Efficient Method for Future Items Inference in Structured Multi-Item Recommendations

Mari Ashiga

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2023

Abstract

This thesis is the first to describe a method for predicting the order of a set of items a user will purchase based on the user's purchase history. We refer to this method as future items inference. The proposed method can be used for dense data models with bias-corrected mixed negatives with the time feature and sparse data models. We demonstrate the method performance with three comparisons using movie and product review data: 1: the next-item inference models for the multi-item predictions, 2: next-item inference vs. future items inference, and 3: all-item prediction vs. dense-all-item prediction. The method involves inferring the future items based on the last action followed by optimizing the relevance scores for each item position with mixed negative sampling and Sampled Softmax Loss. The dominant part of the method is multiple target pairing, automatically selecting the structured target items for every input item at each time step without random target sampling. Our method drastically improves accuracy compared to the conventional next-item inference method. The latter, in addition to being inefficient and inaccurate, can be erroneous when analyses require output items to be in a particular order.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Mari Ashiga*)

Acknowledgements

I am grateful to my supervisors, David Wardrope, Timos Korres, Thomas Baumhauer, and Michael la Grange, for their invaluable discussions and feedback on this dissertation, starting from the IPP. I sincerely thank Professor Iain Murray for all his support and sophisticated course and work for this program, which allowed me to complete this exciting project. I am also grateful to Ali Eslami for his early inspiration in my machine learning research through his consistent belief in the evolution of AI. I thank my family, Emi Taketatsu, Michael Omoto, Ingram Weber, and Tetsuji Kanamori, for their generous support that led me to this degree program.

Table of Contents

1 Introduction **1**

2 Background **5**

2.1 Training and testing of SASRec 5

2.2 Time feature encoding 7

2.3 Based metrics 8

3 Overview of method **10**

3.1 Next item inference 10

3.2 Future items inference 11

3.3 Metrics 16

4 Experiments with discussions **18**

4.1 Application: Movie/product review 18

4.2 Implementation details 20

4.3 Predict future items with next item inference 20

4.3.1 Next-item and all-item predictions procedure 20

4.4 Predict future items with future items inference 24

4.4.1 FutureAllItem and mixed negative sampling procedure 24

4.4.2 Comparison of our method to NextAllItem 25

4.4.3 Limitations of the model and our inference method 27

4.4.4 Effect of mixed negative sampling 28

4.4.5 Effect of window size 31

4.4.6 FutureDenseAll and time feature incorporation procedure . . 31

4.4.7 Comparison of densely trained models to FutureAllItem . . . 32

4.4.8 Limitation of the time feature model 34

5 Conclusions **36**

Bibliography **38**

Chapter 1

Introduction

In recommendation systems, many applications have been formulated as machine-learning tasks, solving a typical classification task that predicts a user's preference from the user's behavior history. Recommendation systems cover a wide range of research topics in information retrieval, such as general recommendation (predicting items from user feedback, e.g., clicks, purchases, likes), temporal recommendation (modeling timestamp of users' activities), and sequential recommendation (capturing sequential patterns from an interaction sequence or item-item transition matrices). These topics have the common goal of predicting user preferences and differ in the forms of inputs.

Matrix Factorization (MF) and Collaborative Filtering (CF) are popular methods for general recommendation [22] and temporal recommendation [14], respectively. MF predicts user preference using the inner product between user and item embeddings. Successful CF models often build on MF and can represent temporal patterns of time features, using a matrix of ratings, and can handle concept drift (e.g., change of movie preferences over time). MF and CF are sensitive to data sparsity compared to sequential methods like Markov Chains (MCs) and Recurrent Neural Networks (RNNs). MCs and RNNs are typically used to represent latent attributes (e.g., state of user or movie) given item-item transition matrices in capturing sequential patterns [21, 9]. In MC, a first-order MC assumes the next action is related to a previous one, while higher-order MCs does it is related to several previous ones. While this assumption strongly works on sparse datasets, it fails when a complex scenario easily violates it. In RNNs, the latent variables of the features summarize all previous actions to improve expressiveness, but it requires large amounts of data to perform well the next-action prediction. Recently, attention mechanisms have emerged for sequence modeling and transduction models in various

tasks and for interpretation analysis of results. They focus on relevant parts of some input successively without distance constraints. Inspired by transformer [24], self-attention models such as [11, 15] perform better than the conventional shallow/complex structures (e.g., MF, RNNs).

Advances over the last decade in sequential methods and their implementation in open-source software have drastically improved sequential modeling for applied researchers. For example, with sequential recommendation models such as SASRec [11], we can predict the next item a user would buy given a sequence of purchased items with efficient training and better accuracy. For its primary advantage of features, SASRec trains the model faster than other sequential models by using transformer [24] and efficiently selecting a *(input item, next item)* pair in their next-item inference. This next-action recommendation with transformer is a self-attention-based prediction method that has become ubiquitous in recommendation systems research over the last years due to its being efficient, accurate, and general.

Despite its widespread use, the next-action recommendation methods can have two drawbacks in sequential problems — they can be inaccurate in future actions recommendation and sensitive to data split design. For example, suppose predicting future n items using the next-item predictor such as SASRec or GRU4Rec [11, 10]. These next-item inference methods can quickly amplify mistakes made early by feeding them as input to the model because the model might be in a part of the state space it has never seen at training time. To make matters worse, the data splitting strategy changes the number of actions in user sequences and can impact performance [17].

An alternative method using PINNERFORMER [20] for future actions recommendation has been proposed for problems where the next-action recommendation is impractical. This approach involves future multi-action prediction by selecting multiple *(input action, target action)* pairs by sampling embeddings of those actions. For example, PINNERFORMER predicts all 3 actions after time step T: $A_{T+3}, A_{T+8}, A_{T+12}$ within a K-day window by randomly sampling 32 *(input action, target action)* pairs of embeddings using the user sequence $\{a_T, a_{T-1}, ..., a_{T-15}\}$ and the target sequence $\{a_{T+1}, a_{T+2}, ..., a_{T+K}\}$. It also demonstrates the effectiveness of mixed negative sampling [25, 26] on future recommendations. However, since the model randomly samples embeddings of the input and target actions in training, the predicted multi-action is not inherently ordered,

i.e., Bag of Actions output. Also, this method requires evaluations of performance and the model features (e.g., effects of window size, mixed negative sampling, time feature, etc.) using public datasets.

While PINNERFORMER predicts future actions without order using a private dataset, in this thesis, we introduce an efficient inference method for predicting a set of structured items for two particular training approaches of sequential recommendation models — all-item prediction models and dense-all-item predictions models, using public datasets. These models find a broad range of applications in, for example, information retrieval, movie recommendation, e-commerce, and social networking services [4, 3, 11, 20]. Furthermore, in the broader context of future actions recommendation workflow, these predictions can serve as baselines.

Using the conventional next-item inference methods for predicting future multi-item can be inaccurate for problems with more predicted items or require item order. By specializing in these particular training approaches of models, we leverage their inference to create a customized method for efficient and accurate future items inference.

The two sequential future recommendation models we consider are:

1. **All item prediction**: The all-item prediction model we define takes a sequence of user actions as input and predicts a set of n items based on a previous item. For model inference, we experiment with the next-item inference of SASRec [11] and the future items inference of our method. The model performance is largely dependent on the choice of these inference methods. The next-item inference model produces Bag of Actions output without item order, whereas the future items inference model produces structured output with item order.

2. **Dense all item prediction**: The dense-all-item prediction model differs slightly from the all-item prediction model. Instead of using a previous item, multiple previous items are used to predict a set of n items. We do not use the next-item inference for this model, as our focus is to demonstrate our method.

Since proven in the PINNERFORMER approach, we hypothesize that the mixed negative sampling and time feature would also improve performance with our inference method and examine its effect.

The models we discuss in this thesis are novel models of sequential recommendation and could appear when seeking to model an outcome as multiple consecutive items (or actions) in the correct order. The future items inference on the model enables outperforming the state-of-the-art next-item prediction model on sparse and dense public datasets on different evaluation metrics — confirms the future recommendation approach by [20] works better than the conventional next-action recommendation. We demonstrate these models on three comparisons.

1. **SASRec vs. next all-item.** Using SASRec as a baseline, we demonstrate the inefficiency of the next-item inference method on the all-item prediction model for future recommendation, which predicts a set of n items as Bag of Actions output.

2. **Next all-item vs. future all-item.** Compared to the next-item inference model, we constructed another all-item prediction model to demonstrate the efficiency of our future items inference method as well as mixed negative sampling [25] and window size effects on this model. In the case that requires analysis of the user's longer-term actions, the inferred items by the next-item inference and PINNERFORMER methods are not inherently ordered. Using the method of this thesis, we obtain more accurate results in the correct order.

3. **Future all-item vs. future dense-all-item.** Compared to the all-item prediction, we show performance improvement and temporal effects on the dense-all-item prediction. The model with dense data aims to predict the longer term of a set of 12 items, and output via all-item prediction can be impractically inaccurate. We use the dense-all-item prediction model with time feature via the future items inference method introduced in this thesis to outperform SASRec for this hardest challenge.

The structure of this thesis is as follows. In the following chapter, we provide background on the SASRec training and testing with metrics we use for the model inference and evaluation stages of the method. In Chapter 3, we summarize the method used in this thesis and the intuition behind the performance improvements. In Chapter 4, we evaluate the method on public datasets for discussion. Conclusions and generalizations of the method of this thesis are presented in Chapter 5.

Chapter 2

Background

The method of this thesis relies heavily on the next-item prediction model by SASRec. This section provides a brief overview of training and testing of SASRec, time feature encoding, and based metrics that will be used throughout the remainder of the paper. A more in-depth discussion of the contents of this section can be found in papers by [11, 15] and a book by [22].

2.1 Training and testing of SASRec

Learning to predict the next item starts by splitting a dataset. A sequence of item IDs as a user's behavior history is split using *Leave One Last* item split [18] (e.g., a user sequence *[1,2,3,4,5]* becomes *[1,2,3]* for training, *[4]* for validation, and *[5]* for testing).

