to ensure a comprehensive graph without loss of data.

3.2.4 Training Data Generation

We created training examples for the SkipGram model using sequences from the random walks. Employing a custom implementation of the `skipgram` function in PyTorch, we generated pairs of nodes with a balance of positive and negative samples. This approach diverges significantly from the Keras implementation used in the original project.

3.2.5 Model Training and Evaluation

Our project utilizes a PyTorch-based Skip-Gram neural network model. We defined a custom `SkipGramModel` class and a `train_and_evaluate` function to calculate average loss over multiple runs. These adaptations and optimizations of the model architecture and training process are key differentiators from the original TensorFlow implementation. The `SkipGramModel` class defines the neural network architecture, embedding layers, and forward pass logic. Training involves calculating regularized loss and optimizing the model over multiple epochs.

3.2.6 Embeddings and Recommendations

Post-training, we extract and normalize movie embeddings from the model, employing them for making movie recommendations. This step involves processing query movies and using cosine similarity to find similar movies, a technique that aligns with the foundational methodology but is enhanced in our implementation for greater flexibility and applicability to various datasets.

4 Experiments

4.1 Dataset Analysis

We tested our different methodologies on three datasets: the MovieLens 100K dataset, MovieLens 25M dataset, and the Book-Crossing dataset.

Table 1: MovieLens 100K

Statistic	Value
Number of movies	1682
Number of users	943
Number of ratings	100000

Table 2: MovieLens 25M

Statistic	Value
Number of movies	62423
Number of users	162541
Number of ratings	25000095

Table 3: Book Crossing

Statistic	Value
Number of books	271379
Number of users	105283
Number of ratings	1149780

Data visualizations for MovieLens 100K dataset: Data visualizations for MovieLens 25M dataset:

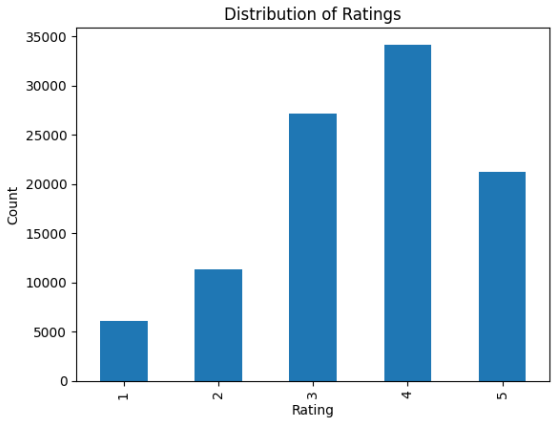

Figure 1: Distribution of Ratings

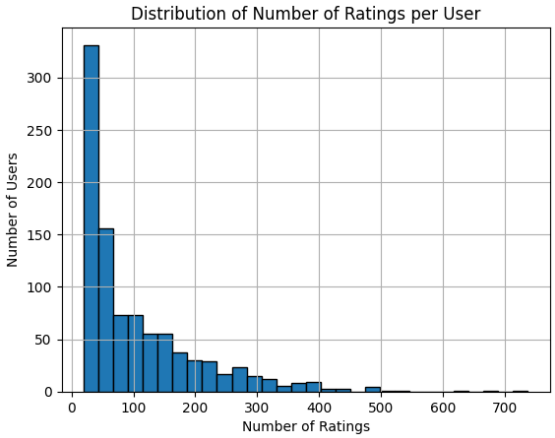

Figure 2: Distribution of Number of Ratings per User

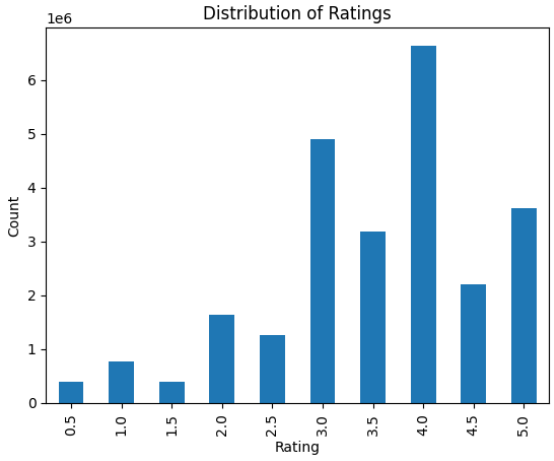

Figure 3: Distribution of Ratings

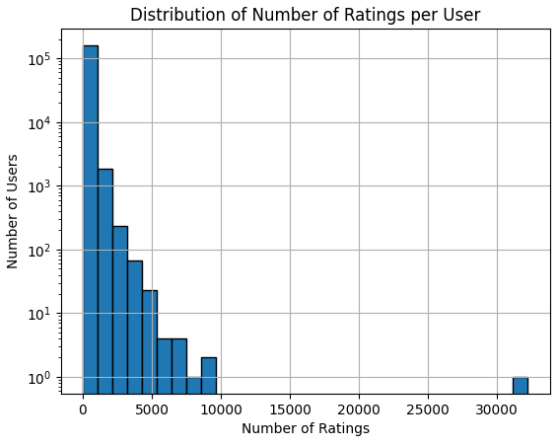

Figure 4: Distribution of Number of Ratings per User

Data visualizations for Book-Crossing dataset:

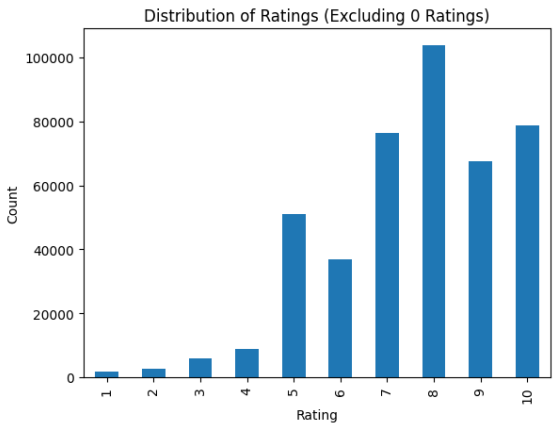

Figure 5: Distribution of Ratings

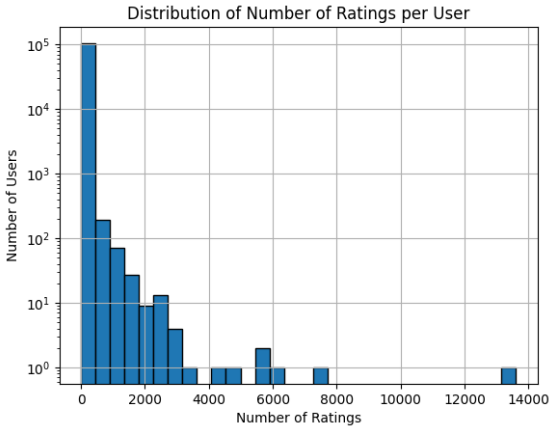

Figure 6: Distribution of Number of Ratings per User

4.2 Training Details and Data Preprocessing

4.2.1 Method 1

Table 4: Hyperparameters for Method 1

Hyperparameter	**Value**
random-walk-length	2
random-walk-restart-prob	0.5
num-random-walks	10
num-neighbors	3
num-layers	2
hidden-dims	16
batch-size	7
num-epochs	15
batches-per-epoch	20000
lr	3e-5
k	50/100

Table 5: Users Datset

user_id	gender	age	occupation	zip
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455

Table 6: Movies Dataset

movie_id	title	year	Children's	Animation	Comedy	Adventure	Fantasy
1	Toy Story	1995	True	True	True	False	False
2	Jumanji	1995	True	False	False	True	True
3	Grumpier Old Men	1995	False	False	True	False	False
4	Waiting to Exhale	1995	False	False	True	False	False
5	Father of the Bride Part II	1995	False	False	True	False	False

Table 7: Ratings Dataset

user_id	movie_id	rating	timestamp	train_mask	val_mask	test_mask
1	1193	5	978300760	True	False	False
1	661	3	978302109	True	False	False
1	914	3	978301968	True	False	False
1	3408	4	978300275	True	False	False
1	2355	5	978824291	True	False	False

Table 8: Graph Construction Summary

Property	Detail	Value
Number of Nodes	Movie User	3704 6040
Number of Edges	Movie to User User to Movie	988129 988129
Metagraph	-	[('movie', 'user', 'watched-by'), ('user', 'movie', 'watched')]
Val-Matrix	Type Size Stored Elements	Sparse Matrix 6040×3706 6040 (numpy.int64)
Test-Matrix	Type Size Stored Elements	Sparse Matrix 6040×3706 6040 (numpy.int64)

4.2.2 Method 2

Table 9: Hyperparameters for Method 2

Hyperparameter	Value
Embedding Dimension	50
Learning Rate	0.01
L2 Regularization (l2_reg)	1×10^{-6}
Number of Epochs	11
Batch Size	1024
Number of Walks (num_walks)	10
Walk Length (num_steps)	7
Random Walk Return Param (p)	1
Random Walk In-Out Param (q)	1
Window Size	7
Number of Negative Samples	4

