SimRec: Mitigating the Cold-Start Problem in Sequential Recommendation by Integrating Item Similarity

SHAKED BRODY, Amazon
SHOVAL LAGZIEL, Amazon

Sequential recommendation systems often struggle to make predictions or take action when dealing with cold-start items that have limited amount of interactions. In this work, we propose SimRec – a new approach to mitigate the cold-start problem in sequential recommendation systems. SimRec addresses this challenge by leveraging the inherent similarity among items, incorporating item similarities into the training process through a customized loss function. Importantly, this enhancement is attained with identical model architecture and the same amount of trainable parameters, resulting in the same inference time and requiring minimal additional effort. This novel approach results in a robust contextual sequential recommendation model capable of effectively handling rare items, including those that were not explicitly seen during training, thereby enhancing overall recommendation performance. Rigorous evaluations against multiple baselines on diverse datasets showcase SimRec's superiority, particularly in scenarios involving items occurring less than 10 times in the training data. The experiments reveal an impressive improvement, with SimRec achieving up to 78% higher HR@10 compared to SASRec. Notably, SimRec outperforms strong baselines on sparse datasets while delivering on-par performance on dense datasets. Our code is available at https://github.com/amazon-science/sequential-recommendation-using-similarity.

CCS Concepts: • **Information systems** → **Recommender systems**; • **Computing methodologies** → *Machine learning*.

1 Introduction

Sequential recommendation systems play a vital role in diverse applications, including e-commerce websites and video streaming platforms, providing valuable recommendations that are tailored to the user's preferences. These systems model each user by past interactions and use this information to predict the next item.

Many efforts have been made to improve modeling methods in sequential recommendations. Hidasi et al. [10] used recurrent neural networks (RNNs) to process user history sequentially. Later, SASRec [12] used the self-attention mechanism of the Transformer architecture [28] to predict the next item. While some approaches make predictions based solely on the sequence of item IDs (i.e. *ID-based methods* [30]), others take advantage of additional contextual features. TiSASRec [15] considered time intervals between actions in the user's history when predicting the next recommendation. CARCA [22] used additional attributes such as the item's brand and other dense features. Furthermore, several attempts incorporated self-supervised learning techniques into their training process. BERT4Rec [26] adopted BERT's [5] pre-training objective in the sequential recommendation setting. Similarly, as done in CARCA, S3-Rec [32] employed item attributes, but in a self-supervised pre-training phase.

In recommendation systems, the cold-start problem, characterized by reduced performance when dealing with items with sparse or no appearances, is a longstanding challenge, particularly in the context of sequential recommendation [7, 27].

Authors' Contact Information: Shaked Brody, Amazon, shakedbr@amazon.com; Shoval Lagziel, Amazon, shovall@amazon.com.

Manuscript submitted to ACM

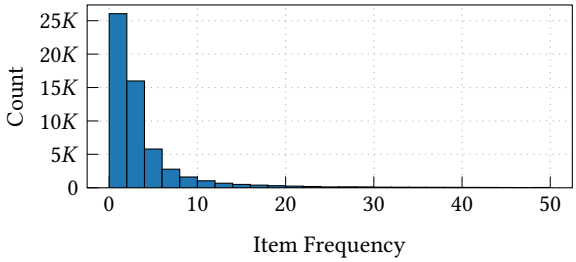

Fig. 1. The Cold-Start Problem: The item frequency of the Amazon Beauty training set. 91% of the items appear less than 10 times.

Figure 1 shows an example of the cold-start problem in the Amazon Beauty dataset [9, 18], in which 91% of the items appear less than 10 times in the training set. This means that most items are infrequent, limiting their contribution to the recommendation system's training and leading to poor performance in predicting them during test time.

To mitigate this limitation, we propose SimRec. Our method leverages the fact that many distinct items are similar to each other. In particular, rare items may share traits with abundant items that are frequently seen during training. For example, consider the following items from the Amazon Beauty dataset, which differ only in size: "NOW Foods Organic Lavender Oil, **1** ounce" and "NOW Foods Organic Lavender Oil, **4** ounce". SimRec exploits this observation by incorporating a *similarity distribution* for each item, describing how similar the item is to other items in the dataset. In this work, we employ a text embedding model [17] to represent the items and subsequently calculate the similarity between them. It is worth noting that there are no technical limitations regarding the choice of similarity metric, offering flexibility for different scenarios and datasets. We introduce a new loss function that incorporates this similarity distribution during training, allowing the model to better learn on cold-start items even if they are not explicitly seen during training[1]. Our approach is simple, requires no additional learnable parameters, and as a result, maintains the same inference time and complexity while significantly enhancing recommendation performance across a wide range of datasets.

Recent studies have tackled the cold-start problem from different perspectives. The Recformer model [16] adopts the Reformer architecture [14], designed to handle long sequences, for recommendation purposes. Positioned as a *Text-Only Method*, Recformer processes sentence inputs for each item, leveraging language representations. Although it has demonstrated superior performance, it comes with a trade-off in computational complexity compared to *ID-based methods*. SimRec inference time is identical to an *ID-based method* (i.e., SASRec[12]). Its computational cost is evident in the similarity calculation phase which happens once before training and in the training phase during the computation of our new loss. Several works have explored the use of meta-learning frameworks to address the cold-start problem [11, 31]. However, this line of works differ from ours as they focus on scenarios where a history of interactions and a small set of cold-start items are provided during inference, enabling the system to recommend one item from this limited set. In contrast, we work in a distinct setting, where given a user's history, the objective is to recommend an item from the entire item pool. This approach allows us to predict both rare and abundant items, offering a comprehensive solution to the cold-start problem in recommendation systems. A few works tackled the cold-start problem within domain specific datasets [1, 3, 4, 25]. Our approach is applicable to a wider array of domains. It only requires item similarity data, which can be calculated based on basic features such as item title that are commonly available. This eliminates the need for labor-intensive feature curation specific to each dataset.