To train the model, we first transform the training sequence $(S_1^u, S_2^u, ..., S_{|S^u|-1}^u)$ into a fixed-length sequence $s = (s_1, s_2, ..., s_n)$, where n is the maximum length that our model handles. If the sequence length is greater than n, we take the most recent n actions. If the sequence length is less than n, we repeatedly add zero as a 'padding' item to the right until the length is n. We then create an item embedding matrix $M \in \mathbb{R}^{|\mathcal{I}| \times d}$ where $\mathcal{I}$ is the item set and d is the latent dimensionality. The converted sequence s is used to retrieve the input embedding matrix $E \in \mathbb{R}^{n \times d}$ from M.

We next multiply this matrix E to the model's hidden dimension d using a learnable matrix $W \in \mathbb{R}^{d \times d}$, then add a larnable position embedding $P \in \mathbb{R}^{n \times d}$. This generates an input

$$V^{(0)} = EW + P \in \mathbb{R}^{n \times d} \tag{2.1}$$

for the transformer [24]. Following this, we apply a standard transformer model consists of alternating the point-wise two-layer feedforward network (FFN) blocks and self-attention (SA) blocks. In each SA block, we apply causal masking so a given output only attends to current or previous elements of the sequence. The model architecture is described as follows:

$$\begin{aligned} \mathrm{U}^{(l)} &= \mathrm{V}^{(l-1)} + \mathrm{Dropout}(\mathrm{SA}(\mathrm{LayerNorm}(\mathrm{V}^{(l-1)}))) \\ \mathrm{V}^{(l)} &= \mathrm{U}^{(l)} + \mathrm{Dropout}(\mathrm{FFN}(\mathrm{LayerNorm}(\mathrm{U}^{(l)}))), l = 1, \ldots, L \end{aligned} \tag{2.2}$$

That is, the layer normalization is applied on the input before feeding into the network to stabilize and accelerate training [1], apply dropout regularization on the network output to alleviate overfitting [23], and apply residual connections to propagate low-features to higher layers [7] by adding the input to the final output.

The final hidden state $V^{(L)} \in \mathbb{R}^{n \times d}$ represents the input features e used for supervised learning to predict a target o_i, a next item given the first t items. This target o_i is selected from the training sequence using the target pairing process described in Section 3.1 and retrieved from M, being a target embedding $O_i \in \mathbb{R}^{1 \times d}$. The input feature of item j, which is the previous item of the target item i, represents the last input embedding e_t at time step t:

$$V_{j,t}^{(L)} = e_t. \tag{2.3}$$

Using this last input embedding e_t and the target embedding O_i, we can compute a single relevance score of the target item i:

$$r_{i,t} = e_t \cdot O_i^{\top}, \tag{2.4}$$

where $r_{i,t}$ is the relevance of item i being the next item given the first t items (i.e., $s_1, \ldots, s_t$). This relevance score is then optimized by minimizing a binary cross entropy loss with Adam optimizer [13].

At test time, we use the trained model to output a list of ranked items and evaluate the rank of the test item. To obtain this list, we first randomly sample 100 items from an entire dataset and include the test item to create 101 target items, retrieved from the item embedding matrix M as target embeddings $O \in \mathbb{R}^{101 \times d}$. We then concatenate the training sequence and the validation item as a test input sequence, retrieve it from M as the input embedding matrix E, and pass it to the transformer to get a test input embedding e_t given the test input sequence with t items. Using Equation 2.4, the

obtained 101 relevance scores are sorted in descending order to see the ranking of the test item score. This ranking is evaluated using NDCG@K and HIT@K described in Section 2.3.

2.2 Time feature encoding

To capture more personalized user interest, we can feed a time feature embedding matrix T additionally into the transformer as another dimension of the input embedding matrix E. This matrix T can be created in several ways, for example, encoding the timestamp of each user action with Time2Vec [12]. Time2Vec encoding is known as periodicity representation, e.g., transforming a timestamp into minutes, days, or weeks to predict the next event time. PINNERFORMER uses this approach to represent K-day time window and to feed time features into their model [20].

If the periodicity is irrelevant to represent output or data split design, the time feature matrix T can be created using the encoding approach by TiSASRec [15]. Specifically, suppose that t is a fixed-length time sequence of user u, $t = (t_1, t_2, ..., t_n)$. Then, the TiSASRec encoding is used to obtain $T' \in \mathbb{N}^{n \times n}$ as a relation matrix that represents personalized time intervals:

$$T' = \begin{bmatrix} r^u_{11} & r^u_{12} & \dots & r^u_{1n} \\ r^u_{21} & r^u_{22} & \dots & r^u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r^u_{n1} & r^u_{n2} & \dots & r^u_{nn} \end{bmatrix}$$

For all time intervals on n items, we compute the scaled time intervals: $r^u_{ij} = [\frac{|t_i - t_j|}{r^u_{min}}]$, where $|ti - tj|$ is a time interval of two items i and j and r^u_{min} is the minimum time interval defined as $r^u_{min} = min(\boldsymbol{R}^u)$, where $\boldsymbol{R}^u$ is the time interval set (except for 0) of user u. This relation matrix T' is used to create the time feature embedding $T \in \mathbb{R}^{n \times n \times d}$ where d is the latent dimensionality.

In the method of this thesis, we use this TiSASRec approach due to its desirable representation property. In particular, this approach encodes the relative length of time intervals between every item in a user sequence and captures the user's interaction frequency, e,g., some users have more frequent interactions while others do not. As the output of our models is the item rankings, and we use item-split without temporal pe-

riodicity for our data split design, this encoding scheme possesses suitable representation.

Since the matrix size can be large, depending on the user sequence length, it can be computationally expensive, e.g., if the maximum sequence length n is set to 200, each user has a 200×200 relation matrix T' learned along with the input feature in the transformer. However, this explicit time representation can still improve performance with the shortened maximum sequence length [15].

2.3 Based metrics

In Section 3.3, we define new metrics to evaluate our models. In this section, we briefly review their underlying existing metrics, NDCG@K and Recall@K.

Recall. We start by defining Recall as an aggregation-based score that counts the fraction of times that the ground-truth future items are among the top K items. Recall@K for a set of users $\mathcal{U}$ is defined as:

$$\text{Recall@K} = \frac{1}{|\mathcal{U}|} \sum_{U \in \mathcal{U}} 1\{rank_U(i) \leq K\} \tag{2.5}$$

where $rank(i)$ is the ranking of item i produced by the model. In this case, the higher the Recall@K, the more relevant items are ranked within K. In the next-item prediction, where we only have one test item i for each user, this Recall@K is equivalent to Hit Rate (Hit@K) [11].

Normalized Discounted Cumulative Gain. We also observe a Cumulative gain (CG), a form of Top-N metric that can be used for measuring ranking performance in recommendation systems. As another aggregation-based score, CG measures how relevant predicted items are to user interest among the top K results [16]. For a user U, it counts follows:

$$\text{CG@K} = \sum_{i \in \{i | rank_U(i) \leq K\}} y_{U,i} \tag{2.6}$$

where the binary label $y_{U,i}$ is 1 if the number of relevant item i is among the top K results. In sequential recommendation models, the higher the CG more the relevant items are ranked high. The results should ideally be closer to the top of the ranked list.

Essentially, we care more about the difference between ranks 1 and 2 than those between 91 and 92. To penalize highly relevant items appearing lower in a ranking result list, we discount the low-rank CG@K as Discounted Cumulative Gain (DCG):

$$\text{DCG@K} = \sum_{i \in \{i | rank_U(i) \leq K\}} \frac{y_{U,i}}{\log_2(rank_U(i)+1)}. \tag{2.7}$$

Normalizing the DCG@K by the ideal discounted cumulative gain IDCG@K, we obtain the Normalized Discounted Cumulative Gain:

$$\text{NDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}}. \tag{2.8}$$

IDCG is the DCG that would have been achieved via an optimal item rank, which is obtained by sorting $y_{U,i}$ in decreasing order of relevance.

Chapter 3

Overview of method

3.1 Next item inference

SASRec [11] introduced a next item inference method for learning a target o_t as the expected output at time step t given a fixed length sequence of items $s = \{s_1, s_2, ..., s_n\}$:

$$o_t = \begin{cases} \texttt{<pad>} & \text{if } s_t \text{ is a padding item} \\ \mathrm{s}_{t+1} & 1 \leq t < n \\ \mathrm{S}^u_{|\mathcal{S}^u|} & \mathrm{t = n} \end{cases} \tag{3.1}$$

which changes items in s to the target items, resulting in the vector of targets o:

$$\begin{aligned} o &= \{o_1, o_2, ..., o_n\} \\ &= \{s_2, s_3, ..., \mathcal{S}^u_{|\mathcal{S}^u|}\} \end{aligned} \tag{3.2}$$

where s is converted from each user sequence $\mathcal{S}^u = (\mathcal{S}^u_1, \mathcal{S}^u_2, ..., \mathcal{S}^u_{|\mathcal{S}^u|-1})$ (excluding the last item) via truncation or padding items <pad> and n is the number of maximum sequence length. As described in Section 2.1, we retrieve this vector o from an item embedding matrix M as a target embedding matrix $O \in \mathbb{R}^{n \times d}$, and s as an input embedding matrix $E \in \mathbb{R}^{n \times d}$. This matrix E is used as input of the transformer to produce input embeddings e. The target embedding matrix O and the input embeddings e, which contain information of the different length sequences at each time step, can then be automatically paired via multiplication of e and O. Multiplying a target embedding O_t and the corresponding last input embedding e_t (Eq. 2.3), produces a relevance score for the next item (Eq. 2.4), optimized with binary cross entropy loss using one random negative sample.

Unfortunately, for the model we consider in this thesis, the approach of [11] cannot be directly applied — there is no change of items over s such that the target o_t is future multi-output. For the remainder of this section, we summarize the strategy used by the method of this thesis and provide intuition for the performance improvements obtained. The method described in the following section relies on a change of items that facilitates the automatic pairing of multiple target embeddings per input followed by optimization of the relevance score for each item position.

