Don't recommend the obvious: estimate probability ratios

R.PELLEGRINI, WENJIE ZHAO*, and IAIN MURRAY†, Amazon, UK

Sequential recommender systems are becoming widespread in the online retail and streaming industry. These systems are often trained to predict the next item given a sequence of a user's recent actions, and standard evaluation metrics reward systems that can identify the most probable items that might appear next. However, some recent papers instead evaluate recommendation systems with *popularity-sampled metrics*, which measure how well the model can find a user's next item when hidden amongst generally-popular items. We argue that these popularity-sampled metrics are more appropriate for recommender systems, because the most probable items for a user often include generally-popular items. If the probability that a customer will watch *Toy Story* is not much more probable than for the average customer, then the movie isn't especially relevant for them and we should not recommend it. This paper shows that optimizing popularity-sampled metrics is closely related to estimating point-wise mutual information (PMI). We propose and compare two techniques to fit PMI directly, which both improve popularity-sampled metrics for state-of-the-art recommender systems. The improvements are large compared to differences between recently-proposed model architectures.

Additional Key Words and Phrases: Algorithms, Evaluation metrics and Studies, Novel Machine Learning Approaches

ACM Reference Format:
R.Pellegrini, Wenjie Zhao, and Iain Murray. 2022. Don't recommend the obvious: estimate probability ratios. In *Sixteenth ACM Conference on Recommender Systems (RecSys '22), September 18–23, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3523227.3546753

1 INTRODUCTION

Recommender systems have become a central part of the business for many online retailers and video streaming services; it is estimated that a large proportion of sales or views are driven by these systems [1]. An effective recommender system should be able to provide users with strongly personalised suggestions that are especially relevant to a user's particular interests. Sequential recommender systems have been put forward [2] to capture the dynamical nature of users needs and interests. In a sequential recommender system the task is to predict the next item that the user will interact with, given the sequence of past interactions. These recommenders have been widely adopted by the industry for both e-commerce and video streaming services [3, 4] and have been applied to publicly available datasets using state-of-the-art sequence model architectures [5–7].

Despite their successes and widespread adoption, we argue that the standard task of predicting the most likely item that a user will interact with next does not align very well with strong personalisation, in fact very often the most likely next item a user will interact with is a popular item. However, if the probability that a customer will watch *Toy Story*[1] is not much more probable than under a baseline model for the average customer, then the movie is not especially personalised to that user and the recommendation is of limited usefulness. Some recent papers [6, 8–10] instead evaluate recommendation systems using *popularity-sampled metrics*, which measure how well the model can

*The first two authors contributed equally to this research.
†Also University of Edinburgh. This paper describes work performed at Amazon.
[1]The most popular movie in the MovieLens 20M dataset.

Manuscript submitted to ACM

find a user's next item when hidden amongst generally-popular items. Popularity-sampled metrics are appropriate for personalized recommendation, because they directly measure a system's ability to identify items special to a user over generally-probable items.

This paper shows that popularity-sampled metrics are closely related to the point-wise mutual information (PMI). We propose two different techniques to fit the PMI to optimise the popularity-sampled metrics which both improve popularity-sampled metrics for state-of-the-art recommender systems. The improvements are large compared to differences between recently proposed model architectures.

Another study [11] compares popularity-sampled and uniformly-sampled metrics, and finds that the ranking of recommender systems changes under these two different sampling schemes. We show that these differences are expected, because the sampling strategies correspond to different objective functions. The conclusion will discuss that these metrics and objective functions could be generalized using other baseline models, that use more information than an average popularity model. The methods in this paper would apply to training recommendation systems for these alternative metrics, creating systems that are more personalized when producing recommendations in multiple contexts.

2 SEQUENTIAL RECOMMENDATIONS

Recommender systems focus on finding a subset of items that are useful and interesting to a specific user, and are typically trained on historical user interactions (clicks, purchases, reviews, ratings) with the items. Matrix Factorization (MF) methods use vectors as low-dimensional representations of users and items, and estimate the probability of a user interacting with (or rating highly) an item through an inner product between the vectors for a user and an item. More recently various deep learning techniques have been proposed to model the complex non-linear interactions between users and items [12].

Large-scale services can have hundreds of millions of users, with enough data to fit representations with hundreds of parameters, which would result in MF models with billions of user-specific parameters. Moreover, some users will have very few interactions for fitting their parameters. Asymmetric matrix factorization [13] removes the users' free parameters, representing users as combinations of the parameters of the items they have interacted with. A model can learn how to combine a user's items, and how to use that combination to predict their next item, from many example users. This type of approach lets us fit flexible models for combining items with many free parameters, none of which are specific to a single user.