[1]By saying "not explicitly seen" we mean that the item has no interactions with users in the training data, but is included in the item similarity data.

2 Cold-Start Items and Density

Cold-start items are items that have few interactions in the dataset or are entirely absent, meaning they have been rarely interacted by users. We define the *density* of a dataset as the average number of interactions per item:

$$Density(\mathcal{D}) = \frac{\sum_{u \in \mathcal{U}} |u|}{|\mathcal{V}|} = \frac{\#interactions}{\#items} \tag{1}$$

Where $\mathcal{U}$ is the set of users, $|u|$ is the number of interactions per user, and $\mathcal{V}$ is the set of items. We can say that the sparsity of a dataset increases when its density decreases, i.e., a dataset is said to be sparse when its density value is relatively small.

An important observation is that although sparse datasets contain many rare items, many of them are similar. Table 1 shows an example of an item from the Amazon Beauty dataset and its most similar items. Note that while the item is relatively frequent, it is similar to rare items. This observation can be used to improve performance on rare items, ultimately leading to overall performance improvement, especially on sparse datasets. In Section 3, we explain how we measure the similarities between items and how we incorporate this information into the training process.

Table 1. Example of an item and its most similar items from the Beauty dataset.

	Item	Frequency
	Olay Pro-X Advanced Cleansing System 0.68 Fl Oz, 1-Count	**351**
k=1	Olay Pro-X Microdermabrasion Plus Advanced Cleansing System Refill, 1-Count, 1.7 fl oz	8
k=2	Olay Professional Pro-X Advanced Cleansing System Set	12
k=3	Olay Professional Pro-X Restorative Cream Cleanser, 5 Ounce	1
k=4	Olay Pro-X Microdermabrasion Plus Advanced Cleansing System, 1-Kit	63
k=5	Olay Age Defying Classic Cleanser , 6.8-Fluid Ounce (Pack of 4)	2

3 Methodology

In this section, we elaborate on SimRec – our proposed approach. We begin by describing the computation of item similarities. Then, we outline how SimRec incorporates similarity distributions during the training process, utilizing a new loss function.

3.1 Item Similarity Distribution

SimRec requires a similarity metric between pairs of items, which is then used to produce the similarity distribution for each item. One way to find item similarities is by considering textual features, and calculating the similarity between their dense representation. Large Language Models (LLMs) have demonstrated promising results in many Natural Language Processing (NLP) tasks such as language modeling, question answering and sentiment analysis [2, 20, 21]. The notion of generating dense text representations has recently gained significant attention, as these representations hold great potential for various tasks such as semantic search, text retrieval and text similarity [23, 24].

Specifically, our process involves applying a pre-trained text embedding model to the item titles, resulting in a dense representation for each item. Subsequently, we compute the similarity between two items by utilizing the cosine similarity between their respective representations. Notably, while we employed text for similarity calculations, our method is not limited to this modality. Without inherent constraints, our suggested loss can be applied to any modality and any similarity function.

Initially, we generate the embedding for each item title, denoted as $\boldsymbol{v}_i$. Then, we proceed to assess the similarities of the i^{th} item with all other items in the dataset, resulting in a similarity vector $\boldsymbol{s}^i \in \mathbb{R}^{|\mathcal{V}|}$. Then, to create a valid distribution, we apply the softmax function to $\boldsymbol{s}^i$ resulting in $\widehat{\boldsymbol{s}^i}$ that describes how similar the i^{th} item is to the other items. We incorporate these distributions during the training process, as we explain next.

3.2 New Loss Function

Since our model architecture is equivalent to SASRec [12], we refer the reader to Kang and McAuley [12] for more details.

Given a user $u \in \mathcal{U}$ and the user-item interaction history sequence $h_u = \left[a_1^u, a_2^u, ..., a_{|u|}^u\right]$, the model outputs the sequence $\left[\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_{|u|}\right] \in \mathbb{R}^{d \times |u|}$ such that d is the hidden size and $\boldsymbol{f}_t$ corresponds to the t^{th} item of the input sequence. To determine the relevance score of an item $i \in \mathcal{V}$ as the next item after the t^{th} item of the sequence h_u (considering the initial t items), we multiply $\boldsymbol{f}_t$ with $\boldsymbol{E}_i$ (the embedding of i^{th} item that is been learned during training) which results in $r_i^t = \langle \boldsymbol{f}_t, \boldsymbol{E}_i \rangle$. A high score indicates a high level of relevance. Based on the ranking of these scores, we can provide recommendations.

SimRec uses a new loss function $\mathcal{L}_{\text{SimRec}}$ that combines (a) binary cross-entropy loss, and (b) *similarity* loss. We first describe the binary cross-entropy loss which is commonly used in training sequential recommendation systems [12, 15], then our proposed *similarity* loss, and finally $\mathcal{L}_{\text{SimRec}}$ that combines the two.

