Comparative Analysis of Graph Neural Networks and Matrix Factorization for Movie Recommender Systems

Giacomo Visentin, Matteo Baldoni, Ortisa Poci

06/01/2025

GitHub repository:
`https://github.com/giacomovisentin/LightGCN-vs-MF-for-Movie-Recommendations.git`

Abstract

Recommender systems have become an integral part of various industries, including e-commerce, streaming services, and social media, by providing personalized content and product suggestions to users. Traditional recommender systems rely on techniques like Matrix Factorization, which decomposes user-item interactions to predict preferences. Recently, Graph Neural Networks (GNNs) have gained attention for their ability to model complex relationships in data, especially in user-item interaction networks. This project aims to compare the performance and suitability of Matrix Factorization and GNN-based approaches for recommendation tasks, by observing how each technique recommends movies.

1 Our Models

1.1 Matrix Factorization

Matrix Factorization (MF) in recommender systems is based on the idea of approximating the original rating matrix $\mathbf{R}$, which contains the values r_{ui} given by users u to items i. The matrix is largely incomplete, as not all users have rated all items and is approximated as the product of two smaller matrices:

$$\mathbf{R} \approx \mathbf{P} \cdot \mathbf{Q}^T \tag{1}$$

where $\mathbf{P} \in \mathbb{R}^{n_u \times k}$ is a matrix representing the latent factors of users, with each row $\mathbf{P}_u$ corresponding to the latent factor vector of a user u. Similarly, $\mathbf{Q} \in \mathbb{R}^{n_i \times k}$ is a matrix representing the latent factors of items, where each row $\mathbf{Q}_i$ corresponds to the latent factor vector of an item i. The parameter k denotes the number of latent factors, which are the hidden characteristics describing users and items.

Additionally, to account for systematic biases, the model can include bias terms for users and items, as well as a global bias term in our case.

1.1.1 Scalability of the matrix factorization

We would like to clarify our interpretation of the question regarding the scalability of Matrix Factorization. In our midterm report, we addressed this question by focusing on the scalability of the specific algorithm we use, namely the one described in [4]. However, we realized that the question might have been intended in a broader sense.

It is important to highlight that classical SVD is not scalable and operates on dense matrices, making it unsuitable for handling sparse matrices, which are commonly encountered in recommender systems. Its high computational and memory costs make it impractical for large datasets.

To overcome these challenges, the algorithm that won the Netflix Prize introduced an optimized and scalable approach to Matrix Factorization. This method, inspired by SVD, avoids explicitly computing singular values and instead uses an approximate factorization of latent factors through the iterative

minimization of a loss function. Specifically, it relies on stochastic gradient descent (SGD) to approximate the factor values.

Thanks to this formulation, the algorithm is both efficient and scalable, even for very large datasets, as it was designed with such requirements in mind. For this reason, in response to the question about the scalability of Matrix Factorization, we did not propose a specific solution: the issue does not arise with the algorithm we use, as scalability is ensured by its design.

1.1.2 Predictions

The prediction of a rating $\hat{r}_{ui}$ for a user u and an item i is performed using the following formula:

$$\hat{r}_{ui} = \mathbf{P}_u \cdot \mathbf{Q}_i^T + b_u + b_i + \mu \tag{2}$$

where $\mathbf{P}_u \in \mathbb{R}^k$ is the latent factor vector for user u, $\mathbf{Q}_i \in \mathbb{R}^k$ is the latent factor vector for item i, b_u represents the bias associated with the user, b_i is the bias associated with the item, and μ is the global bias, computed as the mean of all ratings in the dataset.

1.1.3 Loss function

The model is optimized by minimizing an objective function that combines the mean squared error between the observed and predicted ratings with a regularization term to prevent overfitting. The objective function is defined as:

$$L = \sum_{(u,i)\in\mathcal{R}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(\|\mathbf{P}_u\|^2 + \|\mathbf{Q}_i\|^2 + b_u^2 + b_i^2 \right), \tag{3}$$

where $\mathcal{R}$ is the set of user-item pairs for which ratings are available, r_{ui} is the actual rating, $\hat{r}_{ui}$ is the predicted rating, and λ is the regularization parameter.