Given that a user's interactions are gathered over time, it is natural to summarize a user's items with a sequence model. Multiple neural architectures have been used for this task, including a Convolutional Sequential Model [14], Recurrent Neural Networks [2, 8, 10, 15–17] and, more recently, Transformer architectures [18] based on a self attention mechanism, which showed state-of-the-art performances on a variety of datasets [5–7]. For a more in-depth survey of sequential recommender systems we refer to [19].

Good recommendations depend on context and change over time. Early work on the Netflix prize incorporated timestamps of the interactions as a feature in a MF approach [20]. Current sequence modeling architectures naturally model the time series of events, capturing a user's changing interests. Conditioning the predictions on the current time also controls for trends and seasonality of the items. Any other features about the user, the items, or the context of the recommendation, can be added as inputs to the neural network. In this paper we are interested in strongly-personalized recommender systems, and so adopt this flexible neural sequence approach, however the methods that we propose generalize to any probabilistic recommender system including Matrix Factorization. Given current trends, we

experiment with Transformer-based systems that reported state-of-the-art performance [e.g. 5, 6]. Although to keep our experiments simple and comparable, we do not include extra contextual information in our sequences.

We denote a set of users with $U = \{u_1, u_2, \ldots, u_{|U|}\}$, and a set of items with $V = \{v_1, v_2, \ldots, v_{|V|}\}$. The ordered list $S_u = \left[v_1^u, v_2^u, \ldots, v_{n_u}^u\right]$ denotes the items that user u interacted with, in chronological order. The traditional task of sequential recommender system is to predict the next item given the current sequence of events, or more formally to fit the probability distribution

$$P\left(v_{n_u+1} \mid S_u\right). \tag{1}$$

The conditional probability distribution in (1) is used at inference time to rank the items. The k items with largest probability could be shown directly to the user, or chosen as candidates to be re-ranked by another system.

3 METRICS AND EVALUATION

For evaluation we adopt a *leave-one-out* split of the data that has been widely used [e.g. 5, 6]. For each user the last item is used for testing and the item before it is used for validation. The rest of the sequence is used for training. This split means that some training items appear later in time than items in the validation and test sets, which doesn't reflect the reality of a production recommendation system. However, we follow this setup so we can directly compare with previous results.

We focus on the hit rate at k, HIT@k, which can be defined as follows

$$\text{HIT@k} = \frac{1}{|U|} \sum_{i=1}^{|U|} \mathbb{1}_{r(v_i) \le k}, \tag{2}$$

where r is the rank of the test item v_i when mixed with a list of candidate items $C = \left[c_1, \ldots, c_{|C|}\right]$ and sorted using the conditional probability in (1).

There are different strategies to pick the candidate items C: some recent papers [e.g. 6, 9] and a KDD Cup [21, Track 2] evaluated recommenders by sampling the candidates in proportion to their popularity, while another recent paper [11] experimentally studied the differences between sampling the candidates uniformly at random, sampling according to popularity, and using all the items. To understand these choices, we first review the point-wise mutual information (Section 4), which motivates considering an item's overall popularity, and then show the precise connection to popularity-sampled metrics (Section 5).

4 POINT-WISE MUTUAL INFORMATION

The Point-wise Mutual Information (PMI) is a widely-used and long-standing measure of association between two outcomes x and y:

$$\text{PMI}(x, y) = \log \frac{P(y \mid x)}{P(y)} = \log \frac{P(y, x)}{P(x)\, P(y)}. \tag{3}$$

In the context of recommender systems, [22] considered a closely-related score for recommenders based on co-occurrence counts. This paper returns to this issue for neural recommenders. We can think of (3) as a measure of how much more likely two outcomes are to occur together compared to what we would expect by random chance assuming independence. A PMI of zero, corresponding to a probability ratio of one, means that we won't observe the outcomes x and y together more often than if they are independent. In contrast, a large positive value of the PMI implies a

strong association between the outcomes. In neuroscience, Barlow [23] hypothesised that the brain should consider two outcomes "suspicious", or worthy of attention, when $P(x, y) \gg P(x)\,P(y)$, equivalent to $\text{PMI}(x, y) \gg 0$.