We define $h_u^+ = \left[a_2^u, ..., a_{|u|}^u\right]$ to be the positive sequence that represents the next true item for any prefix of h_u, as the left-shifted sequence h_u. We also define $h_u^- = \left[a_1^{-u}, ..., a_{|u|-1}^{-u}\right], \forall i, a_i^{-u} \notin h_u$ to be the negative sequence that consists of items that are not part of h_u, which are selected randomly during training. Now, we define the binary cross-entropy loss as follows:

$$\mathcal{L}_{BCE} = -\sum_{u \in \mathcal{U}} \sum_{t=1}^{|u|-1} \left[log\left(\sigma\left(r_{a_t^{+u}}^t\right)\right) + log\left(1 - \sigma\left(r_{a_t^{-u}}^t\right)\right)\right] \tag{2}$$

where σ is the sigmoid function.

To incorporate item similarity data, we define *similarity* loss as the cross-entropy loss between the model output relevancy distribution over all items and the similarity distribution.

We denote $\boldsymbol{r}^t = [r_1^t, ..., r_{|\mathcal{V}|}^t]$ as the model output relevancy scores for the next item after the t^{th} element of h_u. To maintain a distribution, we apply the softmax function to $\boldsymbol{r}^t$ resulting in $\widehat{\boldsymbol{r}^t}$. We denote $\widehat{\boldsymbol{s}^{a_t^{+u}}}$ as the similarity distribution of the next true item a_t^{+u}.

We define the similarity loss as follows:

$$\mathcal{L}_{Similarity} = -\sum_{u \in \mathcal{U}} \sum_{t=1}^{|u|-1} \sum_{i=1}^{|\mathcal{V}|} \left[\widehat{s_i^{a_t^{+u}}} \cdot log\left(\widehat{r_i^t}\right)\right] \tag{3}$$

The final loss function we optimize is:

$$\mathcal{L}_{\text{SimRec}} = (1 - \lambda) \cdot \mathcal{L}_{BCE} + \lambda \cdot \mathcal{L}_{Similarity}, \quad \text{where } \lambda \in [0, 1] \tag{4}$$

By using the similarity distribution during training, we actually incorporate new information into the model compared to other baselines such as SASRec [12]. In Section 4.4, we explore an alternative approach for integrating this information into the model through the item embedding, and demonstrate that the loss function we have developed yields superior results.

4 Experiments

We conduct a comprehensive analysis of our model's effectiveness, exploring: (a) the overall recommendation performance, (b) the cold-start performance, and (c) the correlation between the density of the datasets and the performance improvement facilitated by our approach. Finally, we present an ablation study analyzing the different components of our approach.

In all our experiments, we compute similarity between embeddings generated by the GTE-Large model [17] applied to item titles. Further details on datasets and implementation can be found in Appendices A and C.

Table 2. SimRec outperforms other strong baselines on sparse datasets, and achieves on-par performance on dense datasets. (*) marks statistical significance improvement compared to the second best model, with $p < 0.01$.

Dataset	Metric	TopPop	SASRec	TiSASRec	BERT4Rec	CARCA	SimRec	SimRec vs. SASRec	Density
Tools	HR@10	0.444	0.407	0.409	0.410	0.501	**0.517***	27.03%	6.2
	NDCG@10	0.261	0.252	0.252	0.242	0.311	**0.333***	32.14%	
Beauty	HR@10	0.460	0.461	0.437	0.455	0.559	**0.594***	28.85%	6.9
	NDCG@10	0.263	0.311	0.289	0.294	0.374	**0.425***	36.70%	
Pet Supplies	HR@10	0.483	0.542	0.536	0.536	0.618	**0.632***	16.61%	9.1
	NDCG@10	0.283	0.356	0.347	0.338	0.399	**0.419***	17.70%	
Home & Kitchen	HR@10	0.527	0.498	0.480	0.458	0.529	**0.565***	13.45%	9.2
	NDCG@10	0.318	0.303	0.293	0.267	0.320	**0.363***	19.80%	
ML-1M	HR@10	0.440	0.819	0.804	0.802	0.608	**0.827***	0.97%	292.6
	NDCG@10	0.241	0.594	0.576	0.603	0.351	**0.604**	1.68%	
Steam	HR@10	0.760	0.862	0.857	0.856	0.828	**0.871**	1.04%	322.9
	NDCG@10	0.500	0.639	0.635	0.645	0.610	**0.651***	1.88%	

4.1 Recommendation Performance

Our evaluation covers both dense datasets (ML-1M, Steam [8, 12]) and sparse datasets (Tools, Beauty, Pet Supplies and Home & Kitchen [9, 18]). We compare SimRec performance to one naïve baseline (TopPop) and four sequential recommendation models. The latter include two *ID-based methods* [30] (SASRec [12] and BERT4Rec [26], representing items with IDs which are subsequently converted to embeddings), and two *ID-based methods with additional context* (TiSASRec [15] and CARCA [22]), such as SimRec.

Table 2 shows the results of our experiments. SimRec outperforms on the sparse datasets – Tools, Beauty, Pet Supplies and Home & Kitchen, and is on-par with other strong-baselines [12, 15, 22, 26] on the dense datasets – ML-1M and Steam.

CARCA [22], which utilizes contextual features alongside item embeddings, provides competitive results on the sparse datasets, however, it achieves relatively lower performance on the dense datasets – ML-1M and Steam. Nevertheless, SimRec excels on sparse datasets and achieves on-par performance as the other strong baselines on dense datasets. This suggests that SimRec is compatible with a wider range of datasets compared to the other baselines, and CARCA in particular.

