Recommendation Systems: From Basics to Deep Neural Networks

ELMIMOUNI Zakarya
ENSAE Paris
zakarya.elmimouni@ensae.fr

KHAIRALDIN Ahmed
ENSAE Paris
ahmed.khairaldin@ensae.fr

RAZIG Amine
ENSAE Paris
amine.razig@ensae.fr

Abstract

Recommendation systems are integral to many digital platforms, enabling personalized experiences by suggesting products, content, or services tailored to user preferences. This project explores various machine learning methods, ranging from basic algorithms to advanced deep learning techniques, aiming to identify effective strategies for delivering high-quality recommendations. Using the 2023 Amazon Reviews dataset, we evaluate these approaches to provide insights into their performance and implementation in real-world scenarios. All our experiments are available in this Github repository, where you can find the models code, some theoretical explanations and data visualization.

1 Introduction

Recommendation systems are a critical component of modern digital platforms, offering personalized user experiences by predicting preferences based on historical data. These systems rely on implicit feedback, such as user-item interactions, or explicit information, including ratings and reviews. Enriched metadata, such as product descriptions, prices, and features, further enhances their predictive capabilities.

The primary goal of this project is to evaluate and compare multiple machine learning approaches, ranging from foundational models to sophisticated deep learning techniques, as outlined in the existing literature. By systematically analyzing these methods, we aim to identify strategies that maximize recommendation quality while maintaining computational efficiency.

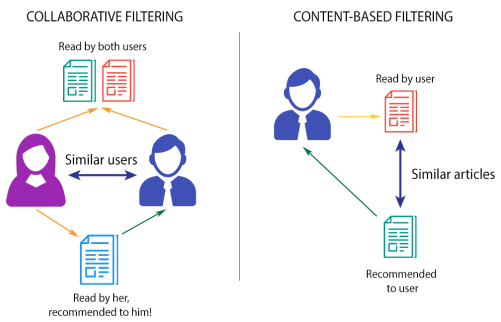

Figure 1: Illustration of the recommendation mechanism. Source: Medium post

We would like to point out that in this project, we're focusing in particular on methods based on learning and optimization based on Koren et al. [2009] and He et al. [2017] works, although we'll also be presenting deterministic methods such as cosine comparison and singular value decomposition.

2 Modeling and Results

2.1 User Similarity Recommendations

Basic models are based on user similarity, leveraging historical data to match user preferences, wich form the baseline for our analysis. Cosine similarity, is a common metric used in such models. The global idea is that it measures the similarity between two users by calculating the cosine of the angle between their preference vectors. It is what is called collaborative filtering (cf figure 1).

These vectors typically represent user interactions with items (for instance it can be ratings or interactions clicks). A cosine similarity value of 1 indicates perfectly aligned preferences, while a value of 0 indicates orthogonal, or no shared preferences. By identifying users with high similarity, the model can recommends items preferred by similar users. In our context the similarity formula is given by :

$$\text{Cosine Similarity}(u, v) = \frac{\sum_i r_{u,i} \cdot r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \cdot \sqrt{\sum_i r_{v,i}^2}}$$

- Where u and v represent two users.
- and $r_{u,i}$ and $r_{v,i}$ are the ratings of users u and v for item i.

2.1.1 SVD

Singular Value Decomposition is a foundational method in recommendation systems, used to decompose the user-item interaction matrix into three matrices: user features, singular values, and item features.

$$R \approx U \Sigma V^T$$

This decomposition identifies latent features that capture patterns in user preferences and item attributes, enabling predictions of missing ratings. The figure 2 shows a visual representation of users and itemp from the same dataset. However, SVD can be computationally expensive for large datasets due to its reliance on matrix factorization of potentially massive interaction matrices. The cost increases significantly with the number of users and items, making scalability a challenge.

To address this, dimensionality reduction is often applied by retaining only the top-k singular values and their corresponding singular vectors. This reduces computational requirements and storage while maintaining the most critical information. However, as it's show on the figure 5, this trade-off sacrifices some precision, as it eliminates less significant features that may still contribute to the recommendations.

Despite its utility, standard SVD and similar methods require the matrix to be fully populated, typically necessitating the imputation of missing values, which can introduce bias. This limitation leads to the exploration of matrix factorization learning techniques such as Alternating Least Squares and Stochastic Gradient Descent. These methods are designed for sparse data, offering more scalable and efficient approaches to recommendation systems by directly optimizing on the observed interactions.