The optimization is performed using the gradient descent method. The parameters $\mathbf{P}_u$, $\mathbf{Q}_i$, b_u, and b_i are updated iteratively by computing the gradients of the objective function. The updates are defined as follows:

$$\mathbf{P}_u \leftarrow \mathbf{P}_u + \alpha \cdot (2 \cdot e_{ui} \cdot \mathbf{Q}_i - \lambda \cdot \mathbf{P}_u), \tag{4}$$

$$\mathbf{Q}_i \leftarrow \mathbf{Q}_i + \alpha \cdot (2 \cdot e_{ui} \cdot \mathbf{P}_u - \lambda \cdot \mathbf{Q}_i), \tag{5}$$

$$b_u \leftarrow b_u + \alpha \cdot (2 \cdot e_{ui} - \lambda \cdot b_u), \tag{6}$$

$$b_i \leftarrow b_i + \alpha \cdot (2 \cdot e_{ui} - \lambda \cdot b_i), \tag{7}$$

where $e_{ui} = r_{ui} - \hat{r}_{ui}$ is the prediction error, and α is the learning rate.

1.1.4 Training

During training, the Root Mean Squared Error (RMSE) on the validation data is computed to monitor the model's improvement. If the RMSE does not improve for a certain number of consecutive iterations, the training process is stopped early (early stopping). Additionally, the best parameters $(\mathbf{P}, \mathbf{Q}, b_u, b_i)$ are saved to achieve optimal performance on the test data.

Finally, to ensure that the predicted ratings fall within the expected range of reviews (between 1 and 5), the result $\hat{r}_{ui}$ is clipped:

$$\hat{r}_{ui} \leftarrow \text{clip}(\hat{r}_{ui}, 1, 5).$$

1.1.5 Improvements since the Midterm Report

In the initial implementations of the algorithm, we encountered scalability issues concerning coverage. We observed that as the dataset size increased, the Mean Squared Error (MSE) decreased. However, this improvement came at the cost of reduced coverage, indicating that the model tended to recommend a narrower selection of movies as the dataset grew. Although the MSE values aligned with our expectations, a following analysis of the model's predictions in terms of recall and precision revealed that these metrics were unacceptably low. Therefore, we decided to improve the baseline model to achieve acceptable values across all evaluation metrics.

Our new advanced version of the Matrix Factorization algorithm includes several improvements. An enhancement involves the initialization of matrices: the Xavier initialization has been adopted for matrices P and Q, while biases are now initialized with small normal values (mean = 0, standard deviation = 0.01), replacing the previously used standard deviation = 1 that was too high and caused overflow during training.

Several optimizations have also been introduced during the training phase. Gradient descent is now performed using mini-batches of 1000 elements, in order to balance execution speed and learning stability. An early stopping mechanism with validation data has been implemented, effectively mitigating the risk of overfitting. The learning rate is subjected to controlled decay, being multiplied by 0.995 at each iteration.

Predictions have been further improved with the introduction of a clipping mechanism to constrain predicted values within the range $[1, 5]$, ensuring realistic ratings for the dataset used. Hyperparameters such as the learning rate, the number of latent factors, and regularization have been carefully optimized, resulting in better performance in most applications.

Overall, these enhancements have significantly improved the performance of the algorithm, ensuring more accurate predictions and stable convergence during training.

Regarding coverage, the model is no longer affected by the size of the dataset, and all metrics now show improvement when a larger number of reviews is used. This suggests that the initial model was likely too simple and prone to overfitting, causing it to rely heavily on recommending only popular items as a safe choice thereby reducing the coverage.

1.2 LightGCN

For the Graph Neural Network, we decided to use the LightGCN model [2]. As explained by the authors, LightGCN was designed to simplify Graph Convolutional Networks for recommendation tasks. It stands out for its simplicity and efficiency compared to traditional GCN models. LightGCN removes computationally expensive components, such as feature transformation layers and nonlinear activation functions. Instead, it focuses only on the essential aspects of GCN for collaborative filtering, namely neighborhood aggregation. The model learns user and item embeddings by propagating them linearly on the user-item interaction graph. This simplicity makes the model much easier to train than others, such as Neural Graph Collaborative Filtering (NGCF), and also brings substantial improvements, with a 16% performance increase.

