Robust Solutions for Ranking Variability in Recommender Systems*

Bonifacio Marco Francomano[1], Federico Siciliano[1] and Fabrizio Silvestri[1]

[1]Sapienza University of Rome, Rome, Italy

Abstract

In the field of recommender systems, an important issue within the current state-of-the-art is the inconsistency in item rankings produced by models initialized with different weight seeds. Despite these models achieve convergence and obtain similar average performance metrics, their item rankings differ significantly. This phenomenon is quantitatively demonstrated using metrics such as Rank List Sensitivity (RLS) and Normalized Discounted Cumulative Gain (NDCG) across different model pairs. In this paper, we reaffirm the existence of this problem and provide new insights by analysing models with common item embeddings but different network initialization, and different item embeddings but common network initialization, to identify which network components most influence ranking variability. To address the general issue, we propose an ensemble approach that averages the output of multiple models. Our ensemble maintains the NDCG of the original model while significantly improving ranking stability: the RLS FRBO@10 value shows an approximate increase of 30.82%.

Keywords

Recommender Systems, Evaluation of Recommender Systems, Model Stability

1. Introduction

In the recent years, neural sequential recommender systems have gained importance due to their ability to model user behavior over time, providing more accurate and personalized recommendations[1, 2]. Unlike traditional recommender systems that consider user preferences in a static context, neural sequential recommender systems capture the temporal dynamics of user interactions[3]. This capability is central in domains such as e-commerce[4], streaming services[5], and social media[6]. By analyzing the sequence of items a user interacts with, these systems can predict future preferences[7].

Despite the advancements in neural sequential recommender systems, a significant issue persists: the variability in item rankings generated by models initialized with different weight seeds[8]. This rank variability is problematic as it affects the consistency and reliability of recommendations[9, 10]. Consider the following example of ranking variability between two different initializations of a model (tables 1 and 2):

In both initializations, when the models arrive at convergence, **Item A** is consistently ranked as the top item, which is expected as it is the correct positive item that the model should prioritize. However, there is significant variability in the ranking of the other items.

For instance, **Item B** is ranked 2nd in Initialization 1 but drops to 3rd in Initialization 2. Similarly, **Item C** rises from 3rd in Initialization 1 to 2nd in Initialization 2. This variability can occur because the loss function used in training typically focuses on ensuring the correct positive item is ranked highest, but does not enforce a specific order for the remaining negative items.

Even when models converge and predict the same top-ranked item for a given user sequence, the subsequent items in the ranking often differ. This inconsistency can affect tasks that require multiple item predictions simultaneously, generate relevance-ordered rankings, and impact the explainability of the model's recommendations. In particular, we are referring to these related works that address the sensitivity and robustness of recommender systems[11]. Oh et al.[9] have shown that recommender systems are highly sensitive to perturbations in the training data, where even minor changes can significantly alter the recommendations for users. They introduce Rank List Sensitivity (RLS) as a measure to assess this instability and propose the CASPER method, which identifies minimal perturbations that induce significant instability. Their experiments reveal that such perturbations, even if minimal, can drastically impact the recommendation lists, particularly for users who receive low-quality recommendations. Similarly, Betello et al.[12]

Table 1
Ranking Initialization 1

Rank	Item ID
1	Item A
2	Item B
3	Item C
4	Item D
5	Item E

Table 2
Ranking Initialization 2

Rank	Item ID
1	Item A
2	Item C
3	Item B
4	Item E
5	Item D

RobustRecSys: Design, Evaluation, and Deployment of Robust Recommender Systems Workshop @ RecSys 2024, 18 October, 2024, Bari, Italy.