2.2 Matrix Factorization

Matrix factorization plays a key role in collaborative filtering, where it is used to uncover latent factors that explain patterns in user-item interactions. The method works by decomposing a large user-item interaction matrix R into two smaller matrices: the User Matrix Q which encodes latent features of users, and the Item Matrix which encodes latent features of items. The latent features are hidden or underlying characteristics that are not directly observed but can be inferred from the data.

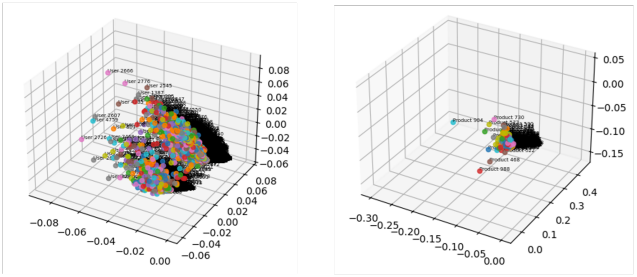

Figure 2: ***3D visualization of user (left) and product (right) embeddings***, showing the spatial clustering of products and users in the embedding space.

In this context, latent features can represent abstract properties of users and items that influence their interactions.

Therefore, the objective is to approximate this user-item interaction matrix using the product of these two matrices:

$$R \approx Q \cdot P^\top$$

where R represents the original user-item interaction matrix.

2.2.1 Stochastic Gradient descent Algorithm

As it's said above, there is many optimization techniques. For instance in the *Alternating Least Squares*, we fix P and solve for Q, then fix Q and solve for P. But, we decided to use *Stochastic Gradient Descent* by updating Q and P by iteratively taking gradient steps on a specific loss function.

The iterative training of this method helps decompose the user-item interaction matrix, uncovering latent factors that drive user preferences. Overall the idea is the same as the classic SVD, Matrix factorization by learning predicts missing entries in the user-item interaction matrix by decomposing it into user embeddings and item embeddings.The first one captures user preferences in a latent space while the second captures item characteristics in the same latent space.

Loss function

The objective of matrix factorization is to minimize the squared error between the predicted and observed values in the user-item interaction matrix, with a regularization term to prevent overfitting. As in the paper Koren et al. [2009] we define :

$$L = \sum_{(i,j)\in R} \left(R_{ij} - Q_i \cdot P_j^\top\right)^2 + \lambda \left(\|Q_i\|^2 + \|P_j\|^2\right)$$

Where:

- R: User-item interaction matrix.
- R_{ij}: Observed rating for user i and item j.
- Q_i: Latent vector for user i.
- P_j: Latent vector for item j.
- λ: Regularization parameter to prevent overfitting.
- $\|Q_i\|^2$ and $\|P_j\|^2$: Regularization terms for Q and P.

Algorithm 1 Matrix Factorization Algorithm

Input: User-item matrix R, regularization parameter λ, number of latent features k, learning rate η, max iterations T.
Step 1: Initialize Matrices
Randomly initialize $Q \in R^{m \times k}$ and $P \in R^{n \times k}$.
(Optional) Use metadata embeddings for informed initialization.
Step 2: Optimize for Loss Function
while not converged or iteration $< T$ **do**
    for $(i, j) \in R$ **do**
        Compute prediction error $e_{ij} = R_{ij} - Q_i \cdot P_j^\top$.
        Update Q_i:

$$Q_i \leftarrow Q_i + \eta \left(e_{ij} P_j - \lambda Q_i\right)$$

        Update P_j:

$$P_j \leftarrow P_j + \eta \left(e_{ij} Q_i - \lambda P_j\right)$$

    end for
end while
Output: Matrices Q and P with dimensions $m \times k$ and $n \times k$ respectively.

In this algorithm, the update rules are based on the stochastic gradient descent algorithm. We calculate the derivative of the loss function with respect to the latent vectors, its inverse gives us a direction of descent.