Since the architecture of SimRec is identical to that of SASRec [12] and our approach differs in the optimization process, we show the relative improvement of SimRec compared to SASRec in Table 2. Notably, SimRec achieves up to **28.85%** HR@10 improvement and up to **36.7%** NDCG@10 improvement over SASRec.

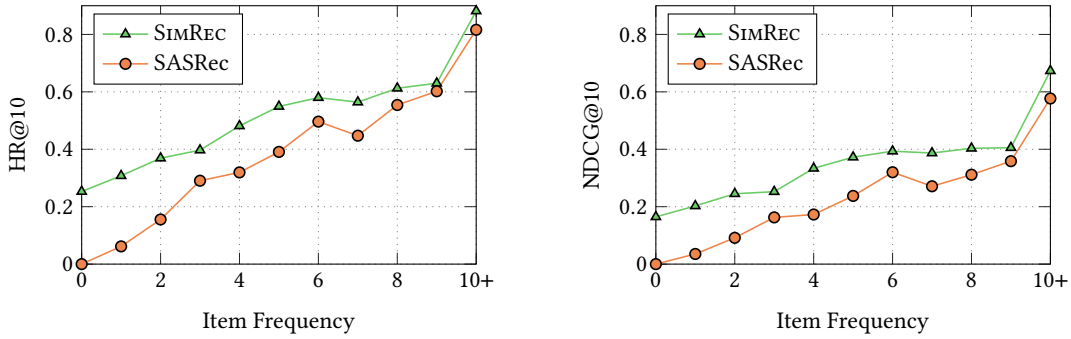

Fig. 2. Cold-start performance analysis reveals that SimRec excels on rare items, showcasing improvements attributed to enhanced performance in such scenarios, particularly evident in the Beauty dataset. SimRec achieves a remarkable **78%** higher HR@10 and a **101%** higher NDCG@10 compared to SASRec [12] for items that appear less than 10 times in the training set (accounting for 61% of the test set).

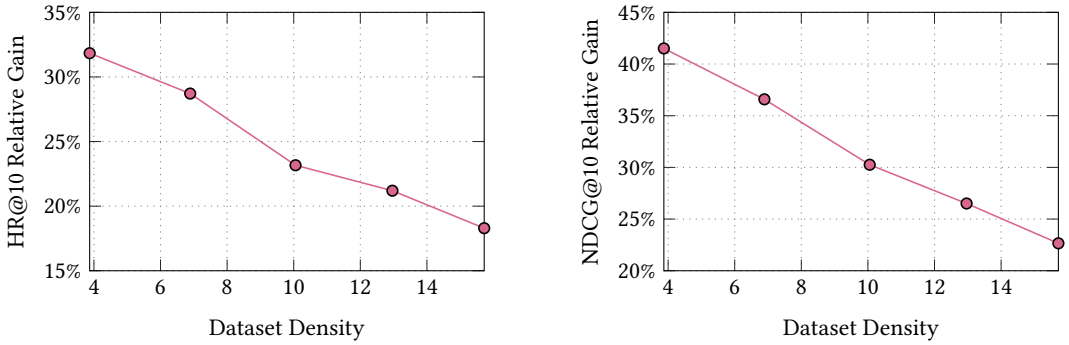

Fig. 3. Relative performance gain across various dataset densities: SimRec exhibits a higher relative gain over SASRec [12] with the sparser variants of the Beauty dataset, which decreases as dataset density increases.

4.2 Cold Start Performance

Our evaluation suggests that SimRec's competitive performance arises from its capacity to enhance recommendations for cold-start items, which are more common in sparse datasets. In this section, we compare the performance of SimRec and SASRec [12], both sharing the same architecture, specifically focusing on cold-start items. Figure 2 shows the performance of SimRec and SASRec on the Beauty dataset [9, 18]. Each point shows the HR@10 for samples in the test set where the true next item has a specific frequency (calculated on the training set).

First, it is worth noting that SimRec outperforms SASRec on cold-start items with low frequency. The largest difference is evident when considering the rarest items that occur less than 10 times in the training set. SimRec achieves

78% higher HR@10 and **101%** higher NDCG@10 compared to SASRec for such items, accounting for 61% of the test set (where SimRec achieves 0.405 HR@10 and 0.269 NDCG@10 compared to 0.227 and 0.134 achieved by SASRec, respectively). For more abundant items, i.e., those occurring 10 times or more, the improvement is less significant, with a relative improvement of 8% HR@10 and 16.6% NDCG@10 (0.882 HR@10 and 0.673 NDCG@10 for SimRec, compared to 0.816 and 0.577, respectively, achieved by SASRec). In addition, SimRec demonstrates the ability to predict items not explicitly seen during the training process (item frequency = 0), achieving a HR@10 of 0.253 and an NDCG@10 of 0.165, while SASRec fails completely with an HR@10 and an NDCG@10 of 0. Notable improvements with SimRec are evident for other sparse datasets and can also be found in Appendix E.

4.3 Density vs. Performance

To investigate the relationship between the density of a dataset and the performance of SimRec, we generated several variants of the Beauty dataset [9, 18], with different densities. Previous work [12, 15, 22] has eliminated items occurring less than $n = 5$ times in the raw dataset. We repeated this pre-processing method, and used different values of n to derive dataset variants with different densities. The statistics of the datasets can be found in Appendix F.