*This work was partially supported by projects FAIR (PE0000013) and SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. Supported also by the ERC Advanced Grant 788893 AMDROMA, EC H2020RIA project "SoBigData++" (871042), PNRR MUR project IR0000013-SoBigData.it. This work has been supported by the project NEREO (Neural Reasoning over Open Data) project funded by the Italian Ministry of Education and Research (PRIN) Grant no. 2022AEFHAZ.

francomano.1883955@studenti.uniroma1.it (B. M. Francomano); siciliano@diag.uniroma1.it (F. Siciliano); fsilvestri@diag.uniroma1.it (F. Silvestri)

0000-0003-1339-6983 (F. Siciliano); 0000-0001-7669-9055 (F. Silvestri)

investigate the robustness of Sequential Recommender Systems (SRSs) in the face of training data perturbations. They identify limitations in existing robustness measures like Rank-Biased Overlap (RBO) and propose Finite Rank-Biased Overlap (FRBO), a more suitable metric for finite rankings. Their findings highlight that perturbations at the end of a sequence can severely degrade system performance, emphasizing the importance of the position of perturbations within the training data. To address this challenge, this paper investigates the root causes of ranking variability, focusing on the role of weight initialisation. We investigate whether this variability is primarily due to the initialisation of item embeddings or to the initialisation of the whole network. Through a detailed analysis using metrics such as RLS and NDCG, we show how different weight initialisation seeds lead to significant discrepancies in ranking results, highlighting the need for more robust approaches. As a solution, we propose the use of ensemble models, which combine the predictions of multiple models initialized with different seeds. By averaging the scores of these models, we can improve the stability of the rankings while maintaining or even improving the overall performance as measured by the NDCG.

Our experiments show that ensemble methods effectively reduce ranking variability. Specifically, models with different initialization seeds obtain an average RLS-FRBO@10 of 0.542, while an ensemble of models achieves a score of 0.709. Furthermore, we show that the average NDCG@10 across models with different seeds is 0.129, while an ensemble approach improves it to 0.132. This results show that ensemble methods not only reduce variability but also maintain or even slightly enhance recommendation quality. Our contributions can be summarized as follows:

- We identified the significant impact of weight initialization on the variability of item rankings in recommender systems.
- We proposed the use of ensemble models to reduce this variability, demonstrating that shared embeddings can further improve ranking consistency.

2. Methodology

2.1. Sequential Recommendation Model

The experimental framework employs the SASRec model[13], a state-of-the-art sequential recommendation system that uses self-attention mechanisms. The model processes user interaction sequences in the following stages:

- **Embedding Layer:** Items in the user interaction sequence are transformed into dense vector representations in a continuous vector space. Given an item i from the vocabulary of size N, the embedding layer maps it to a dense vector $\mathbf{v}_i \in \mathbb{R}^d$, where d is the embedding dimension:

$$\mathbf{v}_i = \mathbf{E}_i \mathbf{x}_i$$

  Here, $\mathbf{E}_i \in \mathbb{R}^{d \times N}$ is the embedding matrix, and $\mathbf{x}_i$ is a one-hot encoded vector representing item i.
- **Unidirectional Self-Attention Mechanism:** The SASRec model uses a unidirectional self-attention mechanism, which focuses on identifying temporal dependencies in the sequence of user interactions, considering only the items that precede a given item in the sequence. The attention score α_{ij} for items i and j (with $j < i$) is computed as:

$$\alpha_{ij} = \text{softmax}\left(\frac{(\mathbf{q}_i)^\top \mathbf{k}_j}{\sqrt{d_k}}\right)$$

  where $\mathbf{q}_i = \mathbf{W}_Q \mathbf{v}_i$, $\mathbf{k}_j = \mathbf{W}_K \mathbf{v}_j$, $\mathbf{W}_Q$, $\mathbf{W}_K$ are learned weight matrices, and d_k is the dimension of the key vectors. This mechanism ensures that the model only attends to past items, maintaining the sequential nature of the recommendation process.
- **Prediction and Ranking:** The attention outputs are used to predict the next item in the sequence, generating a ranked list of recommendations based on the user's interaction history. The output of the self-attention mechanism yields a context-aware embedding $\mathbf{z}_i$ for each item:

$$\mathbf{z}_i = \sum_{j=1}^{i-1} \alpha_{ij} \mathbf{v}_j$$

  The next item $\hat{i}_{t+1}$ is predicted by applying a softmax function over the dot products between $\mathbf{z}_i$ and the embedding matrix $\mathbf{E}$:

$$\hat{i}_{t+1} = \text{argmax}\left(\text{softmax}\left(\mathbf{z}_i^\top \mathbf{E}\right)\right)$$