Therefore, this approach provides an approximation of the rating. The predicted rating for user i and item j is simply given by:

$$\hat{R}_{ij} = Q_i \cdot P_j^\top$$

In the next steps of the project, we transform the problem into a binary recommendation framework to improve the relevance of the recommendations by focusing on interactions. This approach is commonly referred to as **implicit feedback**. In this case, the loss function is called the Binary Cross entropy. She comes from the negative log-likelihood of a Bernoulli distribution, measuring the difference between predicted probabilities and actual binary labels (interaction or not) and is defined as:

$$L_{\text{binary}} = -\sum_{(u,i) \in R} \left(r_{ui} \log(\hat{r}_{ui}) + (1 - r_{ui}) \log(1 - \hat{r}_{ui})\right),$$

where:

- $r_{ui} \in \{0, 1\}$ represents the interaction (or lack thereof) between user u and item i,
- $\hat{r}_{ui} = \sigma(\mathbf{p}_u^\top \mathbf{q}_i)$ is the predicted interaction probability,
- $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

To optimize this loss function, we compute its gradient with respect to the latent vectors ($\mathbf{p}_u$ and $\mathbf{q}_i$) as before.

$$\frac{\partial L_{\text{binary}}}{\partial \mathbf{q}_i} = \sum_{(u,i) \in R} (\hat{r}_{ui} - r_{ui}) \mathbf{p}_u + 2\lambda \mathbf{q}_i, \quad \frac{\partial L_{\text{binary}}}{\partial \mathbf{p}_u} = \sum_{(u,i) \in R} (\hat{r}_{ui} - r_{ui}) \mathbf{q}_i + 2\lambda \mathbf{p}_u.$$

These gradients guide the updates of the latent vectors during training to minimize the binary loss function and improve the recommendation quality.

2.3 Neural Collaborative Filtering

Deep learning models, including neural collaborative filtering, are implemented to capture complex, non-linear relationships between users and items, enhancing recommendation quality.

In He et al. [2017], neural collaborative filtering is introduced as a multi-layer representation to model user-item interaction. Two models have been proposed in the literature involving neural collaborative filtering: a Multi-Layer Perceptron (MLP) using a non-linear kernel to learn the user-item interaction function and a combination of the MF algorithm and the MLP.

2.3.1 Multi-Layer Perceptron

As shown in Figure 2, NCF uses a one-hot encoding of both users and items. These sparse vectors are projected into a user embedding space and item embedding space as dense vectors.

In the MLP configuration, the dense representation of both users and items, which we denote as p_u and q_i, are concatenated before passing them to the MLP hidden layers. In other words, the user-item interaction function is obtained using the following equations:

$$z_1 = \begin{bmatrix} p_u \\ q_i \end{bmatrix}$$

$$z_j = a_j(W_j^T z_{j-1} + b_j) \quad , \text{ for } j = 2, ..., L$$

$$y_{ui} = \sigma(h_L^T z_{L-1})$$

where W_j, b_j, and a_j denote the weight matrix, bias vector, and activation function of the j-th layer for $j = 2, ..., L$ where L is the number of layers (the first layer is a concatenation layer). In the final layer, as the problem is binary, we apply the sigmoid function denoted σ to a linear combination of the last activation. The different activations of the hidden layers enable us to use non-linearity to model the user-item interaction function. In the structure implemented for our experiments, we opted for the ReLU function as the activation function for each hidden layer, which is well-suited for sparse data, as it encourages sparse activations.

2.3.2 Neural Matrix Factorization

As stacking different models has been proven to be an efficient method in machine learning, combining matrix factorization and an MLP, as shown in Figure 2, can be a good approach to estimate the user-item interaction.

For this, we use two different embeddings for users and items: one used for the matrix factorization algorithm and one for the MLP. For the matrix factorization part, we use the element-wise product of the two dense vectors of the first embedding for each of the users and items:

$$\phi^{MF} = p_u^G \odot q_i^G$$

where p_u^G and q_i^G are respectively the input matrix factorization dense vectors for user u and item i, and ϕ^{MF} is the output of the matrix factorization. For the MLP part, we use the two dense vectors of the second embedding for each of the items and users, as described earlier. The output of the MLP is given by:

$$z_1 = \begin{bmatrix} p_u \\ q_i \end{bmatrix}$$

$$z_j = a_j(W_j^T z_{j-1} + b_j) \quad , \text{ for } j = 2, ..., L$$

$$\phi^{MLP} = z_L$$

where W_j, b_j, and a_j denote the weight matrix, bias vector, and activation function of the j-th layer for $j = 2, ..., L$ where L is the number of layers (the first layer is a concatenation layer). The two outputs are concatenated and we apply the sigmoid function to them:

$$y_{ui} = h^T \begin{bmatrix} \phi^{MF} \\ \phi^{MLP} \end{bmatrix}$$

2.3.3 Learning in NCF

To update the parameters of these two models, we use the binary cross-entropy loss defined as:

$$L = -\sum_{(u,i)\in\mathcal{Y}\cup\mathcal{Y}^-} [y_{ui}\log(\hat{y}_{ui}) + (1-y_{ui})\log(1-\hat{y}_{ui})]$$

where y_{ui} is the ground-truth, $\hat{y}_{ui}$ is the probability of interaction between user u and item i obtained with the sigmoid function of the two models. The set $\mathcal{Y}$ contains all the observed interactions between users and items, and the set $\mathcal{Y}^-$ contains uniformly sampled negative samples, which are pairs of users and items with unobserved interactions ($y_{ui} = 0$).

3 Experiments

In this section, we will evaluate all models with a real dataset. Since these models predict whether there is an interaction between each user and item, we will recommend for each user new items with which there is most likely an interaction and evaluate these recommendations with recommender metrics.

3.1 Dataset and Methodology

3.1.1 Amazon Reviews 2023 Dataset

The dataset used in this study is the large-scale Amazon Reviews dataset, collected in 2023 by McAuley Lab[1]. It provides a comprehensive view of user interactions and product details, making it an excellent resource for recommendation system research. The dataset includes detailed user reviews, comprising ratings, textual feedback, and helpfulness votes. It also features rich item metadata, such as product descriptions, prices, and raw images, alongside graphs representing user-item interactions and "bought-together" relationships. This diverse and extensive dataset supports both user-centric and item-centric analyses, facilitating the development and benchmarking of advanced recommendation models.

3.1.2 Evaluation

Our methodology involves preprocessing the dataset to clean and structure the data for machine learning algorithms. We evaluate various models, splitting the data into training, and testing sets to ensure robust performance assessments.

To evaluate the performance of our recommendation models, we employed the following training and evaluation procedure. The model was trained by minimizing a defined loss function, optimizing the learned latent factors and neural network's weights for users and items. For evaluation, we split the dataset using a "Leave-Two-Out" strategy, ensuring that the last two items interacted with was used as the test instance for each user. The model's performance was assessed using four key metrics: Relative Absolute Error and Root Mean Squared Error for the matrix factorization and the binary cross-entropy, which quantify the training inference of the model. As for recommender systems metrics we use Hit Ratio (HR), Normalized Discounted Cumulative Gain (NDCG), precision@K and recall@K, which evaluate the quality of top-K recommendations. HR measures the frequency with which the relevant item appears in the top-K recommendations, while NDCG assesses the ranking quality by assigning higher importance to relevant items appearing earlier in the list.

Recommenders Metrics

The **Hit Ratio (HR)** measures whether at least one relevant item is present among the top K recommended items:

$$HR = \frac{\text{Number of hits}}{\text{Total number of test cases}}$$

[1] `https://amazon-reviews-2023.github.io`

The **Normalized Discounted Cumulative Gain (NDCG)** evaluates the quality of recommendations by considering the positions of relevant items in the recommended list:

$$DCG@K = \sum_{i=1}^{K} \frac{rel_i}{\log_2(i+1)}$$

$$NDCG@K = \frac{DCG@K}{IDCG@K}$$

where $rel_i \in \{0, 1\}$ is the relevance of the item at position i, and $IDCG@K$ is the ideal DCG (with all relevant items ranked at the top).

The **Precision@K** measures the proportion of relevant items among the top K recommended items:

$$Precision@K = \frac{\text{Number of relevant items in the top } K}{K}$$

The **Recall@K** measures the proportion of relevant items retrieved among the top K recommendations relative to the total number of relevant items (2 items in our case) available:

$$Recall@K = \frac{\text{Number of relevant items in the top } K}{\text{Total number of relevant items}}$$

3.2 Experiments

First, we evaluate our models with fixed parameters. We choose to recommend 10 items ($topK = 10$).

To make a comparison between the NeuMf and MLP models proposed by He et al. [2017], we use the adaptation of the matrix factorization algorithm for implicit feedback presented at the end of section 2.1.1.