1.2.1 Light Graph Convolution

Between each layer, LightGCN uses the following propagation rule for user and item embeddings. As can be seen, feature transformation and nonlinear activation are omitted. Let N_u denote the set of all neighbors of user u (representing the items liked by u), and similarly, N_i represents the neighbors of item i. The embedding $\boldsymbol{e}_u^{(k)}$ is the k-th layer user embedding, and $\boldsymbol{e}_i^{(k)}$ is the k-th layer item embedding:

$$\boldsymbol{e}_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} \boldsymbol{e}_i^{(k)} \quad \boldsymbol{e}_i^{(k+1)} = \sum_{u \in N_i} \frac{1}{\sqrt{|N_i|}\sqrt{|N_u|}} \boldsymbol{e}_u^{(k)} \tag{8}$$

1.2.2 Layer Combination and Model Prediction

We combine the embeddings obtained at each layer of propagation to form the final embeddings for all users and items, $\boldsymbol{e}_u$ and $\boldsymbol{e}_i$ via the following equation.

$$\boldsymbol{e}_u = \sum_{k=0}^{K} \alpha_k \boldsymbol{e}_u^{(k)} \quad \boldsymbol{e}_i = \sum_{k=0}^{K} \alpha_k \boldsymbol{e}_i^{(k)} \tag{9}$$

α_k : hyperparameter which weights the contribution of the k-th layer embedding to the final embedding

The model prediction is defined as the inner product of user and item final representations:

$$\hat{y}_{ui} = \boldsymbol{e}_u^T \boldsymbol{e}_i \tag{10}$$

1.2.3 Mathematical Representation

To formalize our approach, we define the interaction between users and items as a matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$, where M represents the total number of users and N the total number of items in our system. Each element R_{ui} in this matrix takes a binary value: 1 indicating user u has interacted with item i, and 0 indicating no interaction.

From this interaction matrix, we construct a symmetric adjacency matrix:

$$\mathbf{A} = \begin{pmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^T & \mathbf{0} \end{pmatrix} \tag{11}$$

For the embedding representation, we start with an initial matrix $\mathbf{E}^{(0)} \in \mathbb{R}^{(M+N) \times T}$, where T defines the dimension of our embedding space. The propagation rule for updating embeddings across layers follows:

$$\mathbf{E}^{(k+1)} = (\mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}) \mathbf{E}^{(k)} \tag{12}$$

Here, $\mathbf{D}$ represents a diagonal degree matrix of size $(M+N) \times (M+N)$, with each diagonal element D_{ii} corresponding to the number of connections in row i of matrix $\mathbf{A}$.

The final embedding representation combines information from all layers through a weighted sum:

$$\mathbf{E} = \sum_{k=0}^{K} \alpha_k \mathbf{E}^{(k)} \quad = \alpha_0 \mathbf{E}^{(0)} + \sum_{k=1}^{K} \alpha_k (\tilde{\mathbf{A}})^k \mathbf{E}^{(0)} \tag{13}$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ denotes the normalized adjacency matrix.

1.2.4 Loss function

The original LightGCN paper employs Bayesian Personalized Ranking (BPR) loss, a pairwise loss function that encourages the model to predict higher scores for observed entries compared to unobserved ones. However, the authors note that alternative loss functions could potentially yield better results.

In our work, we chose to use Mean Squared Error (MSE) for its simplicity and because it provides a more straightforward comparison with our matrix factorization model. The results were very promising, and we found no need to switch to a different loss function.

Unlike MSE, BPR is designed to improve ranking rather than predict absolute values. Its implementation typically focuses only on positive interactions (e.g., ratings above a threshold, such as 4 out of 5), discarding the rest of the data. Since we had access to the true rating values, we decided to utilize them directly rather than treating all ratings above the threshold as equally positive. This approach allowed us to better capture the nuances of the data, and the results confirmed the effectiveness of this choice.

1.2.5 Complexity

The only trainable parameters in LightGCN are the embeddings from the 0-th layer, represented as $\omega = \{\mathbf{E}^{(0)}\}$. This means that the model's complexity is equivalent to that of standard matrix factorization.

1.2.6 Implementation

The paper [2] also provided the code for the model implemented in PyTorch [9], which we used as a reference to develop our own. Additionally, we consulted the torch-geometric library, which offers its own implementation of LightGCN [8] and an article on Medium [10]. For the construction of our model, we used the 'MessagePassing' class [7], a base class designed to simplify the creation of message-passing graph neural networks by automatically handling message propagation.

2 Dataset used

In our experiment, we will use the MovieLens dataset [1], which is widely used for evaluating recommender systems. This dataset contains user ratings for a large collection of movies. Initially, we planned to use only the 1M version, but we later decided to include the 100K version as well to improve the depth and quality of our analysis. The MovieLens 1M dataset includes 1,000,209 ratings for approximately 3,900 movies from 6,040 users, while the MovieLens 100K dataset contains around 100,000 ratings from 943 users for 1,682 movies.