For example, if the user interacted with items $[i_1, i_2, \ldots, i_t]$, the model will generate predictions for the next likely item $\hat{i}_{t+1}$ based on the context-aware embeddings. Assume i_1 corresponds to "Book A", i_2 to "Book B", and so on. If the attention mechanism strongly associates "Book B" with "Book D" in the past, "Book D" might be ranked higher as the next recommendation $\hat{i}_{t+1}$.

For this study, the SASRec model configuration includes an embedding dimension of 50, a single self-attention head, and a dropout rate of 0.2, the latter being employed to reduce the risk of overfitting.

2.2. Dataset

The dataset employed for this study is the MovieLens 1M (ML-1M) dataset[14], which is a widely recognized benchmark for evaluating recommendation algorithms. The MovieLens 1M dataset contains 1 million ratings provided by approximately 6,000 users on 4,000 movies. Each user in the dataset has rated at least 20 movies, making it a dense dataset that is well-suited for assessing the performance of sequential recommendation models like SASRec.

2.3. Dataset Preparation

The experimental dataset is curated with careful consideration of the following preprocessing steps:

- **Rating Threshold:** To ensure meaningful interaction data, only users who have rated at least 5 items, and items rated by at least 5 users, are retained.
- **Data Partitioning:** A leave-one-out strategy is adopted for data splitting. In particular, for each user, the most recent interaction is used for testing, while the one before the last is reserved for validation. The remaining data constitutes the training set.

- **Negative Sampling:** Negative sampling is employed to balance the dataset. During training and validation, one negative sample per positive instance is generated, whereas, in the testing phase, all non-interacted items are considered as negative samples.

2.4. Implementation Details

The training pipeline is meticulously designed to optimize performance and efficiency:

- **Batch Processing:** Training is conducted with a batch size of 128, ensuring the model can efficiently process substantial data per iteration.
- **Model Checkpointing:** To capture the most optimal model configuration, checkpoints are saved at intervals. The best model is determined based on the highest NDCG@10 score observed during validation.
- **Training Setup:** The model undergoes training for up to 200 epochs, utilizing the Adam optimizer. The objective function is the Binary Cross-Entropy Loss, which is particularly suited for this task. It is a suitable choice because it computes probabilities over two classes, the positive one and the negatives, since the model has to guess if the items is or not the next one in the sequence. Additionally, BCE Loss outputs probabilities that can be used directly to rank items by their likelihood of being the next interaction.

2.5. Metrics

The evaluation of model performance is based on several metrics, addressing both predictive accuracy and robustness.

- **Performance Metrics:** The primary metric for assessing predictive performance is NDCG (Normalized Discounted Cumulative Gain), calculated at various cutoffs (5, 10, and 20). NDCG evaluates the ranking quality of the recommended items, giving higher importance to items ranked closer to the top of the list.
- **Robustness Metrics:** To evaluate the robustness of the recommender system, we use the Rank List Sensitivity (RLS) metric.

$$RLS = \frac{1}{|X_{\text{test}}|} \sum_{X_k \in X_{\text{test}}} \text{sim}(R^{X_k}_{\mathcal{M}}, R^{X_k}_{\mathcal{M}'})$$