3.2 Future items inference

Training objectives. In order to summarize the method, we demonstrate its use in evaluation of the three training objectives. Suppose the time runs along the sequence from left to right, those objectives are defined below and depicted in Figure 3.1.

1. *Next item prediction:* Predict the next item I_{t+1} using the last input embedding e_t given the first t items $\{s_1, s_2, ..., s_t\}$.

2. *All item prediction:* Use the last input embedding e_t to predict the sets of 3, 5, and 12 items in the correct order, respectively, i.e., $\{I_{t+1}, ..., I_{t+3}\}$, $\{I_{t+1}, ..., I_{t+5}\}$, and $\{I_{t+1}, ..., I_{t+12}\}$ for a dense dataset. As for a sparse dataset, predict the sets of 2 and 3 items, respectively, due to its limited sequence length, i.e., $\{I_{t+1}, I_{t+2}\}$ and $\{I_{t+1}, I_{t+2}, I_{t+3}\}$. We exclude the user sequence from the dataset if it has no set of n items.

3. *Dense all item prediction:* Predict the structured items as same as All Item Predictin, but with a set of past input embeddings, e.g., the last input embedding e_t and the second last input embedding e_{t-1}.

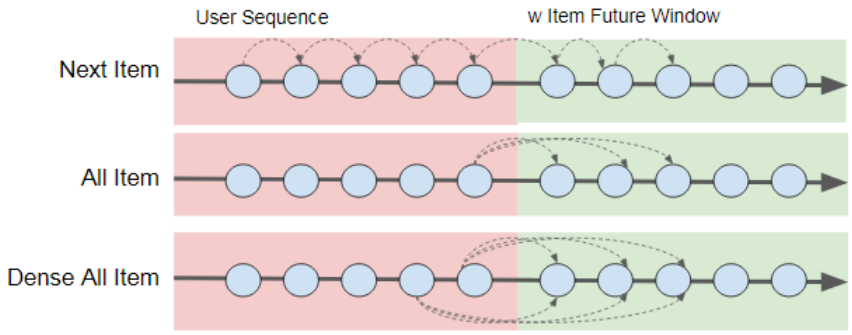

Figure 3.1: Training objectives for the number of predicted items of 3

Future items prediction using a next item predictor. It is known that due to the nature of simultaneous multi-feature training of the transformer [24], we can evaluate those objectives with the SASRec's next item predictor after the preconstruction of the dataset. For example, this can be done using the dataset design of PINNERFORMER [20]. Since SASRec splits data by item and the first two training objectives use the last input embedding e_t, one natural approach for evaluating the objectives is to use SASRec with *Leave N Out* split, extending the test window of *Leave One Out* split (see Section 2.1), that is, taking one item for validating [1], N items for testing, and the rest for training. Such a combination would result in an unstructured multi-output of Bag of Actions. This way, the training sequence length can change by the window size, i.e., the longer the window size, the more the padding items (Figure 3.2), which can affect performance. Alternatively, we could treat the window size as a maximum, as described in the following paragraph and depicted in Figure 3.4. In any case, for many advanced problems that require capturing the user's future actions, this unstructured output can result in erroneous analysis.

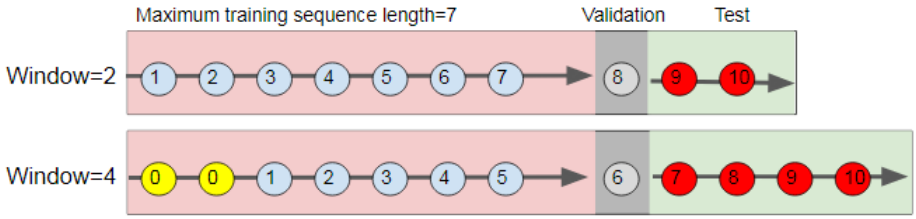

Figure 3.2: Data split design using *Leave N Out*. The longer window shortens training sequence, which is padded with 0.

Multiple target pairing. In order to improve the inadequate inference of this approach, we start with a change of this single target o_t to multiple targets $\boldsymbol{o}_t$:

$$\boldsymbol{o}_t = \{o_t, ..., o_{t+w}\} = \begin{cases} \texttt{<pad>} & \text{if } s_t \text{ is a padding item} \\ \mathrm{s}_{t+1}, & 1 \leq t < n \\ \mathrm{S}^u_{|\mathcal{S}^u|} & \mathrm{t = n} \end{cases} \tag{3.3}$$

where w is the window size.

The new target $\boldsymbol{o}_t$ is now a list of w targets. That is, the target can be expressed as a $n \times w$ matrix of multi-targets $o = \{\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_n\}$. Using this property, multiple

[1] Since taking N validation items makes the training sequence shorter and increases the number of padding items, affecting the performance difference in training and testing, we decided not to bother with the validation process and take one item for validation, leaving it as it was in the SASRec codebase.

target embeddings of $\boldsymbol{O}_t$ can be efficiently retrieved in the correct order given the window size of w. Now, the multiplication of this multiple target embeddings $\boldsymbol{O}_t$ and the corresponding last input embedding e_t produces multiple relevance scores for each item position (Figure 3.3).

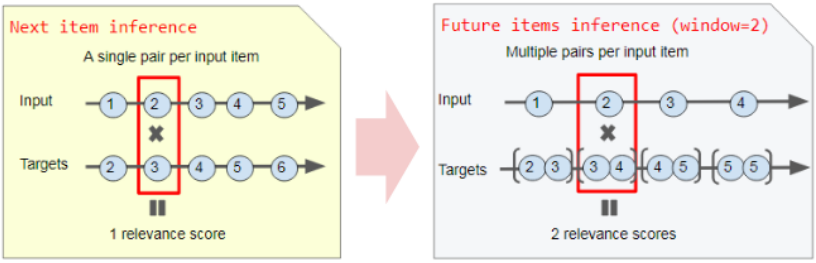

Figure 3.3: The *(input, target)* pair selection by the next item inference of SASRec (left) and by the future items inference of our method (right).

For this change, we partition the dataset using *Leave N Out* split *with a threshold* (Figure 3.4). A threshold is a boundary for taking the same number of test items regardless of the number of user actions and is determined by the maximum window size. This line fixes the training sequence length and allows us to compare performance fairly between our models. For example, if the number of items to predict is 2, and the max window size is 5, we take 2 test items from left to right, starting from the threshold, and can experiment with a window size of up to 5. This data split design also simplifies the pre-processing of the dataset. Now, instead of manually sampling the input embedding at a different time step and the corresponding target embedding post transformer and creating 32 *(input, target)* pairs per user, as done in [20], the multiple target embeddings per input embedding can be automatically paired.

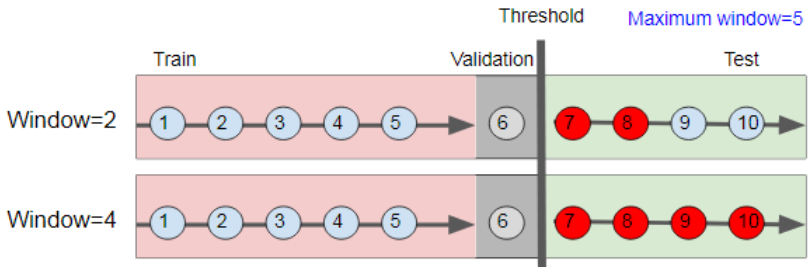

Figure 3.4: Data split design using *Leave N Out with a Threshold*, where maximum window size is 5. The training sequence length is fixed regardless of window size.

Time feature. We next perform a further change of the input for the transformer, $V^{(0)} = EW + P + T \in \mathbb{R}^{n \times n \times d}$ (Eq. 2.1), where T is an $n \times n \times d$ time feature embedding

matrix that represents the personalized time intervals (see Section 2.2 for details on T). This embedding T converts the input $V^{(0)}$ to a form that allows for effective learning of the inferred items for different window sizes of w.

Mixed negative sampling. Since in-batch negative samples and random ones can compensate for their sampling bias and variance each other, recommendation performance is known to improve by mixing those samples and increasing their number [25, 20]. We evaluate the impact of this sampling approach using our method. For each user, we randomly sample items from the whole item corpus as random negative samples and from a current training batch as in-batch negative samples. True items from the same user sequence $\mathcal{S}^u$ in different positions are not used as negative samples, e.g., s_2 cannot be used as a negative for s_1. Those samples are also retrieved from the item embedding matrix M as negative embeddings. Since in-batch negative samples are drawn from the distribution of items people interact with, they can demote popular items but support performance improvement by random samples [20].

Loss function. We now define a parameterized loss function for each *(input q, target p)* pair, where q and p are embeddings of an input item and a target item inferred by our method. The target embedding p is aware of its position and competes with the mixed negative embeddings m, which allows for better optimizing the multiple relevance scores for each future item position. Specifically, we define Sampled Softmax loss:

$$L(q, p) = -\log\left(\frac{e^{\langle q,p\rangle}}{e^{\langle q,p\rangle} + \sum_{j=1}^{N} e^{\langle q,m_j\rangle}}\right) \tag{3.4}$$

where $\langle q, p\rangle$ indicates the inner product of an input embedding and a target embedding, which produces logit as a relevance score.

In mixed negative sampling, the relevance score cannot be well estimated because a different rate of in-batch negative samples m produce the bias. We adjust for this sampling bias $Q(m)$ by correcting the estimates of the proportion of item that belong to in-batch or item corpus group. This sampling bias correction is known to improve performance by subtracting the term $\log(Q(m))$ from both target logits and negative logits [20] or from negative ones only [26, 25]. Since we do not sample the target items and also empirically confirmed the latter works best for our method, we subtract the term from negative logits only. In particular, we have the following formula for the

Sampled Softmax loss with a $\log(Q(m))$:

$$L(q,p) = -\log(\frac{e^{\langle q,p\rangle}}{e^{\langle q,p\rangle} + \sum_{j=1}^{N} e^{\langle q,m_j\rangle - \log(Q(m_j))}}) \tag{3.5}$$

where $Q(m)$ is a mixture of negative item frequency-based in-batch sampling and random sampling, characterized by a ratio between the batch size and item corpus size [25]. For a larger dataset, the approximation of $Q(m)$ is available using a count-min sketch [5]. Now, the relevance score can be better optimized with negative sampling bias correction but in increased operations.