The model BMF is trained with fixed learning rate of 0.01. Unlike Koren et al. [2009] here we use an adpatative regularization value which is a $\lambda > 0$ divided by the frequency of user or items. In fact, in many recommendation systems, the amount of interaction data is highly imbalanced: Some users have very few interactions, while others have many. Similarly, some items are very popular, while others are rarely interacted with.

Applying a uniform regularization term (e.g $\lambda = 0.1$) would penalize all parameters equally, regardless of the interaction frequency. For users or items with fewer interactions, uniform regularization can dominate the gradients, leading to underfitting. We train the model over 50 epochs.

The architecture of the MLP used in both models consists of 4 layers with 64, 32, 16, and 8 neurons, respectively, as in He et al. [2017]. The choice of this architecture is motivated by the similar order of magnitude of data sizes. To train the MLP, we use a learning rate of 0.1 and mini-batch training with a batch size of 256 samples. Moreover, to train our models, we sample a certain number of negative items for each user to avoid unbalanced data cases due to the sparsity of the data. For this parameter, we chose to sample 4 negative samples for each user. It is important to note that these parameters were fine-tuned manually due to the computational costs associated with such algorithms. As for the optimizer, we chose AdamW, which is an improved version of Adam that incorporates weight decay regularization to prevent overfitting. In AdamW, the weight decay is decoupled from the gradient updates, which allows for better regularization during training:

$$\theta_t = (1 - \eta\lambda)\theta_{t-1} - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

where θ_t is the parameter (weight or embedding matrix) at iteration t, η is the learning rate, v_t is the moving average of squared gradients, ϵ is a small constant to avoid division by zero, m_t is the moving average of gradients, and λ is the weight decay factor.

First, we evaluate the loss evolution for our models. For the MLP and NeuMF models, we restrict the evaluation to the losses over 20 epochs, even though the loss continues to decrease as shown in the figure. This choice is motivated by the overfitting observed around the 10th iteration, as the

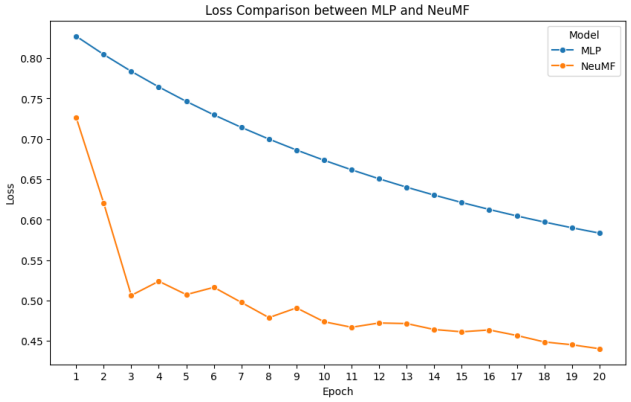

Figure 3: ***Loss evolution for NeuMF and MLP models along 20 epochs***

recommendation metrics reach their peak at this epoch and start to decline thereafter. This might be the cost of choosing such complex models, in addition to their considerable computational cost.

Then, we illustrate the impact of varying the number of recommended items (topK) on the performance of the model on the test set with the recommendation systems metrics. (figure 4)