Figure 3 shows the relative performance gain of SimRec over SASRec [12] on the different variants of the Beauty dataset. We choose to present the relative gain since comparing the HR@10 and NDCG@10 between different datasets with different number of items has an effect on the evaluation process. We find that the relative gain increases as the dataset becomes sparser. Thus, the advantage of SimRec over SASRec decreases as the dataset becomes denser. For example, for the densest dataset ($n = 20$), SimRec achieves an improvement of 18.3% and 22.7% for HR@10 and NDCG@10, respectively, while for the sparsest dataset ($n = 0$), SimRec achieves an improvement of 31.8% and 41.5% for HR@10 and NDCG@10, respectively. These findings align with the results in Table 2, where SimRec shows its most significant performance improvements on sparse datasets.

Table 3. Ablation study results.

Model	HR@10	NDCG@10
SASRec	0.461	0.311
SASRec + embeddings	0.563	0.390
SASRec + embeddings + $\mathcal{L}_{\text{SimRec}}$	0.591	0.421
SimRec (SASRec + $\mathcal{L}_{\text{SimRec}}$)	**0.594**	**0.425**

4.4 Ablation Study

We examine the individual contributions of various components of our approach. We conducted experiments using the following models on the Beauty dataset: (1) **SASRec** [12], (2) **SASRec + embeddings**: the item embeddings are initialized with the embeddings generated by GTE-Large [17], (3) **SASRec + embeddings + $\mathcal{L}_{\text{SimRec}}$**: employing our loss, and (4) **SimRec**. Table 3 summarizes the results. Initializing SASRec embeddings with the GTE-Large embeddings (2) results in a remarkable performance improvement of over 20% HR@10 and 23% NDCG@10 compared to SASRec (1) that uses the Glorot initialization method [6] for the embedding weights. Using the embeddings in this way is similar to CARCA [22] in the sense that additional data is explicitly introduced into the model.

Using our proposed method (4) yields superior performance compared to embedding initialization (2). This approach (4) results in a relative gain of over 28% in HR@10 and 36% in NDCG@10 compared to SASRec (1). Using both embedding initialization and our new loss (3) leads to similar performance as using the new loss alone (4). This indicates that our

new loss is the main contributor to the performance gain. Results regarding the choice of a text embedding model for item similarity are detailed in Appendix B.

5 Conclusions

In this work we present SimRec, a new approach to mitigate the cold-start problem in sequential recommendation systems. SimRec exploits the fact that many different items in a dataset are similar to each other, and incorporates similarity information into the training phase using our proposed loss function, $\mathcal{L}_{\text{SimRec}}$.

SimRec outperforms strong sequential recommendation systems like SASRec [12], maintaining a similar architecture, thus resulting in the same inference time. Furthermore, it effectively tackles the cold-start problem for sparse datasets, significantly enhancing recommendation performance for items with less than 10 occurrences in the training set, resulting in up to a 78% improvement in HR@10. Notably, SimRec was even able to recommend accurately items that were not encountered explicitly during training.

Future research could explore integrating the proposed loss function into diverse sequential recommendation models, alongside investigating alternative similarity functions and diverse data modalities to enhance recommendation performance.

In summary, our approach represents a significant advancement in tackling the challenge of cold-start items with minimal added efforts and maintaining the same inference time as other sequential recommendation systems. Notably, it excels in addressing the cold-start problem in sparse datasets, marking a promising direction for future research and applications in recommendation systems.

Acknowledgments

We would like to thank our team and in particular Iftah Gamzu and Yonathan Aflalo for the helpful discussions.

References

[1] Y. Bi, L. Song, M. Yao, Z. Wu, J. Wang, and J. Xiao. Dcdir: A deep cross-domain recommendation system for cold start users in insurance domain. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 1661–1664, 2020.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] S.-Y. Chou, Y.-H. Yang, J.-S. R. Jang, and Y.-C. Lin. Addressing cold start for next-song recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 115–118, 2016.

[4] Y. Deldjoo, M. F. Dacrema, M. G. Constantin, H. Eghbal-Zadeh, S. Cereda, M. Schedl, B. Ionescu, and P. Cremonesi. Movie genome: alleviating new item cold start in movie recommendation. *User Modeling and User-Adapted Interaction*, 29:291–343, 2019.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[7] J. Gope and S. K. Jain. A survey on solving cold start problem in recommender systems. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 133–138, 2017. doi: 10.1109/CCAA.2017.8229786.

[8] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[9] R. He and J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.

[10] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[11] X. Huang, J. Sang, J. Yu, and C. Xu. Learning to learn a cold-start sequential recommender. *ACM Transactions on Information Systems (TOIS)*, 40(2): 1–25, 2022.

[12] W.-C. Kang and J. McAuley. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pages 197–206. IEEE, 2018.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] N. Kitaev, Ł. Kaiser, and A. Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[15] J. Li, Y. Wang, and J. McAuley. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th international conference on web search and data mining*, pages 322–330, 2020.

[16] J. Li, M. Wang, J. Li, J. Fu, X. Shen, J. Shang, and J. McAuley. Text is all you need: Learning language representations for sequential recommendation. *arXiv preprint arXiv:2305.13731*, 2023.

[17] Z. Li, X. Zhang, Y. Zhang, D. Long, P. Xie, and M. Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

[18] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

[19] J. Ni, G. H. Ábrego, N. Constant, J. Ma, K. B. Hall, D. Cer, and Y. Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *arXiv preprint arXiv:2108.08877*, 2021.

[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[21] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[22] A. Rashed, S. Elsayed, and L. Schmidt-Thieme. Context and attribute-aware sequential recommendation via cross-attention. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 71–80, 2022.