All item prediction. Finally, we show an example learning process for a single input item in All Item Prediction using our method. That is, the model trains the last embedding of item 2 at that time step to learn targets of 3 and 4, and optimizes logits for each target using Sampled Softmax loss, where the window size is 2, and the number of mixed negative samples is 2 (Figure 3.5). We share the target and negative embeddings

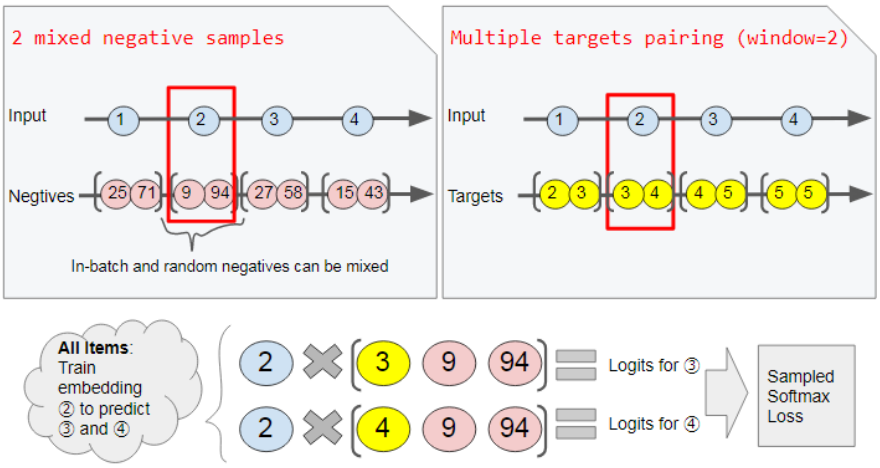

Figure 3.5: Shared embedding learning in All Item Prediction.

when computing the logit. We've improved the model output from Bag of Actions output without order to structured output with the correct order — the loss is optimized for a single target in next item inference, while multi-loss is optimized per target item position in future items inference.

Dense all item prediction. To further strengthen the learning signal of this All item Prediction approach, we also show an example learning process of Dense All Item Prediction, where the window size is 2, the input window size q is 2, and the number of mixed negative samples is 2 (Figure 3.6). Post transformer, we horizontally slide a

sequence of input features one to the right to select a previous input feature and perform the learning process of All Item Prediction q times. Although the input features at the end of the sequence overlap as much as we slide (Figure 3.6, the sequence of features in the right panel), this window sliding technique allows for efficiently computing the set of logits by running the transformer once only for each batch and reusing the learned input features.

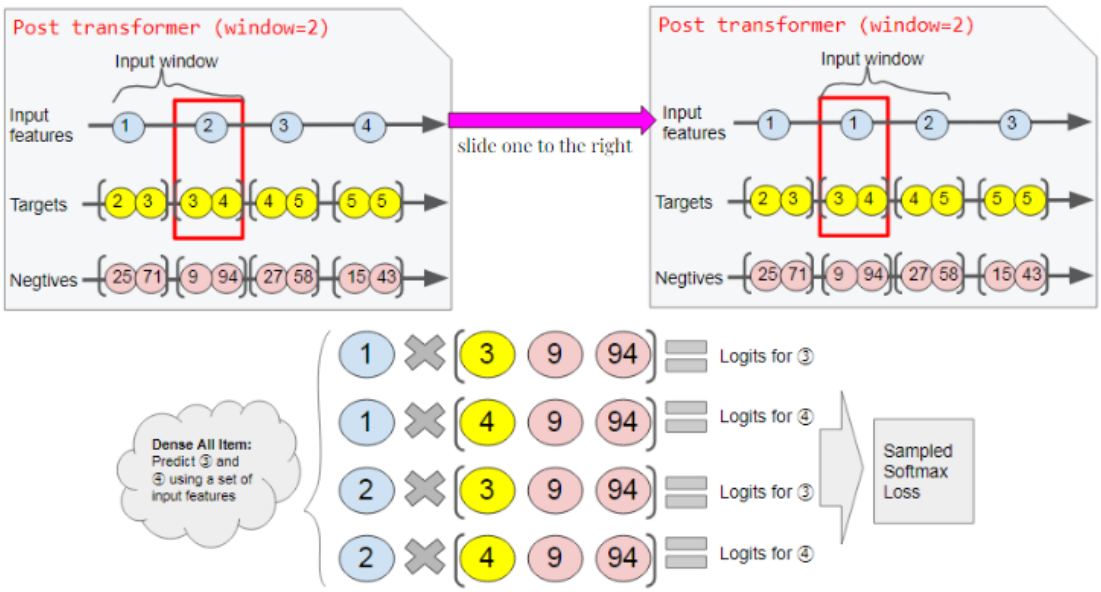

Figure 3.6: Shared embedding learning in Dense All Item Prediction.

The change of model components described in this section allows for efficient pairing of future target items per input item in addition to the structured output.

3.3 Metrics

We define two Top-N metrics to evaluate the performance of the method using the Normalized Discounted Cumulative Gain and the Recall as based metrics (see Section 2.3 for their derivations).

To measure the prediction accuracy of future items, we compute the average NDCG@K of a set of users $\mathcal{U}$ in a batch:

$$\text{NDCG}_s\text{@K} = \frac{1}{|\mathcal{U}|} \sum_{U \in \mathcal{U}} \text{NDCG@K(U)} \tag{3.6}$$

where s is the number of items to predict and does not depend on the user. If a user U does not have s items to predict, we exclude the user from the dataset in preprocessing.

NDCG@K(U) is defined by the average accuracy of predicted items after time step t for each user U:

$$\text{NDCG@K(U)} = \frac{1}{s} \sum_{i=t+1}^{s} \text{NDCG@K}_i, \tag{3.7}$$

where we plot NDCG@K_i later in experiments.

To measure the hit rate of future items that the ground-truth future items are among the top K items, we compute the average RECALL@K of a set of users $\mathcal{U}$ in a batch:

$$\text{RECALL}_s\text{@K} = \frac{1}{|\mathcal{U}|} \sum_{U \in \mathcal{U}} \text{RECALL@K(U)} \tag{3.8}$$

where RECALL@K(U) is defined by the average recall of predicted items after time step t for each user U:

$$\text{RECALL@K(U)} = \frac{1}{s} \sum_{i=t+1}^{s} 1\{rank_U(I_i) \leq K\} \tag{3.9}$$

where I is the item.

Chapter 4

Experiments with discussions

In this section, we present exploratory models used to analyze two inference methods and model components. Fitting public data from MovieLens-1m and Amazon Beauty datasets via next-item inference and our future items inference, we evaluate the effectiveness of the future actions prediction approach by PINNERFORMER [20].

4.1 Application: Movie/product review

Having accurate predictions of recommendation items across scientific and industrial applications can be extremely helpful. For example, reliable recommendations allow users to locate items of interest efficiently, they can retrieve information of the top priority in search engines, and they can facilitate better marketing tools for online businesses. To improve predictions of recommendation items in the absence of real-time data, several institutions created datasets that allowed us to simulate real-world data via publicly available reviews [6, 8].

Datasets. One dataset where the movie reviews provide dense data is MovieLens-1m, where historically watched movie data is provided by a million user ratings. Another dataset where the product reviews provide sparse and variable data is Amazon Beauty, where historically purchased item data is provided by 400 thousand user reviews. Figure 4.1 indicates that the interaction sequence has a different number of actions per user where movie reviews have dense interactions as opposed to product reviews. The large amount of data collected from review respondents provided the applied scientists and service providers with a great resource. However, at the same time, an emerging recommendation objective turns the methodological aspect of sequential modeling into

a substantial challenge of longer and structured action prediction. We use these various densified datasets to tackle this challenge.

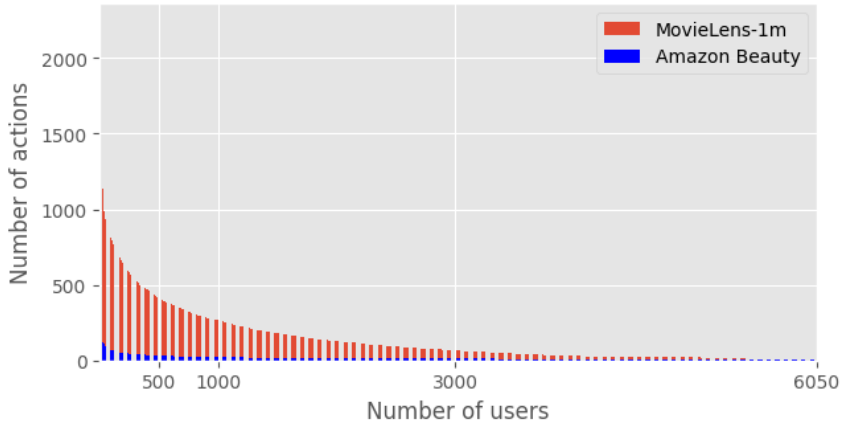

Figure 4.1: Number of actions distribution of two datasets.

Data preprocessing. We summarize the data preprocessing and the data split design used for this objective to model data from the reviews collected in MovieLens-1m and Amazon Beauty. We follow the procedure from [21, 11] and preprocess data. That is, IDing reviewed items per user to create a set of user sequences and using timestamps to order these sequences while discarding ones with less than five items. For data split design, we use *Leave N Out* (see Figure 3.2 in Section 3.2) for the next-item inference models. With this split design, the average number of actions per user varies by window size between 3 and 28, and is the number in the sequence used for prediction (i.e., does not include events used for evaluation). For the future items inference models, we use *Leave N Out With A Threshold* (see Figure 3.4 in Section 3.2) with a maximum window size of 28, fixing the average number of actions. After preprocessing the data for both datasets, data statistics are summarized in Table 4.1.