Here, X_{test} is the set of test items, $R^{X_k}_{\mathcal{M}}$ and $R^{X_k}_{\mathcal{M}'}$ are the rank lists generated by the model $\mathcal{M}$ and a perturbed version $\mathcal{M}'$ for item X_k. The function $\text{sim}(A, B)$ measures the similarity between two rank lists A and B. The RLS metric gives an average similarity score over all test items. RLS measures how much a model's recommendations change in response to small perturbations in the training data. There are two versions of RLS: the RLS-RBO (Rank-Biased Overlap) and the RLS-JAC (Jaccard Similarity). RLS-RBO, based on the Rank-Biased Overlap, computes the similarity between two ranked lists, but it is tailored for infinite rankings, which can limit its applicability in finite settings.

$$RBO(A, B) = (1 - p) \sum_{d=1}^{|I|} p^{d-1} \frac{|A[1:d] \cap B[1:d]|}{d}$$

The Rank-biased Overlap (RBO) measures the similarity of orderings between two rank lists A and B. The parameter p (typically set to 0.9) controls the weighting, with higher weights given to the top ranks. $|I|$ is the total number of items, and $A[1:d]$ represents the top-d items in list A. The RBO score lies between 0 and 1, where higher values indicate more similarity between the rank lists. RLS-JAC, on the other hand, uses the Jaccard similarity coefficient, focusing on the overlap of items between two sets without considering their order.

$$\text{Jaccard} = \frac{|A \cap B|}{|A \cup B|}$$

where A and B are the sets of items in the two ranked lists. Given the limitations of RLS-RBO, we utilize an enhanced version called Finite Rank-Biased Overlap (FRBO)[12], which is specifically designed for finite-length rankings.

$$FRBO(X, Y)@k = \frac{1-p}{1-p^k} \sum_{d=1}^{k} p^{d-1} \frac{|X[1:d] \cap Y[1:d]|}{d}$$

Here, p is a parameter that controls the weight assigned to ranks, and $X[1:d]$ and $Y[1:d]$ represent the top-d items in the two ranked lists being compared.
The RBO value is now normalized to its maximum possible value, ensuring that the metric reaches 1 when the two rankings are identical.
FRBO addresses the shortcomings of RBO by correctly handling identical rankings, making it more suitable for practical evaluation scenarios.
By comparing different versions of RLS, we aim to capture how robustly the models handle perturbations, ensuring a more reliable assessment of system stability. For brevity, we will refer to the Jaccard version of RLS as $RLS_J@k$, and the FRBO version of RLS as $RLS_F@k$ in the subsequent sections and tables. The notation @k indicates a cutoff at the first k item of the lists.

3. Research Questions

We aim to answer the following questions:

- **RQ1**: Does a variation of (an apparently) small factor influence the robustness of Neural Recommender Systems?
- **RQ2**: Do ensemble of models improve the robustness of Neural RecSys?
- **RQ3**: Can we improve performance through ensembling?

3.1. RQ1

Tables 3 and 4 reveal that models initialized with different seeds, which means entirely different weight initializations, obtain an average Jaccard@10 of 0.505 and an average FRBO@10 of 0.542, both of which are significantly below the ideal value of 1.

When initializing only the item embeddings with the same seed, Tables 5 and 6 show a slight but not significant increase, getting a value of 0.514 for the Jaccard@10, and 0.547 for the FRBO@10.

Seed 1	Seed 2	RLS_J@5	RLS_J@10	RLS_J@20
42	43	0.465	0.508	0.565
42	44	0.464	0.507	0.563
42	45	0.456	0.498	0.555
43	44	0.464	0.507	0.563
43	45	0.463	0.506	0.561
44	45	0.461	0.504	0.560
AVG ± SD		**0.462 ± 0.001**	**0.505 ± 0.001**	**0.561 ± 0.001**

Table 3
RLS_J computed between models initialized with different seeds, for different values of k (5, 10, and 20).

Seed 1	Seed 2	RLS_F@5	RLS_F@10	RLS_F@20
42	43	0.499	0.546	0.580
42	44	0.497	0.544	0.579
42	45	0.489	0.536	0.571
43	44	0.497	0.544	0.578
43	45	0.495	0.542	0.577
44	45	0.493	0.541	0.575
AVG ± SD		**0.495 ± 0.001**	**0.542 ± 0.001**	**0.577 ± 0.001**

Table 4
RLS_F computed between models initialized with different seeds, for different values of k (5, 10, and 20).