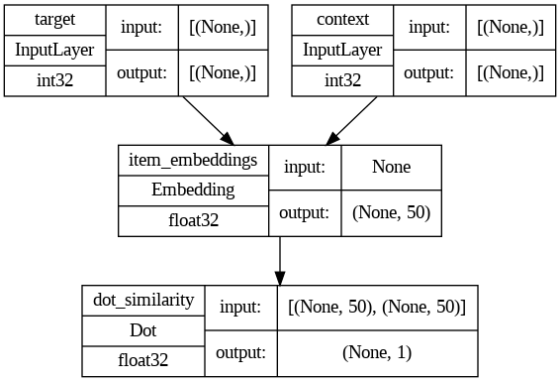

Figure 7: Model Plot from keras.io

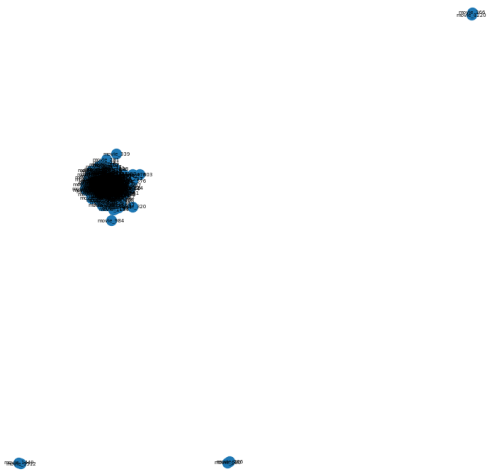

Figure 8: Graph Representation of MovieLens 100K

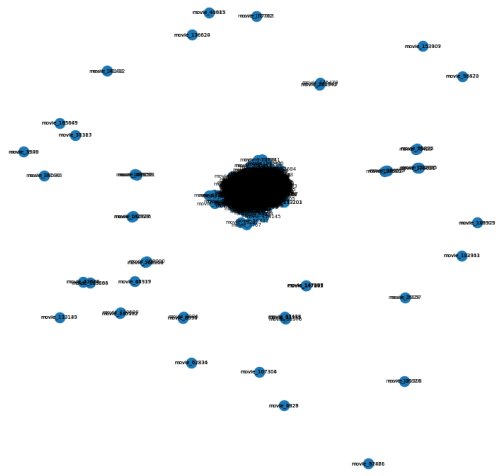

Figure 9: Graph Representation of MovieLens 25M

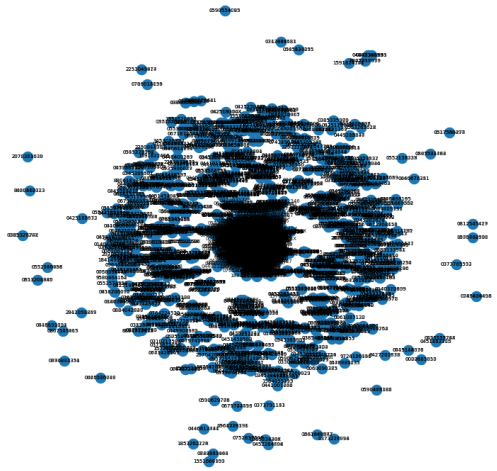

Figure 10: Graph Representation of Book-Crossing

4.3 Results

4.3.1 Method 1

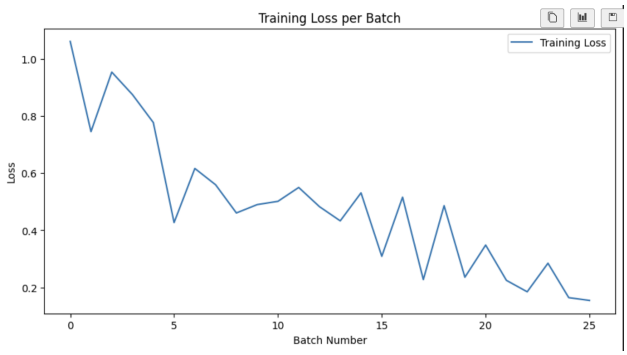

Figure 11: Training Loss Values

Run	1	2	3			
	HK@50 HK@100	HK@50 HK@100	HK@50 HK@100	Mean HK@50 Mean HK@100	Std. Error HK@50 Std. Error HK@100	Mean ± Std. Error HK@50 Mean ± Std. Error HK@100
Our Model	0.138 0.247	0.135 0.253	0.141 0.25	0.138 0.25	0.0017 0.0017	0.138 ± 0.0017 0.25 ± 0.0017

Table 10: Model performance across three runs with mean and standard error calculations.