			Avg. actions per user		
Dataset	#Users	#Items	Leave N Out	Leave N Out With A Threshold	#Actions
MovieLens-1m	6,040	3,416	139.77 - 161.5	139.77	1.0M
Amazon Beauty	52,024	57,289	3.71 - 4.66	3.71	0.4M

Table 4.1: Dataset statistics after preprocessing.

4.2 Implementation details

We implement our models with PyTorch. Since we extend SASRec [11] and use it as a baseline, primary settings remain unchanged. That is, the maximum sequence length is set to enough cover the average number of actions per user for each dataset, the optimizer is the Adam optimizer [13] with a learning rate of 0.001, the batch size is 128, and the dropout rate varies due to data sparsity or density by the dataset. Varied hyperparameters by dataset are shown in Table 4.2. Since the maximum relative time interval between any two items can be large, we clip it to 256, the value considered useful, according to the TiSASRec study [15].

Dataset	Max sequence length	Max time interval	Dropout	Regularization (time feature)	
MovieLens-1m	200	256	0.2	0 (N)	0.00005 (Y)
Amazon Beauty	50	256	0.5	0 (N)	0.00005 (Y)

Table 4.2: Hyperparameter settings.

4.3 Predict future items with next item inference

We first demonstrate future item predictions using the existing next-item inference method, describing the Bag of Actions output over varied window sizes; this is a small example that shows how well the next-item predictor can predict future items without our method. Since the loss is not optimized for each item position, the output multi-item is not inherently ordered.

4.3.1 Next-item and all-item predictions procedure

4.3.1.1 With a dense dataset

We try predicting future 3, 5, and 12 items, respectively, for each model with a different window size, splitting MovieLens-1m data with *Leave N Out*.

We demonstrate the efficiency of the next-item inference method in evaluating predicted accuracies and hit rates of the all-item prediction model compared to SASRec.

SASRec performs prediction of future n items by predicting the next-item n times given the previously-predicted item. As a result, output items are structured, but a miss-prediction can quickly lead to performance degradation, as it is fed as input to the model. The summary of the models is as Table 4.3.

Model	Inference	Training objective	Output
NextItem (SASRec)	Next item	Next Item Prediction	Structured
NextAllItem	Next item	All Item Prediction	Bag of Actions

Table 4.3: Summary of the models using next item inference.

Because the long window shortens the training sequence length and increases the sparsity of training data by more padding items, this affects starting performance. Figure 4.2 (left) plots the effect of varying window size on NextAllItem models. We observe that these all-item prediction models with longer windows perform better in predicting the first (next) item. This behavior indicates that we are changing the prediction task by predicting with the longer windows and this is an easier task for some reason and that the model start with the higher starting performance. As evidence, we don't see this varied starting performance in Figure 4.2 (right) when fixing the training sequence lengths by splitting data with *Leave N Out with a Threshold* with a maximum window size of 28. Also, Figure 4.2 (left) shows the model with a longer window has a more gentle performance drop than the one with a shorter window, which would result in a higher NDCG mean. This characteristic lines with the PINNERFORMER results [20] and is reflected to NextItem performance in predicting future 28 items in Figure 4.2 (left). As a benchmark, we use this NextItem model as a baseline to outperform.

Based on this observation, we compare NextItem to NextAllItem predictions evaluated on *Leave 28 Out* for MovieLense-1M, as the window sizes are consistent. Figure 4.3 shows the accuracy of those all-item predictions via next-item inference as a function of item position. Because the loss of NextAllItem is not optimized for the items after the first item, the difference in performance robusticity between two models becomes apparent as the number of predicted items increases. Table 4.4 summarizes the NDCG mean and Recall mean of those next-item inference models. Less accurate predictions were obtained via the next-item inference method for all-item prediction with more predicted items.

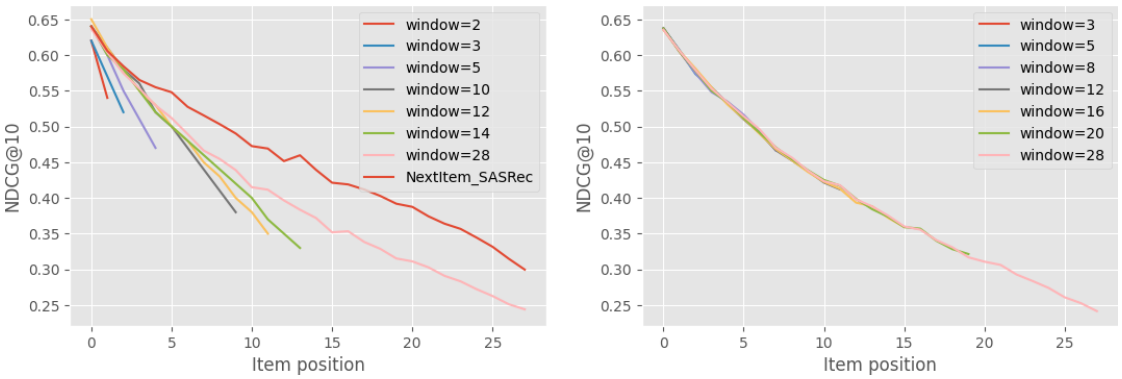

Figure 4.2: Effect of varying window size on NextAllItem with *Leave N Out* data split (left) and *Leave N Out with a Threshold* (right) on MovieLense-1m. Different lengths of the training sequence affects starting performance (left), while fixing it has no effect (right).

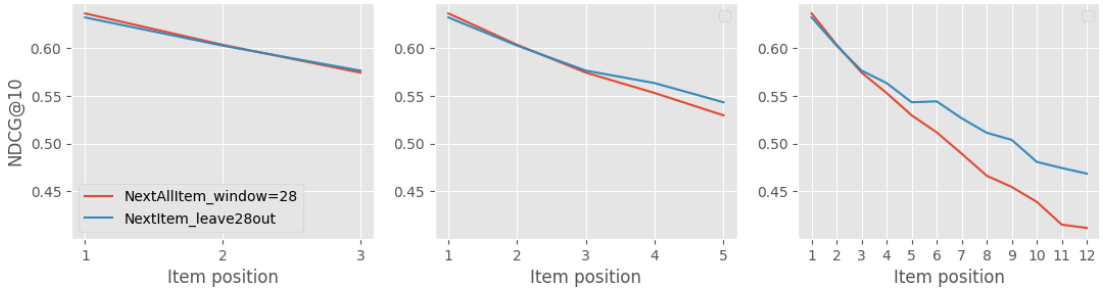

Figure 4.3: Accuracy of predicting future 3, 5, and 12 items via next-item inference as a function of item position. The performance gap between the two models enlarges as the number of predicted items increases.

Model_window	$NDCG_3$	$NDCG_5$	$NDCG_{12}$	$Recall_3$	$Recall_5$	$Recall_{12}$	Ave seq len
NextItem	**0.6067**	**0.5885**	**0.5356**	0.8447	0.8263	**0.7742**	139.77
NextAllItem_28	0.6048	0.5794	0.5070	**0.8502**	**0.8303**	0.7712	139.77

Table 4.4: Performance of the next-item inference models evaluated on *Leave 28 Out*. NextItem (SASRec) performs better than NextAllItem for MovieLense-1m.

4.3.1.2 With a sparse dataset

We also demonstrate the efficiency of the next-item inference method in evaluating NDCGs and Recalls of the all-item prediction model on the Amazon Beauty dataset. Similarly to the dense data models, the model with a longer window has higher accuracies in every item (Figure 4.4). We challenge NextItem with 4 predicted items as a baseline.

NextItem vs. NextAllItem appear to be dataset dependent that the all-item prediction works better than the next-item prediction.

Table 4.5 shows that NextAllItem with a window size of 4 achieves higher NDCG means than NextItem predictions evaluated on *Leave 4 Out*. Recall that the training data sparsity by a long window had a higher first-item prediction in Figure 4.2. Because the average number of actions per user is small and the data sequences have a sparse structure, NextAllItem might take advantage of it. For problems with more predicted items, the data sparsity would make the prediction difficult. Figure 4.5 shows the NDCG@10 of future item predictions via NextAllItem as a function of item position and the NDCG@10 achieved by NextItem. We obtained relatively accurate predictions for additional predicted items via next-item inference.

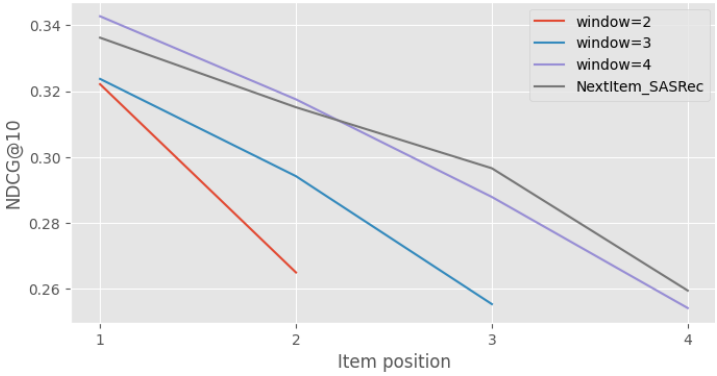

Figure 4.4: Effect of varying window size on NextAllItem with *Leave N Out* data split on Amazon Beauty. Same as Figure 4.2 (left), the longer windows make the prediction task easier.

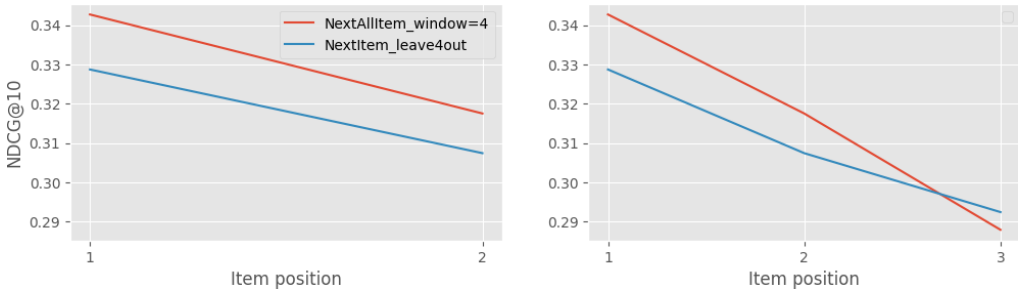

Figure 4.5: Accuracy of all-item prediction via next-item inference as a function of item position with Amazon Beauty. NextAllItem outperforms NextItem evaluated on *Leave 4 Out*.