3 Evaluation Metrics

3.0.1 Mean Squared Error (MSE)

The Mean Squared Error (MSE) measures the average squared differences between predictions and actual values:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{14}$$

where y_i are the actual values, $\hat{y}_i$ are the predicted values, and n is the number of predictions. MSE is not the most important metric for recommender systems because the primary goal is to determine whether a user might like an item, rather than to predict the exact rating. However, there are situations where MSE can still be useful. In many real-world cases, explicit user ratings are unavailable, which limits the applicability of MSE. In our case, since explicit ratings are available, this metric becomes a valuable tool. Moreover, MSE is the loss function used in our LightGCN model and is also part of the loss function for Matrix Factorization (MF), where it is combined with a regularization term. This makes MSE particularly useful for directly comparing the performance of these two models.

3.0.2 Precision@k

Precision@k measures the proportion of relevant recommendations among the top k recommendations:

$$Precision@k = \frac{\text{Number of relevant items in top-}k}{k} \tag{15}$$

In our case, we define a movie as relevant for a user if the user rated it 4 stars or higher. We have chosen $k = 10$ for both Precision and Recall. Naturally, the value of Precision decreases as k increases because recommending more movies increases the likelihood of including less relevant ones.

3.0.3 Recall@k

Recall@k measures the proportion of relevant items retrieved among the top k recommendations:

$$Recall@k = \frac{\text{Number of relevant items in top-}k}{\text{Total number of relevant items}} \tag{16}$$

Unlike Precision, Recall increases as k increases, since recommending more movies raises the chance of retrieving a larger percentage of movies deemed relevant for each user.

3.0.4 Coverage

Coverage measures the proportion of items in the catalog that the system can recommend:

$$Coverage = \frac{|\text{Recommended Items}|}{|\text{Total Items}|} \tag{17}$$

A low coverage value indicates that the model tends to recommend the same set of movies repeatedly. While this is not inherently problematic if the goal is to recommend some relevant items to each user, it also implies that the model may lack the ability to cater to specific user preferences. Generally, excessively low coverage is undesirable.

3.0.5 Gini Index

The Gini Index is a measure of inequality that ranges from 0 (perfect equality) to 1 (maximum inequality). In recommender systems, it quantifies how evenly recommendations are distributed.

$$G = \frac{\sum_{i=1}^{n}(2i - n - 1)x_i}{n\sum_{i=1}^{n} x_i} \tag{18}$$

where x_i are the sorted values in ascending order, and n is the total number of observations.

The Gini Index complements the Coverage metric by assessing the diversity of recommendations. A high Coverage does not necessarily imply diverse recommendations, it could be the case that some movies are recommended to only one user while others are suggested to everyone. This metric helps us evaluate whether the recommendations are distributed uniformly or not.

4 Results

We chose $K = 10$, meaning the models recommend 10 movies each.

Matrix Factorization performs better in reducing the Mean Squared Error (MSE), but this metric is not necessarily the most important. In terms of recall and precision, MF results are particularly low. Additionally, MF has lower coverage and a higher Gini index, indicating that it recommends fewer movies from the catalog and tends to over-recommend certain movies with very high frequency.

In contrast, LightGCN models outperform MF in all metrics except MSE, achieving results that are up to 10 times better.

Dataset	**Test MSE**	**Precision@K**	**Recall@K**	**Coverage**	**Gini Index**
MF - 100k	0.8813	0.04615	0.0374	0.1155	0.9788
GNN - 100k	1.0208	0.5564	0.4079	0.4653	0.7431
MF - 1M	0.7989	0.0570	0.0394	0.1369	0.9829
GNN - 1M	1.1491	0.6492	0.3339	0.6375	0.8212

We decided to compare the results of our MF model with those obtained using the SVD model from the Surprise library [3]. This library, created from the same paper we used, is specifically designed to provide a useful baseline for developing recommender systems. In our case, we used it to evaluate the accuracy and quality of the predictions of our MF model, particularly to assess whether its predictions were valid given the presence of very low values in certain metrics. The results obtained were consistent with ours metrics, showing similar performance with an approximately 10% improvement, keeping both models within the same order of magnitude.