Seeds 1	Seeds 2	RLS_J@5	RLS_J@10	RLS_J@20
43_42	43_45	0.481	0.528	0.581
44_42	44_43	0.469	0.517	0.571
42_43	42_44	0.467	0.515	0.569
42_44	42_45	0.467	0.516	0.569
42_43	42_45	0.466	0.513	0.567
43_44	43_45	0.465	0.513	0.567
43_42	43_45	0.465	0.511	0.566
44_43	44_45	0.462	0.510	0.565
44_42	44_45	0.462	0.511	0.565
45_43	45_42	0.462	0.511	0.565
45_43	45_44	0.463	0.511	0.566
45_42	45_44	0.464	0.512	0.566
AVG ± SD		**0.466 ± 0.001**	**0.514 ± 0.001**	**0.568 ± 0.001**

Table 5
RLS_J computed between models initialized with the same embedding seeds, for different values of k (5, 10, and 20). The notation x_y represents a model where the embedding layer is initialized with seed x, while the rest of the model is initialized with seed y.

Seeds 1	Seeds 2	RLS_F@5	RLS_F@10	RLS_F@20
43_44	43_42	0.514	0.562	0.596
44_42	44_43	0.502	0.551	0.586
42_43	42_44	0.501	0.549	0.584
42_44	42_45	0.501	0.549	0.584
42_43	42_45	0.498	0.546	0.582
43_44	43_45	0.498	0.547	0.582
43_42	43_45	0.496	0.545	0.580
45_43	45_44	0.494	0.543	0.578
44_42	44_45	0.495	0.543	0.579
44_43	44_45	0.496	0.545	0.580
AVG ± SD		**0.499 ± 0.002**	**0.547 ± 0.001**	**0.582 ± 0.001**

Table 6
RLS_F computed between models initialized with the same embedding seeds, for different values of k (5, 10, and 20). The notation x_y represents a model where the embedding layer is initialized with seed x, while the rest of the model is initialized with seed y.

Instead, initializing the rest of the network with the same seed but using different embedding seeds, leads to a significant improvement: in Tables 7 and 8 we observe an value of 0.541 for the Jaccard@10 and 0.575 for the FRBO@10.

Seeds 1	Seeds 2	RLS_J@5	RLS_J@10	RLS_J@20
43_44	42_44	0.461	0.507	0.564
44_42	43_42	0.467	0.515	0.569
44_43	42_43	0.465	0.513	0.567
42_43	45_43	0.485	0.533	0.586
42_44	45_44	0.481	0.529	0.582
43_45	44_45	0.494	0.542	0.594
44_45	42_45	0.503	0.550	0.602
43_45	42_45	0.513	0.561	0.612
43_42	45_42	0.510	0.557	0.609
44_42	45_42	0.516	0.564	0.614
44_43	45_43	0.511	0.558	0.610
43_44	45_44	0.518	0.565	0.616
AVG ± SD		**0.494 ± 0.006**	**0.541 ± 0.006**	**0.594 ± 0.005**

Table 7
RLS_J computed between models initialized with same rest of the network seeds for different values of k (5, 10, and 20). The notation x_y represents a model where the embedding layer is initialized with seed x, while the rest of the model is initialized with seed y.

Seeds 1	Seeds 2	RLS_F@5	RLS_F@10	RLS_F@20
43_44	42_44	0.501	0.546	0.581
44_42	43_42	0.505	0.551	0.585
44_43	42_43	0.502	0.549	0.583
42_43	45_43	0.522	0.568	0.602
42_44	45_44	0.517	0.564	0.598
43_45	44_45	0.529	0.575	0.609
44_45	42_45	0.537	0.583	0.617
43_45	42_45	0.546	0.593	0.626
43_42	45_42	0.543	0.590	0.623
44_42	45_42	0.549	0.595	0.628
44_43	45_43	0.544	0.591	0.624
43_44	45_44	0.551	0.597	0.630
AVG ± SD		**0.529 ± 0.005**	**0.575 ± 0.005**	**0.609 ± 0.005**

Table 8
RLS_F computed between models initialized with same rest of the network seed, for different values of k (5, 10, and 20). The notation x_y represents a model where the embedding layer is initialized with seed x, while the rest of the model is initialized with seed y.