Model_window	$NDCG_2$	$NDCG_3$	$Recall_2$	$Recall_3$	Ave seq len
NextItem	0.3256	0.3159	**0.4883**	**0.4756**	3.71
NextAllItem_4	**0.3301**	**0.3160**	0.4882	0.4746	3.71

Table 4.5: Performance of the next-item inference method on NextAllItem models with Amazon Beauty. Better accuracies were obtained via NextAllItem.

4.4 Predict future items with future items inference

In this section, we present the future items inference models used to model data from the user actions collected in MovieLens-1m [6] and Amazon Beauty [8]. Predicting a set of items with the next-item inference does not give better robust accuracy. Using the method of this thesis, we can recommend future items with higher accuracy in the correct item order.

4.4.1 FutureAllItem and mixed negative sampling procedure

The user actions record several features, including the item name and timestamp where each user has engaged. We can use the data to identify future items in order of purchase, where the relevance scores seem high.

A first prediction model with our method takes a sequence of item IDs s per user as input and uses the last input item embedding e_t at time step t to predict all subsequent n items in the correct order. This model splits data with *Leave N Out with a Threshold* with a maximum window size of 28 for MovieLense-1m and 4 for Amazon Beauty, and the training sequence length has been fixed to evaluate our method without window size benefit. In addition, we use a minimum window size w required to predict the number of items n (i.e.,$w = n$). This setting corresponds to an all-item prediction model with the future items inference, FutureAllItem. In mixed negative experiments where the relevance score cannot be well estimated due to a low rate of in-batch negative sample (the rate of in-batch items may be low because if we sample the same items more frequently, it can demote popular items quickly), we can rely on the rest of the model, an all-item prediction model based on item density.

Only a fraction of the in-batch items contribute to the negative samples, which raises

issues about biases. Let the proportion of items sampled from the batch or item corpus represent item density. It's notably a concern because different item groups densify differently: not only do their difficulties of learning items vary, but their degree to the demotion of popular items also changes. In the all-item prediction and mixed negative sampling model, FutureAllItemMixLogQ, we adjust for these biases by using estimates of the proportion of items that belong to the in-batch or item corpus group. For this model, this proportion Q is estimated using the entire training data. Modifying Eq. 2.4, this leads to a corrected estimate for the relevance score of a negative item i decreasing relevance scores for popular items:

$$\begin{aligned} r_{i,t} &= e_t \cdot m_i^\top - \log(Q) \\ &= e_t \cdot m_i^\top - \log(\frac{c_i}{|\mathcal{I}|}), \end{aligned} \tag{4.1}$$

where e_t is the input embedding at time step t, m is the negative embedding, c_i is the frequency count of negative item i, which is from a batch or item corpus, and $\mathcal{I}$ is the item set in the training data.

Using this negative relevance score, we compute the bias-corrected sampled softmax loss for the *(input, target)* pair, per the loss function (Eq. 3.5). The summary of models is as Table 4.6.

Model	Inference	Training objective	Output
FutureAllItem	Future items	All Item Prediction	Structured
FutureAllItemMixLogQ			

Table 4.6: Summary of the all-item prediction models with future items inference.

4.4.2 Comparison of our method to NextAllItem

4.4.2.1 With a dense dataset

We predict the future 3, 5, and 12 items using our future items inference method fitting on MovieLense-1m. We compare the model with NextAllItem, the all-item prediction model using the next-item inference method by SASRec, which we fitted in Section 4.3.

Our proposed method allows FutureAllItem to produce the higher NDCG@10 and Recall@10 for future 3 and 5 items, improving the output to a structured one.

Next, building on FutureAllItem, we fit the model using mixed negative sampling with LogQ bias correction. After tuning the best mix ratio (see Section 4.4.4 for detail on the mix ratio) for each number of predicted items, we train the model for 200 epochs and compute the average performance. Table 4.7 shows the average NDCG@10 and Recall@10 of models for all predicted items computed by each method. For each item position, we computed the NDCG@10, and Figure 4.6 plots the difference between the model predictions as a function of item position. While NextAllItem takes a longer window to get relatively accurate Bag of Actions results for just 3 item predictions, FutureAllItem with our method achieves structured results with less performance drop for all items with a minimum window. Also, it outperforms the baseline, NextItem, for 3 and 5 predicted items. Recall that NextAllItem does not inherently order the output items — their mean scores in Table 4.7 are mainly contributed by its higher NDCG@10 of first-item prediction.

FutureAllItemMixLogQ uses the best mix ratio of 1:6 for 3 predicted items, 2:8 for 5 items, and 1:8 for 12 items, and provids a substantial accuracy improvement over FutureAllItem. None of Recall@10 for the predicted items scored best on the FutureAllItemMixLgQ. This degradation is because the mixed in-batch negatives with corrected sampling bias can demote popular items, resulting in high prediction accuracy on hits but low hit rates across prediction items.

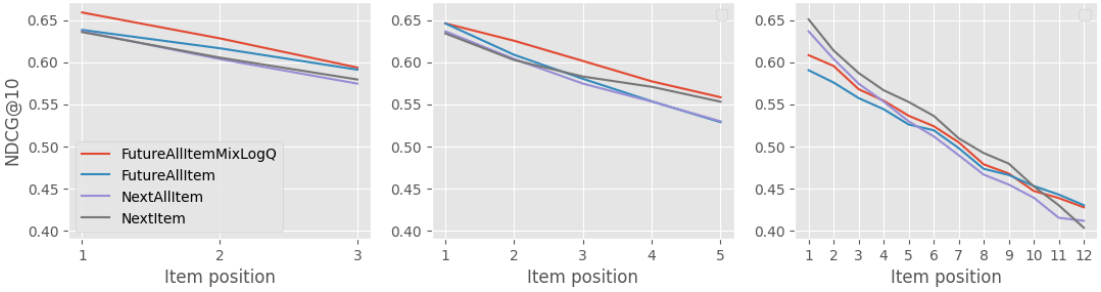

Figure 4.6: Accuracy of all-item predictions via future items inference as a function of item position using MovieLense-1m. FutureAllItem has less performance drop than NextAllItem.

Model	$NDCG_3$	$NDCG_5$	$NDCG_{12}$	$Recall_3$	$Recall_5$	$Recall_{12}$
NextItem (SASRec)	0.6067	0.5885	**0.5356**	0.8447	0.8263	0.7742
NextAllItem	0.6048	0.5794	0.5070	0.8502	0.8303	0.7712
FutureAllItem	0.6151	0.5906	0.5062	**0.8620**	**0.8436**	**0.7826**
FutureAllItemMixLogQ	**0.6268**	**0.6015**	0.5124	0.8548	0.8369	0.7725

Table 4.7: Improved performance for the future items inference models on MovieLense-1m compared to the next-item inference ones.

4.4.2.2 With a sparse dataset

We evaluate the models using our method on Amazon Beauty dataset following the same procedure as in Section 4.4.2.1.

Our proposed method substantially outperforms the next-item inference models for all item positions and takes a minimum window to run. Figure 4.7 plots the difference between our method and the next-item inference as a function of item position. While on sparse data, there is a limit to maximum window size and performance gains for the next-item inference model, as it needs to benefit from window size, our method achieves significantly better results without this benefit.

We also fit the mixed negative model using this sparse dataset tuning the mix ratio with LogQ correction. The best mix ratio for FutureAllItemMixLogQ is 0:8 without LogQ correction for all predicted items. In other words, in the sparse data model, mixing in-batch negative samples, increasing them, and (or) correcting the sampling bias degrade performance. See Section 4.4.4 for detail on this discussion. Table 4.8 shows the NDCG mean and Recall mean for all predicted items, computed by each method, where we present the FutureAllItemMixLogQ result using the mix ratio of 1:6 with LogQ. We obtained results contrary to our assumption via mixed negative models using a sparse dataset containing a few actions per user.

4.4.3 Limitations of the model and our inference method

We believe the presented model improves the future recommendations simulated on the review data because (i) it uses future items inference to capture longer-term user actions better, and (ii) it corrects item order using a multi-loss optimization per item position. A more accurate capturing of future actions would use sequential output

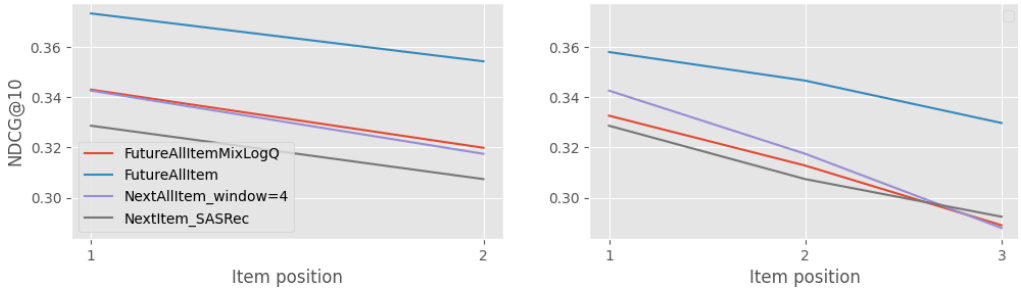

Figure 4.7: Accuracy of all-item prediction via future items inference as a function of item position with Amazon Beauty. FutureAllItem provides substantial improvement over the next-item inference models.

Model	$NDCG_2$	$NDCG_3$	$Recall_2$	$Recall_3$
NextItem (SASRec)	0.3256	0.3159	0.4883	0.4756
NextAllItem	0.3301	0.3160	0.4882	0.4746
FutureAllItem	**0.3639**	**0.3448**	**0.5259**	**0.5015**
FutureAllItemMixLogQ	0.3314	0.3114	0.4696	0.4499

Table 4.8: Improved performance for the future items inference models on Amazon Beauty compared to the next-item inference ones. The simple model (FutureAllItem) works better than the computationally complex model (FutureAllItemMixLogQ).

rather than multi-output. We can generate such a sequence using scheduled sampling [2] of the last input embedding e_t. Extending the next-item inference scheme into a sampling scheme to generate such sequences is a direction we hope actively pursue.