[23] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[24] N. Reimers and I. Gurevych. The curse of dense low-dimensional information retrieval for large index sizes. *arXiv preprint arXiv:2012.14210*, 2020.

[25] J. Ren, J. Long, and Z. Xu. Financial news recommendation based on graph embeddings. *Decision Support Systems*, 125:113115, 2019.

[26] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

[27] S. A. Thorat, G. Ashwini, and M. Seema. Survey on collaborative and content-based recommendation systems. In *2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 1541–1548, 2023. doi: 10.1109/ICSSIT55814.2023.10061072.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[29] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788, 2020.

[30] Z. Yuan, F. Yuan, Y. Song, Y. Li, J. Fu, F. Yang, Y. Pan, and Y. Ni. Where to go next for recommender systems? id-vs. modality-based recommender models revisited. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2639–2649, 2023.

[31] Y. Zheng, S. Liu, Z. Li, and S. Wu. Cold-start sequential recommendation via meta learner. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4706–4713, 2021.

[32] K. Zhou, H. Wang, W. X. Zhao, Y. Zhu, S. Wang, F. Zhang, Z. Wang, and J.-R. Wen. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 1893–1902, 2020.

Table 4. SimRec hyper parameters

Parameter	Tested Values	Best Value					
		Tools	Beauty	Pet Supplies	Home & Kitchen	ML-1M	Steam
Learning rate	[1e-4, 2.5e-4, 5e-4, 1e-3]	1e-4	1e-4	1e-4	1e-4	1e-3	1e-4
Hidden size	[50, 100]	50	100	50	50	100	100
# layers	[2, 3]	3	3	2	3	3	2
Similarity threshold	[0.4, 0.5,..., 1]	0.4	0.9	0.7	0.6	0.5	0.6
Softmax temperature	[0.5, 1, 1.5,..., 4]	0.5	0.5	0.5	0.5	1	1.5
λ	[0.1, 0.2,..., 0.9]	0.8	0.3	0.6	0.7	0.6	0.2
Dropout	[0.2, 0.5]	0.5	0.5	0.5	0.5	0.5	0.5

A Implementation Details

To calculate item embedding for similarity calculation, we used the General Text Embedding Large model (GTE-Large) [17] – a pre-trained text embedding model that has been trained with both unsupervised and supervised contrastive objectives to produce dense text representations. Once we have an embedding vector for each item, we compute the similarities between all items. Please note that although calculating this process has a quadratic complexity (with the number of items), it occurs only once before training. In practical terms, it takes a relatively short time, less than 1 hour for all six tested datasets, each containing up to 100,000 items, on a typical machine. This process results in $|\mathcal{V}|^2$ similarity scores, which can be large for some datasets, therefore, we keep only the 1K most similar items for each item. To reduce noise from less similar items, we also use similarity threshold to filter out items with low similarity. To maintain a valid similarity distribution over all items $\in \mathcal{V}$, we assign negative infinity to the indices of the filtered out items before applying the softmax function. We use linear scheduling with warm-up for the hyper parameter λ that controls the weight of each component of our loss function. This approach allows the training process to initially optimize both loss components ($\mathcal{L}_{BCE}$ and $\mathcal{L}_{Similarity}$) and gradually decrease λ, assigning more weight to $\mathcal{L}_{BCE}$.

We used Adam optimizer [13] with $\beta = (0.9, 0.98)$, 210 training epochs, batch size of 128, and 1000 warm-up steps for the linear scheduling of λ. We used a maximum sequence length of 200 for ML-1M, and 50 for the other datasets. The best hyper parameters we found according to the validation set are listed in Table 4. When training the other baselines, we used the hyper parameters reported by the authors.

CARCA [22] requires additional categorical features. For the Beauty, Tools, Pet Supplies, and Home & Kitchen datasets, we used the price, brand, and categories to which the item belongs. For the ML-1M dataset, we used the year of release and the genres of the movie. For the Steam dataset we used the price, developer and genres.

Table 5. Recommendation performance with different text embedding models for item similarity.

Text Embedding Model	HR@10	NDCG@10
Sentence-T5-XXL	0.571	0.385
all-MiniLM-L6-v2	0.581	0.410
GTE-Large	**0.594**	**0.425**

B Text Embedding Method

To find similarities between items, we use a pre-trained text embedding model that produces high-dimensional text embeddings. To determine the most suitable model, we performed a hyper parameter search (as shown in Table 4) on the Beauty dataset using three different models: (a) GTE-Large [17], (b) all-MiniLM-L6-v2 [29], and (c) Sentence-T5-XXL [19]. Table 5 shows the results of the three models. Since GTE-Large produced the best result, we chose to use it for the other datasets we experimented with.

Table 6. Datasets Statistics

	Users	Items	Interactions	Density
Tools	45,509	51,186	0.31M	6.2
Beauty	52,133	57,226	0.4M	6.9
Pet Supplies	33,571	27,447	0.25M	9.1
Home & Kitchen	114,724	93,358	0.86M	9.2
ML-1M	6,040	3,416	1M	292.6
Steam	334,634	13,045	4.2M	322.9

C Datasets

We consider six different datasets – four of them are part of the Amazon reviews dataset [9, 18], and the other two consist of movie reviews and video game reviews.