These results suggest that embedding initialization has a limited impact on the final model performance and resulting rankings. We therefore aim to investigate the relationship between two final trained embedding spaces.

3.1.1. Investigation of Embedding Spaces

To explore the relationship between embeddings from different models, we applied a linear transformation to map the embeddings from one model to another, followed by a visualization using Principal Component Analysis (PCA). This analysis aims to provide insights into how well the embeddings from different models align after the transformation.

The experimental procedure involved the following steps:

1. **Embedding matrices:** X and Y are the embedding matrices of the first and second models, respectively, each with a shape of $N \times d$.
2. **Fitting a Linear Regression Model:** We use a linear regression model to estimate a transformation matrix W that maps the embeddings from Model 1 to those of Model 2:

$$\hat{Y} = X \cdot W$$

where $\hat{Y}$ is the transformed embeddings obtained from Model 1 that should match those of Model 2.

3. **Dimensionality Reduction with PCA:** To visualize the embeddings, we reduce their dimensionality using Principal Component Analysis (PCA). Both the transformed embeddings $\hat{Y}$ and the original embeddings Y are projected into a 2D space by retaining the first two principal components:

$$\hat{Y}_{\text{PCA}} = \text{PCA}(\hat{Y}) \quad Y_{\text{PCA}} = \text{PCA}(Y)$$

4. **Visualization:** We use scatter plots to visualize the relationship between the transformed and the original embeddings, for each PCA component. A red dashed line representing the bisector ($y = x$) is included in both plots to visually assess the alignment of the components.

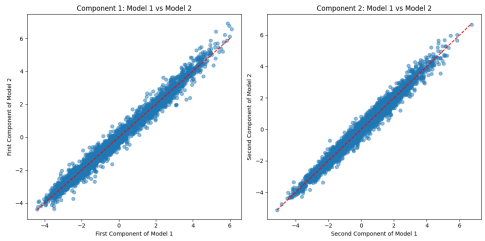

Figure 1: Plot of the PCA components of the original embeddings of Model 2 and the transformed embeddings.

Fig. 1 reveals a strong alignment between the transformed embeddings and the original embeddings from Model 2, as the points are closely distributed along the bisector. This suggests that the linear transformation was highly effective in mapping the embeddings from Model 1 to the embedding space of Model 2. This finding implies that the embedding layers converge to a similar space, that is different from the others apart from a linear transformation. Assuming that the transformation matrix $\mathbf{W}$ has full rank, it represents an endomorphism, meaning that $\mathbf{W}$ is invertible. This implies that the attention mechanism, which applies linear transformations as described in Section 2.1, can effectively learn to align the embeddings, regardless of the specific embedding space to which the layers converge.

As a result, we argue that the position to which the embedding converges is almost independent of the initialization seed. This means that even when the rest of the network is initialized with different seeds, the embeddings tend to converge to similar positions in the embedding space. As a result, changing the seed for the rest of the network has a more significant impact because the representations of the items in the embedding space remain quite similar regardless of the seed, while the rest of the network's components do not. This highlights that the embedding space is relatively stable across different seeds, whereas the rest of the network is more sensitive to the initial seed.

3.2. RQ2

The next step of the analysis is to combine the scores of different models, in order to check if the RLS between combined models shows significant changes.

It is evident from Table 9 that using ensembles significantly improves the RLS compared to individual models, indicating better stability.