This consideration suggested that it might be beneficial from a modeling standpoint to build a more sophisticated model, which is outside of the scope of the method of this thesis. Nevertheless, the model considered here is a significant step in better model development.

4.4.4 Effect of mixed negative sampling

We experiment with the set of mix ratios and the LogQ sampling bias correction to see the effect of the mixed negative sampling approach. We sample items from a current training batch as in-batch negative samples and from the item corpus as random ones. Multiple mixed negative items are sampled per input item, varying the mix ratio.

We first fit FutureAllItem for predicting future 5 items using the MovieLens-1m dataset, exploiting denser data sequences for a standard comparison. We train 6 models for 200 epochs, using different ratios and LogQ sampling correction, and take the best results throughout the epochs. For each model, we computed the NDCG means. Mixing in-batch negative samples simply without LogQ improves $NDCG_5@10$ for dense data. FutureAllItem using 2 in-batch negative samples and 8 random ones with LogQ performs best.

We next fit the model for predicting future 2 items using the Amazon Beauty dataset, exploiting the sparsity of the sequential data for a challenging task. We used the same procedure as dense data and computed $NDCG_2@10$. FutureAllItem using the mix ratio of 0:8 without in-batch negative samples nor LogQ performs best. We also performed the same procedure for the sparse MovieLens-1m, in which the average number of actions per user is reduced from 163.5 to 4.0, and confirmed the similar result.

Table 4.9 shows the NDCG means for all combinations of mix ratio and LogQ, computed using each dataset. As hypothesized that mixed negative sampling would also improve future item predictions with our method, the model with dense data takes more mixed samples to get more accurate results, and the LogQ bias correction has a significant impact. Contradicting this assumption, the model with sparse data degrades with mixing and increasing negative samples, and LogQ was counterproductive.

Item demotion by sparse in-batch negatives. We analyze the results of in-batch negative sampling by illustrating the situation in the mix ratio of 2:8. We sample 2 in-batch items per input item for each user for one epoch. For example, if a user sequence has 5 items (actions), 10 negative items are sampled for the user. Figure 4.8 depicts the histogram of in-batch item frequencies for MovieLens-1m and Amazon Beauty datasets. The number of unique in-batch items was 642 for Amazon Beauty while 2348 for MovieLens-1m. A decrease in $NDCG_2@10$ with increasing in-batch sample indicates that the negative samples frequently taken from fewer in-batch items can significantly demote popular items, making the problem harder.

Model sensitivity in sparse data. We also analyze the results in the LogQ correction and the number of negative samples using the average attention weights from

the SASRec study [11]. LogQ correction did not work even on the random negative sampling in the model with sparse data. Recall that SASRec has intensified attention weights on more recent items for Amazon Beauty and diversified ones on various items for MovieLens-1m [11]. A decrease in $NDCG_2@10$ with LogQ correction or more negative samples implies that the model with sparse data might be sensitive to negative logits, involving attention weights — post transformer, the input embedding e_t reflects these weights, which means that the multiplication of e_t and more negative embeddings or a small logit subtraction can amplify the negative logit (Eq. 4.1) on recent items and overestimate the negative relevance score. With a similar logic, the model with dense data diversifies negative logits to various items and can accept more negative samples.

Mix Ratio		MovieLense-1m (139.77)	Amazon Beauty (3.71)	Sparse MovieLense-1m (4.00)
Batch:Random	LogQ	$NDCG_5@10$	$NDCG_2@10$	$NDCG_2@10$
1 : 6	Y	0.5912	0.3314	0.3936
0 : 8	N	0.5906	**0.3639**	**0.3959**
0 : 8	Y	0.5907	0.3462	0.3879
1 : 8	N	0.5910	0.3426	0.3941
2 : 8	N	0.5952	0.3250	0.3872
2 : 8	Y	**0.6015**	0.3176	0.3919

Table 4.9: Effect of varying mix ratio and LogQ bias correction on FutureAllItemMix. 5 items are predicted with MovieLense-1m and 2 for Amazon Beauty and Sparse MovieLens-1m. The average number of actions per user is in parentheses.

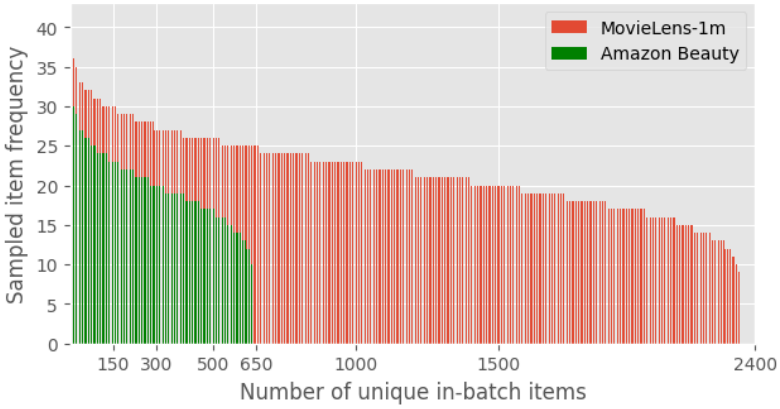

Figure 4.8: Frequencies of in-batch samples for MovieLens-1m and Amazon Beauty.

4.4.5 Effect of window size

We use MovieLense-1m to examine the effect of increasing window size and the number of predicted items on the all-item prediction model by two inference methods. The training sequence is fixed with *Leave 28 Out With A Threshold* so we can experiment with varied window sizes. Figure 4.9 plots the performance impact on the future items inference model as a function of item position.

Recall Figure 4.2 (right) showed that the next-item inference model had no performance difference by window size, and its prediction curves stayed the same for any number of predicted items. In contrast, Figure 4.9 shows that FutureAllItem degrades the performance as the number of predicted items increases, and its prediction curves diverge with an enlarged performance gap. It also indicates that window size and the number of predicted items must match for FutureAllItem to perform best, as it optimizes multiple losses for the corresponding item position.

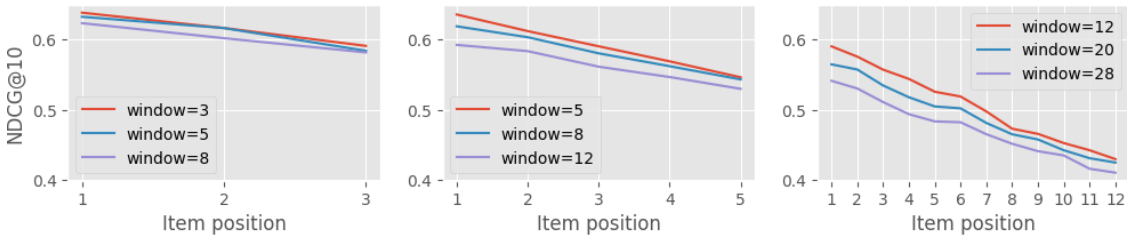

Figure 4.9: Effect of increasing window size on FutureAllItem via future items inference. Models with matching window size and number of predicted items perform best.

4.4.6 FutureDenseAll and time feature incorporation procedure

We finally apply our method to a dense-all-item prediction to strengthen the learning signal by all-item prediction to see if it improves performance. Strengthening this signal requires learning from multiple input embeddings to densely train the last input embedding e_t. Furthermore, we incorporate a time feature that can model temporal user interactions, such as purchase frequency, and capture them using personalized time intervals. As in Section 4.4.2, we use our method to infer future n items and the mixed negative sampling.

We analyze temporal data from MovieLense-1m using the time interval distribution in the study of TiSASRec [15]. Users express the degree of interest in movies via

different time intervals. For example, most users take about 25 seconds to rate the next movie, while some take > 100 seconds. These pre-computed time intervals help to personalize the input feature for the transformer. Each user sequence is given a relation matrix, T, indicating the relative time intervals between any two items.

We tune the best mix ratio for the model and predict sets of 3, 5, and 12 items using MovieLense-1m and sets of 2 and 3 items using Amazon Beauty, and compare the model with FutureAllItem, which we fitted in Section 4.4.2. We use the input window size q of 3 to select multiple input items for *(input, target)* pairing. Using the features of these input items as additional input embeddings, denoted as e_t, e_{t-1}, and e_{t-2}, we compute the relevance scores of future target items. We also incorporate the time feature into the all-item model in addition to this dense-all-item model. The summary of the models is as Table 4.10.

Model	Inference	Training objective	Output
FutureAllItemTime	Future items	All Item Prediction	Structured
FutureDenseAllMix	Future items	Dense All Item Prediction	Structured
FutureDenseAllMixTime			

Table 4.10: Summary of the dense-all-item and time feature models.

The tricky part to estimate here is the sampling bias, LogQ, to which we correct for negative relevance scores. Because the model includes additional input embeddings, it performs bias correction toward optimizing the relevance scores based on the set of previous input embeddings, not the last input embedding e_t. Since the model is designed to train the last input embedding, this diversified bias correction can result in erroneous optimization, which we also confirmed empirically. Thus, we do not correct the negative sampling bias for this densely trained model.

4.4.7 Comparison of densely trained models to FutureAllItem

4.4.7.1 With a dense dataset

As we analyzed in Section 4.4.4, FutureDenseAllMix allows more mixed negative samples and uses the best mix ratio of 6:13 for all predicted items. Table 4.11 summarizes the performance of our method on these densely trained models using MovieLens-1m. We compute the NDCG@10 of model predictions for each item position and plot in

Figure 4.10.

Table 4.11 shows that the dense-all-item predictions improve the all-item predictions. Observing that FutureAllItemTime prediction for a set of 12 items shows competitive performance to NextItem and FutureDenseAllMixTime, we may not need the dense training to outperform the baseline. In Figure 4.10, the performance gap between FutureDenseAllMixTime and FutureDenseAllMix in predicting a set of 12 items indicates the efficiency of the time feature in longer-term predictions. As the best model, Table 4.11 shows that the FutureDenseAllMixTime outperforms NextItem for all the predicted items.