(1) **Tools** [9, 18]: contains items and reviews from the Tools and Home Improvement category.
(2) **Beauty** [9, 18]: contains items and reviews from the Beauty category.
(3) **Pet Supplies** [9, 18]: contains items and reviews from the Pet Supplies category.
(4) **Home and Kitchen** [9, 18]: contains items and reviews from the Home and Kitchen category.
(5) **ML-1M** [8]: contains 1M reviews from the MovieLens website.
(6) **Steam** [12]: contains about 4M reviews from the Steam video games platform.

Following previous work [12, 15, 22], we pre-process the raw datasets and remove items that occur less than $n = 5$ times, and then remove users that have less than 5 interactions. Note that this pre-processing may lead to items appearing even fewer than n times in the dataset due to the removal of certain users. We also remove items that do not have a title. This had a minor effect of removing only 1.3% of the items on average. The statistics for the processed datasets can be found in Table 6.

D Evaluation Metrics

To evaluate our approach and other baselines we use the commonly used HR@10 and NDCG@10 metrics. Following previous work [12, 15, 22], for each user interaction history $h_u \in \mathcal{D}$, we consider the first $|u| - 2$ items for training, the $|u| - 1$ item for validation, and the last item for testing. Consistent with previous work, we randomly select 100 negative items during the evaluation process given the next true item. Then, we use the model to rank these items and record the HR@10 and NDCG@10 metrics. Following Rashed et al. [22], to reduce the effect of random picking, we repeat the evaluation process 5 times, each time with a set of 100 distinct randomly chosen negative items, and report the average metrics. These evaluation scores are also used to calculate statistical significance compared to the baselines.

E Cold-Start Performance

Figures 4 to 7 show the cold-start improvement obtained by SimRec over SASRec [12]. On these sparse datasets, SimRec exhibits better performance on cold-start items and in particular, items that are not explicitly seen during training. Figures 8 and 9 show the performance on the dense datasets – ML-1M and Steam. For these dense datasets, no significant improvement of one model over the other is evident on cold-start items. However, in these datasets, less than 1.5% of the test set samples correspond to rare items that appear less than 10 times in the training set.

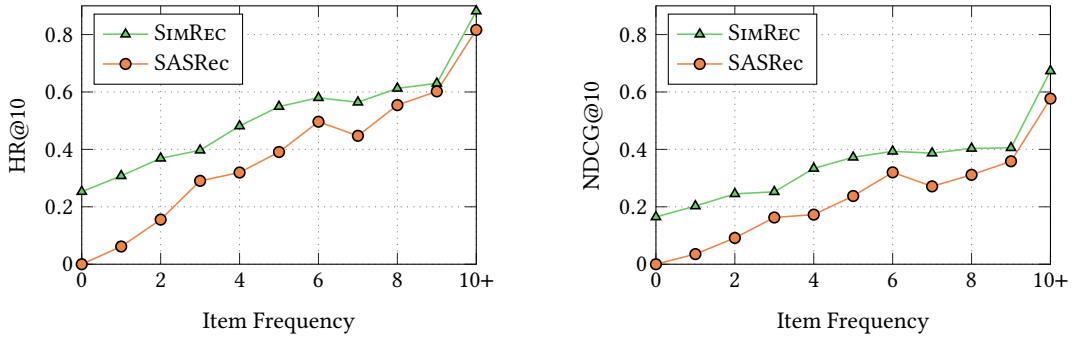

Fig. 4. Cold-start performance on the Beauty dataset. Datapoints with item frequencies less than 10 collectively account for 61% of the test set.

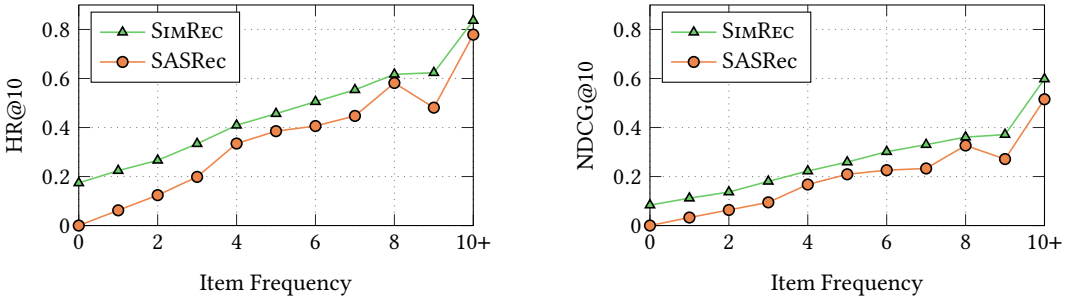

Fig. 5. Cold-start performance on the Tools dataset. Datapoints with item frequencies less than 10 collectively account for 64% of the test set.

F Density vs. Performance – Additional Information

The statistics of the dataset variants we generated in Section 4.3 can be found in Table 7. The exact metric values of the results shown in Figure 3 can be found in Table 8.