Ensemble 1	Ensemble 2	RLS_F@5	RLS_F@10	RLS_F@20
42_43	43_45	0.663	0.705	0.733
42_43	44_45	0.631	0.673	0.702
42_43	43_44	0.646	0.688	0.717
42_43	42_44	0.657	0.698	0.726
42_43	44	0.653	0.695	0.724
AVG ± SD		0.650 ± 0.005	0.692 ± 0.005	0.720 ± 0.005

Table 9
RLS FRBO Improvement with Different Ensembles. The notation x_y represents an ensemble formed by averaging the scores of models initialized with seeds x and y. The last row shows the computation of RLS between an ensemble and a single model.

3.3. RQ3

To investigate the performance, we present Table 10 showing the NDCG scores of the individual models computed @5, @10 and @20.

Seed	NDCG@5	NDCG@10	NDCG@20
42	0.106	0.132	0.157
43	0.105	0.129	0.156
44	0.101	0.127	0.152
45	0.106	0.130	0.156
AVG ± SD	0.105 ± 0.002	0.130 ± 0.002	0.155 ± 0.002

Table 10
NDCG scores for different seeds.

#Models	NDCG_@5	NDCG_@10	NDCG_@20
1	0.105 ± 0.002	0.130 ± 0.002	0.155 ± 0.002
2	0.108 ± 0.001	0.133 ± 0.001	0.160 ± 0.001
3	0.110 ± 0.001	0.135 ± 0.001	0.163 ± 0.001
4	0.110	0.137	0.164

Table 11
NDCG values for ensembles of different sizes. The values are averages computed from all possible model combinations for each number of models, except for the last row which shows the NDCG scores of a single ensemble formed by all four trained models.

In contrast, Table 11 presents the NDCG scores for ensembles of different sizes, with values averaged across all possible model combinations and the final row reflecting the NDCG scores of the ensemble formed by all four trained models. The results demonstrate a slight but consistent improvement in performance with larger ensembles.

4. Considerations on Computational Cost.

In the context of enhancing the robustness of neural recommender systems, the use of an ensemble of models provides significant benefits. However, it also introduces a computational cost that scales linearly with the number of models in the ensemble. Specifically, the computational cost of training an ensemble is approximately the cost of training a single model multiplied by the number of models, denoted as:

$$\text{Cost}_{\text{ensemble}} = N_{\text{models}} \times \text{Cost}_{\text{single\_model}}$$

The same applies to the inference stage. As shown in Tables 9,11, the performance of the system remains relatively stable regardless of the number of models used, while the robustness sees notable improvements when moving from a single model to an ensemble of two models. Given this, a reasonable compromise is to use an ensemble of two models. This choice results in a computational cost of approximately:

$$\text{Cost}_{\text{training}} = 2 \times \text{Cost}_{\text{single\_model}} \times \text{Iterations}$$

for backpropagation during training, and:

$$\text{Cost}_{\text{inference}} = 2 \times \text{Cost}_{\text{single\_model}}$$

for inference during deployment. This allows for a significant improvement in robustness by paying around the double in terms of computational cost.

5. Conclusions

In conclusion, this work addresses the critical issue of ranking variability in recommender systems, which arises due to different model initialization seeds. Our findings suggest that ensemble methods, particularly those incorporating shared embeddings, offer a promising solution to mitigate this variability. By reducing ranking fluctuations, these methods enhance the reliability and consistency of recommendations. However, while our approach significantly reduces variability, some residual variability remains, indicating the need for further research to explore additional techniques or refinements to achieve even greater consistency in recommendations.

References

[1] T. F. Boka, Z. Niu, R. B. Neupane, A survey of sequential recommendation systems: Techniques, evaluation, and future directions, School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China (2024).

[2] F. Betello, A. Purificato, F. Siciliano, G. Trappolini, A. Bacciu, N. Tonellotto, F. Silvestri, A reproducible analysis of sequential recommender systems, IEEE Access 13 (2025) 5762–5772. doi:10.1109/ACCESS.2024.3522049.