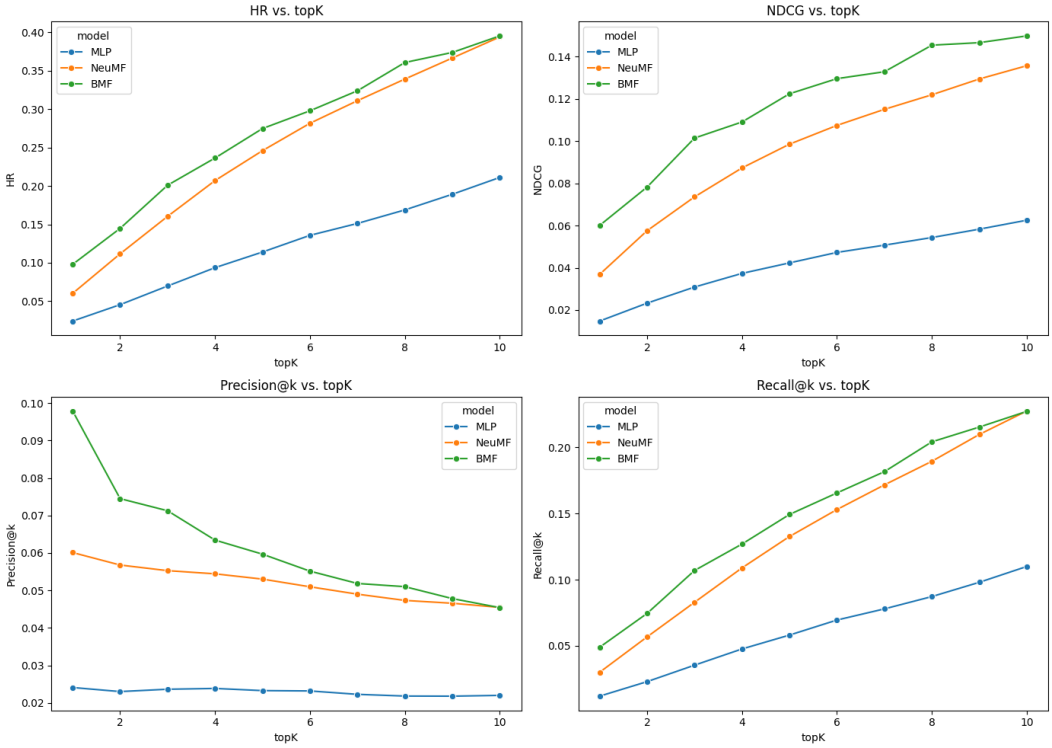

Figure 4: ***Recommendation metrics evaluation on Leave-two-out test set*** *for Binary Matrix factorization, Multi-layer Perceptron and Neural Matrix Factorization*

We note that on the data used in this experiment, the matrix factorization model applied to implicit feedback outperforms NeuMF and MLP across all metrics (cf figure 4), making it the best-performing model in this comparison, regardless of the topK number chosen.

The gap between BMF and NeuMF narrows as topK increases, until both models provide similar performance for k= 10 (cf table 1). MLP's performance is much weaker than the other two methods.

Model	HR	NDCG	Precision@k	Recall@k
MLP	0.211083	0.062519	0.021989	0.109947
NeuMF	0.393969	0.135664	0.045467	0.227335
BMF	0.395340	0.149782	0.045408	0.227041

Table 1: Summary of Metrics for Different Models at $k = 10$, leave-2-out test set

Also, Precision decreases with larger topK, highlighting the trade-off between retrieving more items (recall) and maintaining high accuracy (precision).

4 Conclusion

This study systematically evaluates various machine learning approaches for recommendation systems, emphasizing their strengths and limitations in delivering personalized user experiences. By utilizing the 2023 Amazon Reviews dataset, we demonstrate the potential of advanced models, such as neural collaborative filtering, in improving recommendation accuracy.

We observe implicit feedback matrix Factorization outperforming NeuMF and MLP, which is interesting and we try to understand why. In one hand, The first point to emphasize is that latent Feature Learning of Matrix factorization methods like BMF are highly effective in capturing latent user and item features from implicit feedback. These latent features often align closely with the underlying patterns in the data, resulting in better generalization for recommendation tasks.

Also, BMF's simpler structure and reliance on regularized factorization of observed interactions allow it to gives better results, especially with sparse implicit feedback datasets which is our case. NeuMF and MLP, being more complex neural models, may overfit the sparse data or fail to learn meaningful patterns from the limited signals.

We see that for high topK, the marginal advantage of BMF over NeuMF diminishes because both models start including more items that are moderately relevant, diluting the distinction in their performance. In other hand, we oberve a weaker performance of MLP model. We know that MLP based models often require careful design of input features and sufficient data to perform well. In the absence of explicit features or when working with implicit feedback, their general purpose nature may hinder their ability to model complex patterns compared to specialized approaches like matrix factorization. The sparcity may also impact his performance.

To conclude, the performance of BMF demonstrates the strength of matrix factorization in leveraging implicit feedback to learn meaningful latent factors for recommendation.

There are many other ways of approaching the field of recommendation with more or less complex methods that we haven't explored on in this project. In particular, the use of metadata to enrich the representation of objects and users in more complex latent spaces. The use of hybrid approaches is also very important in this kind of project and give interesting results.