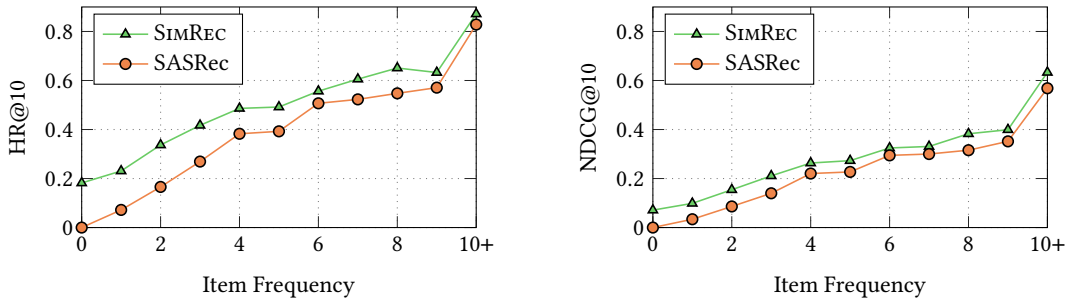

Fig. 6. Cold-start performance on the Pet Supplies dataset. Datapoints with item frequencies less than 10 collectively account for 51% of the test set

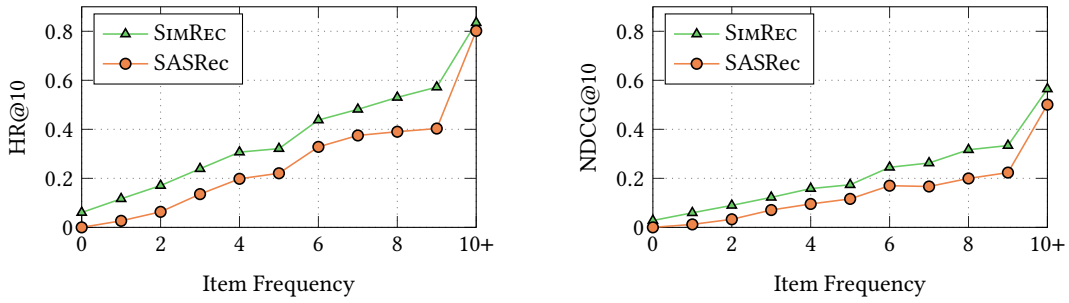

Fig. 7. Cold-start performance on the Home & Kitchen dataset. Datapoints with item frequencies less than 10 collectively account for 47% of the test set.

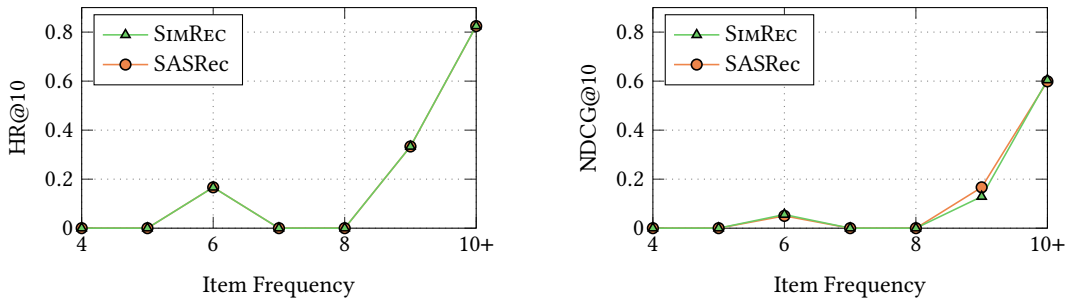

Fig. 8. Cold-start performance on the ML-1M dataset. Datapoints with item frequencies less than 10 collectively account for 0.3% of the test set.

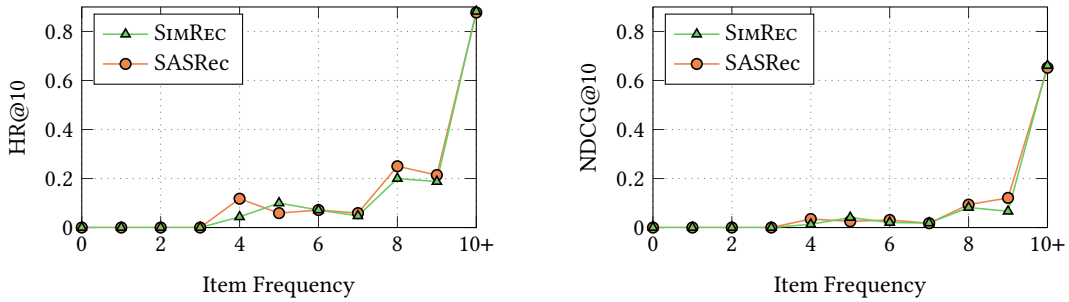

Fig. 9. Cold-start performance on the Steam dataset. Datapoints with item frequencies less than 10 collectively account for 1.5% of the test set.

Table 7. Variants of the Beauty dataset with different densities. Items appearing less than n times are excluded from the dataset. $n = 5$ corresponds to the dataset we present in Appendix C.

n	Users	Items	Interactions	Density
0	52,302	121,144	0.47M	3.9
5	52,133	57,226	0.4M	6.9
10	51,817	34,784	0.35M	10.1
15	51,433	24,618	0.32M	13.0
20	51,004	18,764	0.3M	15.7

Table 8. Recommendation performance on variable density levels of the Beauty dataset. Items that appear less than n times in the raw dataset are discarded.

n	Metric	SASRec	SimRec	Improve vs. SASRec	Density
0	HR@10	0.455	0.599	31.84%	3.9
	NDCG@10	0.310	0.439	41.51%	
5	HR@10	0.461	0.594	28.85%	6.9
	NDCG@10	0.311	0.425	36.70%	
10	HR@10	0.463	0.571	23.16%	10.1
	NDCG@10	0.301	0.403	30.26%	
15	HR@10	0.467	0.566	21.19%	13.0
	NDCG@10	0.314	0.397	26.51%	
20	HR@10	0.470	0.556	18.29%	15.7
	NDCG@10	0.315	0.387	22.65%	