After running the model 3 times, we compare its average results with the Baseline models: GraphSage, PinSage, and HS-GCN, which are current state-of-the-art models.

Dataset	**MovieLens**	
Evaluation Metric	**HR@50**	**HR@100**
GraphSage	0.2132	0.3326
PinSage	0.1587	0.2501
HS-GCN	0.2052	0.3169
Our Model	0.138	0.2533

Table 11: Comparison of model performance

4.3.2 Method 2

Figure 12: Book-Crossing, ML 100K, ML 25M: Losses Over 10 Runs

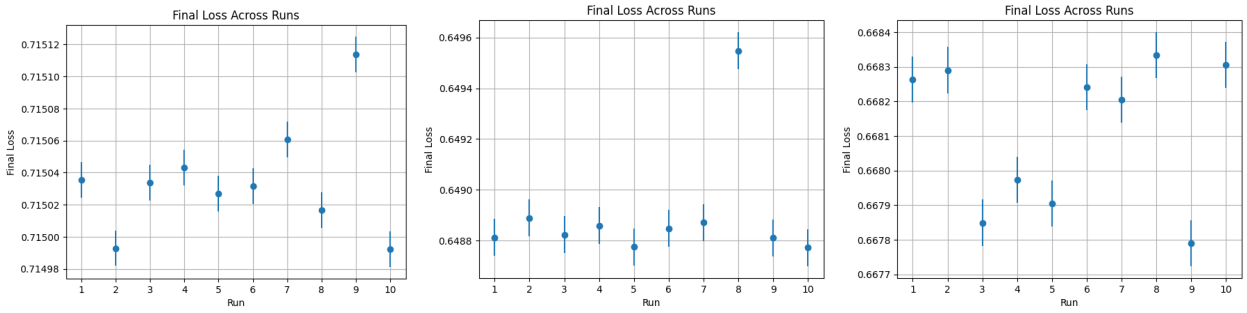

Table 12: Final Loss BC

Run	Final Loss
1	0.715035
2	0.714993
3	0.715034
4	0.715043
5	0.715027
6	0.715032
7	0.715061
8	0.715017
9	0.715114
10	0.714992
Mean	0.715035
Std. Error	0.000011

Table 13: Final Losses ML-25M

Run	Final Loss
1	0.648812
2	0.648889
3	0.648823
4	0.648858
5	0.648774
6	0.648847
7	0.648871
8	0.649548
9	0.648810
10	0.648772
Mean	0.648900
Std. Error	0.000073

Table 14: Final Losses ML-100K

Run	Final Loss
1	0.668263
2	0.668290
3	0.667849
4	0.667973
5	0.667905
6	0.668241
7	0.668204
8	0.668333
9	0.667790
10	0.668305
Mean	0.668115
Std. Error	0.000067

Dataset	ML-100K	ML-25M	BC
MG-GAT	0.890	0.732	0.734
GraphRec	0.904	0.843	0.730
GC-MC	0.910	0.832	-
Our Model	0.843	0.794	0.721

Table 15: Comparison of model performance

4.4 Final Analysis

4.5 Method 1

From the training loss curve, we can see that the model does not decrease steadily, but it does decrease eventually. We believe this is due to the model overfitting as you can see the loss curve jumped around a lot in the range (13-22 batches). There are several reasons as to why this might occur, but we believe the increased amount of complexity we injected into the nodes' feature vectors as well as the necessary fine-tuning of hyper-parameters are the main causes of this. In the future, we hope to tune the parameters by training with different embedding dimensions in each layer to see what might prevent overfitting of our model.

However, as shown from the results tables, our model performs just a bit worse than the original PinSage model for HR@50, but performs slightly better for HR@100. This indicates that our model may be more effective in capturing a broader range of user preferences, leading to more diverse recommendations. The improved performance in HR@100 suggests that while our model may not always rank the most relevant items at the very top, it successfully identifies relevant items within a larger set of top recommendations. This characteristic is particularly valuable in scenarios where providing a wider array of options is beneficial, and where the goal is to offer a diverse set of potentially interesting items to the users. Moreover, the slightly lower performance in HR@50 can be an indication of a trade-off between precision at the top of the recommendation list and the overall diversity and breadth of the recommendations.