5 Appendix

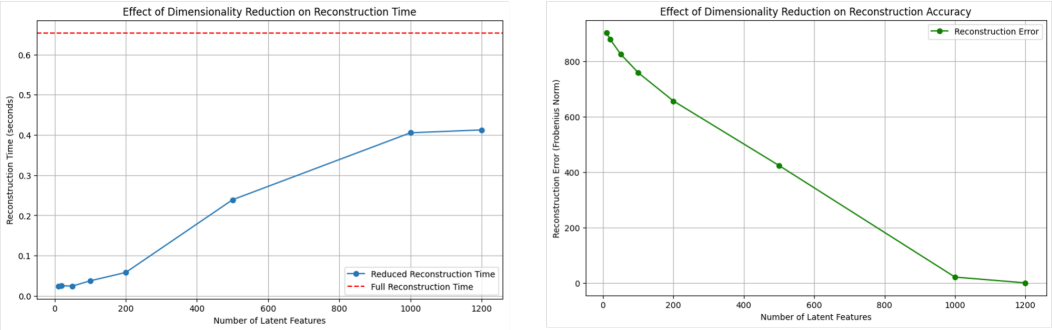

Figure 5: ***SVD : Trade-off analysis between dimensionality reduction, reconstruction time, and reconstruction error.***
We select a specific number of top singular values, along with their corresponding singular vectors, to reconstruct the matrix. By limiting the number of singular values and vectors used, we perform a reduced-dimension reconstruction, focusing only on the most significant components of the matrix as determined by the linear algebra SVD approach.
We measure the time taken for the reduced-dimension reconstructions and compare it to the time required for full reconstruction. This demonstrates how computation time decreases as top singular values number is reduced. This analysis is important because it highlights the trade-off between accuracy and computational efficiency.

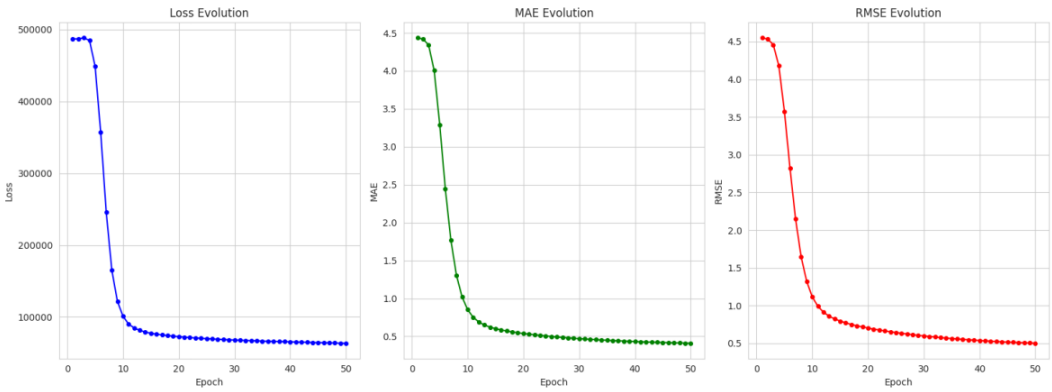

Figure 6: ***Explicit feedback Matrix factorization training loss evolution.***
In the case of explicit feedback (ratings from 1 to 5) MAE measures the average magnitude of errors in predictions, providing a straightforward interpretation of model accuracy. RMSE highlights larger errors by penalizing them more heavily, making it sensitive to outliers.Observation : MAE and RMSE values decrease over epochs, indicating improved prediction accuracy and reduced error magnitude as the model trains.

Derivation of Binary Cross-Entropy Loss

For implicit feedback we are trying to optimize a specifique loss function adapted to binary problems. We show here how to recover the formula in the general case :

For binary classification, the likelihood for a single observation is:

$$P(y|\hat{y}) = \hat{y}^y(1-\hat{y})^{1-y},$$

where $y \in \{0, 1\}$ is the true label and $\hat{y} \in [0, 1]$ is the predicted probability.