In the midterm, we stated our intention to also compare the MF and LightGCN models in terms of training times to determine which was faster. However, we realized that such a comparison would be less meaningful and would not accurately reflect the computational complexity of the two models. This is because they were implemented in entirely different ways: the Matrix Factorization model was developed using the NumPy library, while the LightGCN model was implemented using PyTorch. The latter leveraged the GPU acceleration provided by Colab, significantly reducing its training time, which otherwise would exceed an hour without GPU support. For these reasons, we decided not to include the training time comparison data.

5 Conclusions

LightGCN is no longer the leading model for recommender systems in collaborative filtering based on graph neural networks. Since its introduction, several advancements have been made, resulting in more efficient and effective models such as LightGCN++ [5] and UltraGCN [6].

However, the objective of this report was not to compare the best Matrix Factorization (MF) model with the best graph neural network (GNN). Instead, our aim was to evaluate whether, in general, a GNN could outperform an MF model in the context of recommender systems. For this purpose, we selected two well-known and widely recognized models for comparison.

The results show that the GNN model significantly outperforms the MF model. This can primarily be attributed to how GNNs represent users and items as nodes in a bipartite graph, effectively leveraging

both direct and indirect relationships to create more informative embeddings. In contrast, MF approaches treat user-item interactions as entries in a simple interaction matrix, neglecting the underlying topology of the data.

A key advantage of GNNs lies in their ability to propagate information across the graph. This allows for multi-level connections, for instance a user can be linked to a movie not only because they directly rated it but also because similar users have rated it. By exploiting the graph's topology, GNNs can capture richer relational patterns, resulting in a significant improvement in recommender system performance.

In conclusion, leveraging the graph structure and propagating information through it proves to be a highly effective strategy for enhancing recommender systems.

References

[1] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19:1–19:19, December 2015.

[2] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020.

[3] Nicolas Hug. Surprise: A python library for recommender systems. `https://surprise.readthedocs.io/en/stable/matrix_factorization.html`.

[4] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[5] Geon Lee, Kyungho Kim, and Kijung Shin. Revisiting lightgcn: Unexpected inflexibility, inconsistency, and a remedy towards improved recommendation. In *Proceedings of the 18th ACM Conference on Recommender Systems*, RecSys '24, page 957–962. Association for Computing Machinery, 2024.

[6] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. Ultragcn: Ultra simplification of graph convolutional networks for recommendation, 2023.

[7] PyG Team. Creating message passing networks. `https://pytorch-geometric.readthedocs.io/en/2.6.1/notes/create_gnn.html`.

[8] PyG Team. Source code for torch_geometric.nn.models.lightgcn. `https://pytorch-geometric.readthedocs.io/en/2.5.1/_modules/torch_geometric/nn/models/lightgcn.html`, 2023.

[9] Gustavo Ye. LightGCN-PyTorch. `https://github.com/gusye1234/LightGCN-PyTorch`, 2023.

[10] Ada Zhou. Lightgcn with pytorch geometric. `https://medium.com/stanford-cs224w/lightgcn-with-pytorch-geometric-91bab836471e`, 2022.

Contributions

The project was a collaborative effort among Giacomo, Matteo, and Ortisa, with specific tasks assigned to each team member and others completed collectively. Below is a detailed description of each member's contributions:

Idea Generation and Validation

The team collectively brainstormed the project idea and validated its feasibility by researching supporting sources.

Model Research

- Ortisa focused on identifying models for Matrix Factorization (MF).
- Giacomo researched models for Graph Neural Networks (GNNs).
- Matteo took responsibility for selecting an appropriate dataset.

Initial and Midterm Reports

- Matteo wrote the initial report explaining the project the first draft of the midterm report
- All members participated in revising both reports and addressed the questions raised by the professor in the feedback on the first report.

Implementation

- Ortisa implemented the first version of the MF model and later improved its performance.
- Giacomo implemented the LightGCN model.
- Matteo implemented the functions to compute model metrics and used the Surprise library to compare MF results.

Metric and Result Analysis

- Matteo researched the most appropriate metrics for evaluating the models, analyzed the results, and compared the performances of MF and LightGCN.

Final Report

- Ortisa authored the section on MF.
- Giacomo wrote the section on LightGCN and created the GitHub repository for the project.
- Matteo wrote the sections on metrics and results.
- The team jointly wrote the conclusions and reviewed the final report to ensure consistency and accuracy.