4.6 Method 2

The graph generation process emerged as a pivotal component of our methodology. Ensuring the readiness of the data for processing, particularly by normalizing names and aligning indices, was a critical step. A significant portion of our efforts was dedicated to determining the optimal `min_weight`, which served as a threshold for establishing connections between two movies. Initially, a lower `min_weight` resulted in an excessively dense graph with numerous edges. However, after experimenting with various values, we settled on a significantly higher `min_weight` than what was used in the original GitHub implementation.

Another crucial aspect was tuning the parameters `p` and `q`, which initially led to the model recommending movies that were vastly different in nature. By adjusting the number of steps in the random walk and balancing exploration with exploitation, we were able to enhance the quality of recommendations, surpassing those of the original implementation. The loss plots revealed that, over time, the loss stabilized and eventually decreased. Notably, our implementation maintained loss values below 0.8, a significant improvement compared to the original code's average loss of around 2.

The initial movie embedding process played a vital role, especially towards the end of the training phase, where it was used to obtain embeddings of context movies for similarity calculations. The model demonstrated commendable performance on the MovieLens 25M dataset, attributable to the dataset's extensive size. This resulted in a similar loss performance compared to the original model.

However, the Book-Crossing dataset presented unique challenges. A substantial portion of the data had to be discarded due to numerous books lacking ratings or users not rating any books. Additionally, the absence of genres for the books meant the model couldn't leverage this potentially informative feature, leading to a slightly diminished performance compared to the MovieLens datasets. Despite these challenges, the model exhibited a consistent decrease in loss over epochs for each training run.

This model shows significant potential for further enhancement. Incorporating additional information such as movie genres, directors, book authors, and other relevant metadata could greatly refine the embedding process and the overall effectiveness of the recommendation system.

5 Conclusion

In this project, we advanced movie recommendation systems by integrating graph neural networks (GNNs) and innovative embedding techniques, moving beyond traditional collaborative and content-based methods. Our approach involved a customized PinSage model and a novel Skip-Gram Graph Neural Network, utilizing rich data from MovieLens and IMDb to explore the multifaceted relationships between users and movies.

Our modified PinSage model effectively handled the computational challenges of large datasets, while the Skip-Gram model excelled in capturing nuanced user-item interactions, enhancing recommendation relevance and accuracy. These methods not only improved computational efficiency but also enriched the user experience, offering more personalized, context-aware recommendations.

Acknowledging the computational intensity of GNN-based models, our future work will aim to optimize these systems further, focusing on balancing recommendation diversity and refining the interpretability of models. We also plan to extend our methodologies to other domains, exploring their adaptability and scalability.

Overall, our project contributes significantly to the field of movie recommender systems, demonstrating the potential of GNNs and embedding techniques in overcoming traditional method limitations. This work provides a foundation for future research in personalized recommendation systems, opening new avenues for exploring complex user-item relationships.

6 Reproducability

Github Repository: https://github.com/eric-sun92/CPSC-483-Final-Project/tree/main

References

References

[1] Wolkdevelopment. (2023, May 3). *Pinsage on MovieLens*. Kaggle. https://www.kaggle.com/code/wolkdevelopment/pinsage-on-movielens

[2] Keras.io. *Keras-Io/Examples/Graph/Node2vec_movielens.py at Master · Keras-Team/Keras-Io*. GitHub. https://github.com/keras-team/kerasio/blob/master/examples/graph/node2vec_movielens.py. Accessed 10 Nov. 2023.

[3] Maklin, Cory. (2022, Aug 11). *Word2Vec — Skip-Gram*. Medium. https://medium.com/@corymaklin/word2vec-skip-gram-904775613b4c. Accessed 13 Dec. 2023.

[4] Mikolov, Tomas, et al. *Distributed Representations of Words and Phrases and Their Compositionality*. https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[5] Mikolov, Tomas, et al. *Efficient Estimation of Word Representations in Vector Space*. (2013). https://arxiv.org/pdf/1301.3781.pdf.

[6] *MovieLens*. GroupLens. (2019, Apr 26). https://grouplens.org/datasets/movielens/.

[7] Seidakhmetov, Tamirlan. (2022, Feb 9). *Graph Neural Network Based Movie Recommender System*. Stanford CS224W GraphML Tutorials. Medium. https://medium.com/stanford-cs224w/graph-neural-network-based-movie-recommender-system-5876b9686df3.

[8] *Tf.keras.preprocessing.sequence.skipgrams | TensorFlow V2.14.0*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/skipgrams. Accessed 13 Dec. 2023.

[9] *Imdb.com*. (2023). https://developer.imdb.com/non-commercial-datasets/.