The log-likelihood is:

$$\log P(y|\hat{y}) = y \log(\hat{y}) + (1-y) \log(1-\hat{y}).$$

We define the negative log-likelihood and we want to minimize it :

$$\ell(\hat{y}, y) = -\big(y \log(\hat{y}) + (1-y) \log(1-\hat{y})\big).$$

Finaly, for N observations, the total loss is:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \big(y_i \log(\hat{y}_i) + (1-y_i) \log(1-\hat{y}_i)\big).$$

Results for BMF

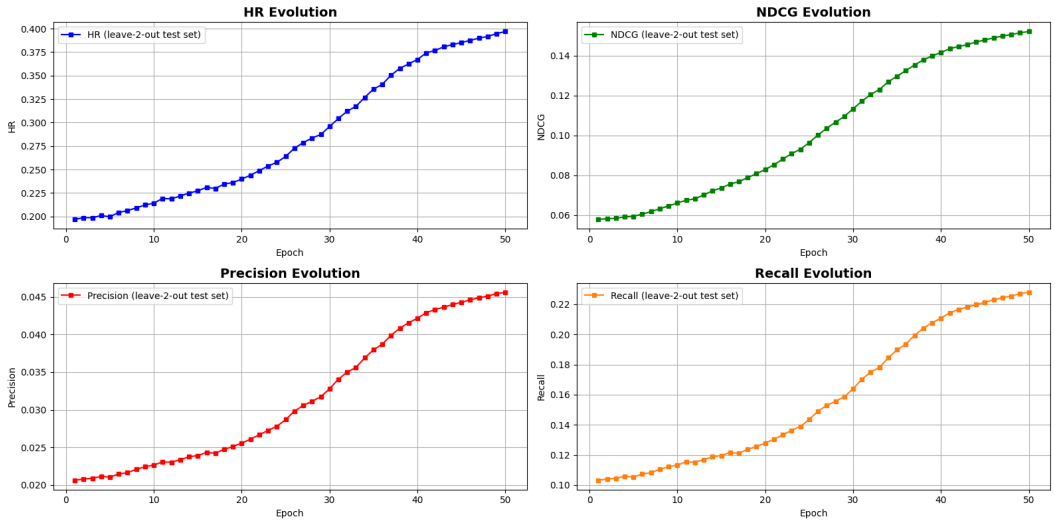

Figure 7: ***Evolution of HR, NDCG, Precision, and Recall over epochs for the leave-2-out test set.*** This plots concern Implicit Feedback Matrix Fcatorization. All metrics demonstrate consistent improvement with increasing epochs, indicating enhanced model performance through training.

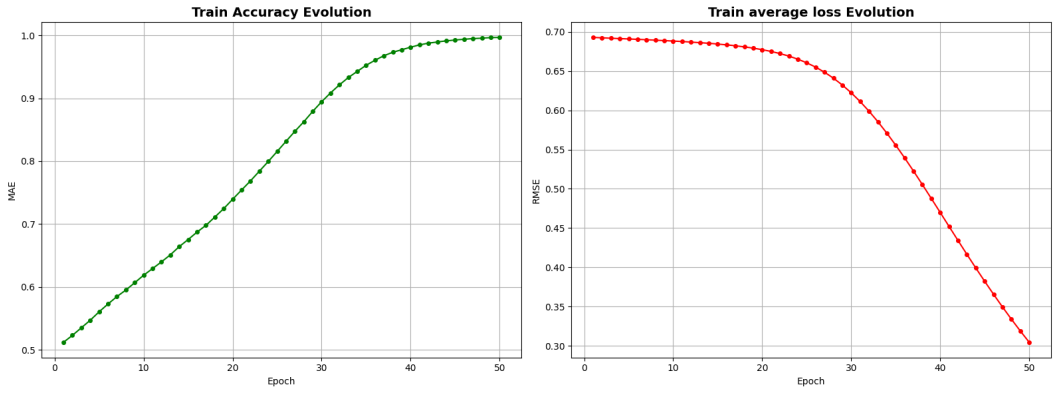

Figure 8: ***Evolution of BMF train accuracy and average loss.*** This plots concern Implicit Feedback Matrix Fcatorization.

GitHub

GitHub Repository link : https://github.com/arazig/Advanced-ML-project.git

References

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

Bhargav Kanagal Steffen Rendle Immanuel Bayer, Xiangnan He. A generic coordinate descent framework for learning from implicit feedback. 2016.

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

Yehuda Koren Yifan Hu and Chris Volinsky. Collaborative filtering for implicit feedback datasets. 2008.