[3] M. Quadrana, P. Cremonesi, D. Jannach, Sequence-aware recommender systems, ACM Computing Surveys (CSUR) 51 (2018) 66:1–66:36. doi:10.1145/3190616.

[4] U. Singer, H. Roitman, Y. Eshel, A. Nus, I. Guy, O. Levi, I. Hasson, E. Kiperwasser, Sequential modeling with multiple attributes for watchlist recommendation in e-commerce, in: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), Association for Computing Machinery, New York, NY, USA, 2022, pp. 937–946. URL: https://doi.org/10.1145/3488560.3498453. doi:10.1145/3488560.3498453.

[5] C. Hansen, C. Hansen, L. Maystre, R. Mehrotra, B. Brost, F. Tomasi, M. Lalmas, Contextual and sequential user embeddings for large-scale music recommendation, in: Proceedings of the 14th ACM Conference on Recommender Systems (RecSys '20), Association for Computing Machinery, New York, NY, USA, 2020, pp. 53–62. URL: https://doi.org/10.1145/3383313.3412248. doi:10.1145/3383313.3412248.

[6] Q. Tan, J. Zhang, J. Yao, N. Liu, J. Zhou, H. Yang, X. Hu, Sparse-interest network for sequential recommendation, in: Proceedings of the 14th ACM International Conference on Web Search and Data Mining (WSDM '21), Association for Computing Machinery, New York, NY, USA, 2021, pp. 598–606. URL: https://doi.org/10.1145/3437963.3441811. doi:10.1145/3437963.3441811.

[7] A. Sbandi, F. Siciliano, F. Silvestri, Mitigating extreme cold start in graph-based recsys through reranking, in: Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 4844–4851. URL: https://doi.org/10.1145/3627673.3680069. doi:10.1145/3627673.3680069.

[8] E. D'Amico, G. Gabbolini, C. Bernardis, P. Cremonesi, Analyzing and improving stability of matrix factorization for recommender systems, Journal of Intelligent Information Systems (2022). URL: https://doi.org/10.1007/s10844-021-00658-9. doi:10.1007/s10844-021-00658-9, © The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021.

[9] S. Oh, B. Ustun, J. McAuley, S. Kumar, Rank list sensitivity of recommender systems to interaction perturbations, in: Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM '22), ACM, Atlanta, GA, USA, 2022, pp. 1584–1594. URL: https://doi.org/10.1145/3511808.3557425. doi:10.1145/3511808.3557425.

[10] F. Betello, F. Siciliano, P. Mishra, F. Silvestri, Finite rank-biased overlap (frbo): A new measure for stability in sequential recommender systems, in: Proc. of the 14th Italian Information Retrieval Workshop, volume 3802, 2024, pp. 78–81.

[11] V. Guarrasi, F. Siciliano, F. Silvestri, Robustrecsys @ recsys2024: Design, evaluation and deployment of robust recommender systems, in: Proceedings of the 18th ACM Conference on Recommender Systems, RecSys '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 1265–1269. URL: https://doi.org/10.1145/3640457.3687106. doi:10.1145/3640457.3687106.

[12] B. Filippo, S. Federico, M. Pushkar, S. Fabrizio, Investigating the robustness of sequential recommender systems against training data perturbations, in: Advances in Information Retrieval: 46th European Conference on Information Retrieval (ECIR 2024), Springer, 2024, pp. 205–220. URL: https://doi.org/10.1007/978-3-031-28241-6_14. doi:10.1007/978-3-031-28241-6_14, first Online: 16 March 2024.

[13] W.-C. Kang, J. McAuley, Self-attentive sequential recommendation, arXiv preprint arXiv:1808.09781 (2018). URL: https://arxiv.org/abs/1808.09781.

[14] F. M. Harper, J. A. Konstan, The movielens datasets: History and context, ACM Transactions on Interactive Intelligent Systems (TiiS) 5 (2015) 1–19. URL: https://doi.org/10.1145/2827872. doi:10.1145/2827872, published: 22 December 2015.