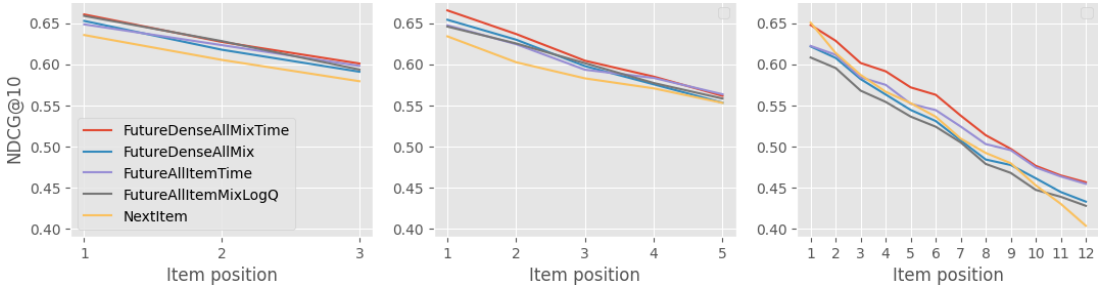

Figure 4.10: Accuracy of dense-all-item predictions via future items inference as a function of item position using MovieLens-1m. FutureDenseAllMixTime fully outperforms NextItem for 3, 5, and 12 predicted items.

Model	$NDCG_3$	$NDCG_5$	$NDCG_{12}$	$Recall_3$	$Recall_5$	$Recall_{12}$
NextItem (SASRec)	0.6067	0.5885	0.5356	0.8447	0.8263	0.7742
FutureAllItem	0.6151	0.5906	0.5062	**0.8620**	0.8436	0.7826
FutureAllItemTime	0.6232	0.6023	0.5335	0.8612	**0.8461**	0.7964
FutureAllItemMixLogQ	0.6268	0.6015	0.5124	0.8548	0.8369	0.7725
FutureDenseAllMix	0.6226	0.6020	0.5237	0.8543	0.8444	0.7909
FutureDenseAllMixTime	**0.6295**	**0.6106**	**0.5457**	0.8560	0.8418	**0.7979**

Table 4.11: Improved performance for the dense-all-item predictions with MovieLens-1m compared to the all-item ones. Time feature further helps increasing accuracy.

4.4.7.2 With a sparse dataset

We also evaluate these densely trained models and time feature models using our method on the Amazon Beauty dataset. Since we found in Section 4.4.4 that the mixed negative sampling and LogQ correction do not work for sparse data, we use 8 random negative samples per input item without LogQ. We also found that using more than 8 random negative samples degraded performance. The input window size of q is set to 2, i.e., use the last input embedding e_t and the second last one e_{t-1} for training.

Table 4.12 summarizes the NDCG mean and Recall mean of the predictions for future 2 and 3 items, and NDCGs for each item position are plotted in Figure 4.11. Figure 4.11 shows that the accuracy for every item increases with dense-all-item prediction while decreasing with incorporating the time feature. We also see this degrading tendency in FutureAllItemTime, the all-item prediction with the time feature. The results obtained so far show that the model with sparse data does not prefer any additional feature. The consistent best Recall means for FutureDenseAll in Table 4.12, and also for FutureAllItem in Table 4.8, indicate the mixed negatives or LogQ correction likely influences Recall. For further insight, we may examine how hyperparameters behave, e.g., maximum time intervals, learning rates, etc.

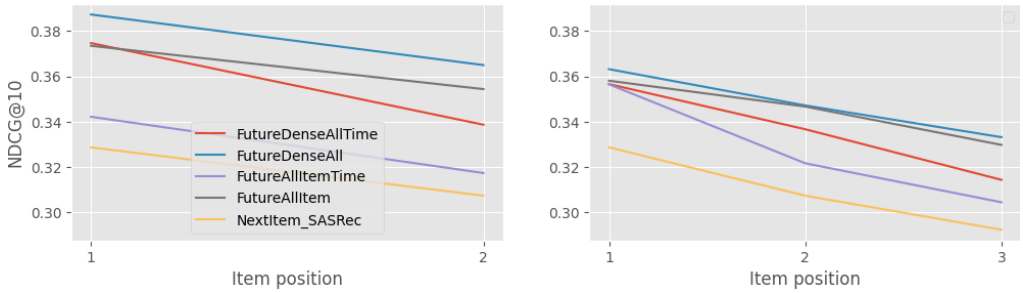

Figure 4.11: Accuracy of dense-all-item predictions via future items inference as a function of item position with Amazon Beauty. FutureDenseAll improves FutureAllItem.

4.4.8 Limitation of the time feature model

The present model certainly has a limitation. One could consider time intervals as targets. This idea seems sensible since, for instance, target items paired with time intervals between the target item and the previous one are likely ordered differently than ones without them. The inference method presented in this thesis can handle time intervals as additional target embeddings when computing logits. This learning

Model	$NDCG_2$	$NDCG_3$	$Recall_2$	$Recall_3$
NextItem (SASRec)	0.3256	0.3159	0.4883	0.4756
FutureAllItem	0.3639	0.3448	0.5259	0.5015
FutureAllItemTime	0.3297	0.3275	0.4900	0.4866
FutureDenseAll	**0.3761**	**0.3478**	**0.5359**	**0.5035**
FutureDenseAllTime	0.3567	0.3358	0.5170	0.4989

Table 4.12: Improved performance for the dense-all-item predictions with Amazon Beauty compared to the all-item ones. Time feature is counterproductive.

approach would involve experimenting with a novel loss function, which could lead to a timing-aware model that accurately predicts when and what to recommend in the future. Even then, our future items inference method would be efficient to implement this approach, for example, by inferring multiple target items along with time intervals.

Chapter 5

Conclusions

In this thesis, we describe an efficient inference method for predicting the order of a set of items that a user will purchase based on the user's purchase history. The obtained results using public datasets confirm the future recommendation approach recently introduced is effective.

Our method allows for recommending the future consecutive multi-item in a particular order. We used movie and product review data to demonstrate our model in three comparisons. Section 4.3 compared the next-item inference models for the multi-item predictions. Section 4.4.2 compared the next-item inference model and the future items inference model. Section 4.4.7 compared the all-item and dense-all-item predictions using our future items inference. These are the comparisons performed extending SASRec. By enabling these models to predict more accurately than conventional next-item predictions, our method can facilitate a workflow in which users can predict and explore many more applications in real-time.

We also examined the impacts of mixed negative sampling, window size, and a time feature on the future items inference model. First, the impact of varying the mix ratio in the mixed negative sampling indicates that more in-batch and random negative samples are effective for dense data and less for sparse data. Second, increasing window size degrades performance in the future items inference (i.e., the window size and the number of predicted items must match) while remaining the same in the next-item inference. Third, adding a time feature improves the model accuracy for the dense dataset while degrading the sparse data model. That said, the model with sparse data prefers no additional features, contradicting our hypothesis.

The method of this thesis is much better at predicting items than the conventional next-item prediction method. This improvement is due to automatically pairing the multiple target items per input item and optimizing the relevance scores for each target item position using mixed negative sampling and Sampled Softmax Loss.

Many of the techniques and analyses used in this thesis generalize to the all-item and dense-all-item prediction models with more than one feature. On dense data, for the dense-all-item model with mixed negative samples and time features, our future items inference method can be applied and outperform the next-item prediction for a set of 3, 5, and 12 items. As for the all-item and dense-all-item models with mixed negative samples (lighter models), our method can still be used, although the prediction of a set of 12 items will underperform the next-item prediction. On sparse data, the future items inference works best without any features. On the other hand, the next-item inference can only be used for the all-item model on a sparse dataset.

Another example of a natural extension of the models we consider is sequence-to-sequence prediction models. Consider, for example, the last input embedding e_s sampled by scheduled sampling [2] and trained with the SASRec's next item prediction model. This e_s differs from the last input embedding e_t we used (Eq. 2.3) in one respect. The model uses a scheduled probability ϵ_i, which decreases as a function of i, to sample an input embedding to compute a relevance score. The next-item inference method of SASRec [11] applies to this sequence prediction with one modification. The model will now use the true last input embedding e_t at the beginning of training and the sampled input embedding e_s with the best likelihood at the end of training. As a result, the model will generate a structured sequence of items until it reaches the end of the window. While the multi-item inference of our method can still be efficient, modeling further features of e_t may call for a sampling-based approach.

Sequence prediction models such as [2], in addition to those with self-attention [19] and better sampling strategies, are directions of future research.

Bibliography

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[2] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. 2015.

[3] Moumita Bhattacharya and Sudarshan Lamkhede. Augmenting netflix search with in-session adapted recommendations, 2022.

[4] David C. Blair. Information Retrieval, 2nd ed. C.J. Van Rijsbergen. London: Butterworths; 1979: 208 pp. 1979.

[5] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. 2005.

[6] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[8] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. 2016.

[9] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. 2018.

[10] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks, 2016.

[11] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation, 2018.

[12] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time, 2019.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[14] Yehuda Koren. Collaborative filtering with temporal dynamics. Association for Computing Machinery, 2009.

[15] Jiacheng Li, Yujie Wang, and Julian McAuley. Time interval aware self-attention for sequential recommendation. 2020.

[16] Julian McAuley. Personalized machine learning, 2022.

[17] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Exploring data splitting strategies for the evaluation of recommendation models, 2020.

[18] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Exploring data splitting strategies for the evaluation of recommendation models, 2020.

[19] Tsvetomila Mihaylova and André F. T. Martins. Scheduled sampling for transformers, 2019.

[20] Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. Pinnerformer: Sequence modeling for user representation at pinterest, 2022.

[21] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. 2010.

[22] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Recommender systems handbook, 2010.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 2014.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[25] Ji Yang, Xinyang Yi, Derek Zhiyuan Cheng, Lichan Hong, Yang Li, Simon Xiaoming Wang, Taibai Xu, and Ed H. Chi. Mixed negative sampling for learning two-tower neural networks in recommendations. 2020.

[26] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. 2019.