In computational linguistics, PMI has been used as a measure of word association for more than 30 years [24]. Simply finding word pairs that frequently co-occur leads to pairs dominated by common words such as "and", that appear in most English documents regardless of context. The most frequent words are often heuristically discarded as "stop words". However, a principle that seems to align much better to the human intuition of "association between words" is to select word pairs with high PMI. See the original paper [24] for example word pairs.

For the same reasons, PMI is an attractive score to rank items to recommend for a user: it avoids recommending products that are not really personalized. High conditional probability $P(v \mid S_u)$ alone does not guarantee that the user is very interested in a specific product, for example in MovieLens 20M, *Toy Story* is much (~60,000 times) more popular than one of the movies from the tail of the distribution. If a user is equally interested in *Toy Story* and the tail movie, the conditional probability would greatly favour the popular one, this behaviour is called popularity opportunity bias by [25]. The PMI, $\log P(v \mid S_u)/P(v)$, corrects for this bias by looking for an increase in interest for an item above its baseline popularity $P(v)$. Because high PMI means that a user is much more likely to interact with that item than the general population, it is a good score for personalized rather than general recommendations.

5 RELATION BETWEEN PROBABILITY RATIOS AND SAMPLED METRICS

To clarify the relation between sampled metrics and estimating probability ratios, we consider a classification task very close to the HIT@k metric's sampling task. For each customer sequence S_u, we create a list of $|C|+1$ items $[v^{(i)}]_{i=0}^{|C|}$. One of the items is set to the true next item v, and the remaining "candidate examples" C are sampled independently from a reference distribution: $v^{(i)} \sim P_C$. The task is for the model to infer the index $k \in [0, \ldots, |C|]$ of the true item.[2]

An ideal Bayes classifier maximizes the conditional probability of a class, which by Bayes rule is proportional to the prior probability of the class and the probability of the observations given the class. For our task, the class is the index k, which has uniform prior probability. Given class k, we assume item k is the user's next item, and the remaining items came from the reference distribution $P_C(x)$:

$$P(k \mid \{v^{(i)}\}, S_u) \propto P(v^{(k)} \mid S_u) \cdot \prod_{i \neq k} P_C(v^{(i)}) \tag{4}$$

$$\propto P(v^{(k)} \mid S_u) \cdot \frac{1}{P_C(v^{(k)})} \prod_{i=0}^{|C|} P_C(v^{(i)}) \tag{5}$$

$$\propto \frac{P(v^{(k)} \mid S_u)}{P_C(v^{(k)})} \quad \text{(dropping constants wrt. } k\text{)}. \tag{6}$$

Thus, ranking items by the probability ratio (6) orders them by their probability of being the correct next item in the sampled task, and thus maximizes the expected HIT@k metric evaluated on the set.

If the candidate items were sampled uniformly, then $P_C(v) = \frac{1}{|V|}$ is a constant and the items should simply be ranked by a standard next item predictor that models $P(v \mid S_u)$. However, if the reference distribution is the popularity distribution, then ordering the items by the ratio (6) is equivalent to ordering the items by the PMI between the customer

[2]Implementation detail: conceptually it's simpler to think of the true item as placed uniformly at random into the list of candidates. However, in our code the true item is always first in the list, and the correct class in the classification task is always $k=0$. As long as the classifier can't use the integer k, the order of the items makes no difference to the classifier we fit.

sequence S_u and the item v. This observation suggests we should estimate probability ratios, or equivalently PMI, to score the items.

The HIT@k metrics actually used in the literature [6, 9, 11], have a slightly different sampling task, because the candidate items are sampled without replacement. When the set size $|C|$ is small compared to the effective number of items under the reference distribution P_C, duplicate items are rare, and the optimal ranking is close to that given by the probability ratios. However, as the set size grows, the metric behaves differently. In the limit where we sample all possible items, $|C|=|V|$, the popularity-sampled metric becomes the same as the uniformly-sampled metric, so the optimal ranking would be given by a standard next-item predictor. Although sampling without replacement seems natural for setting up a classification task, sampling with replacement removes the messy dependence on sample set size $|C|$, and provides a clean link to PMI.

6 METHODS FOR FITTING THE POINT-WISE MUTUAL INFORMATION

This section outlines two approaches to estimate the PMI during the training phase. One is an established reduction to a classification task [26], which is natural in our context given the previous discussion. The other is inspired by simply estimating the two probabilities in the ratio, but with shared parameters, correlating estimation errors in the two probabilities, and making the ratio more efficient to compute at recommendation time.

6.1 Ratio Estimation by Classification

Given that the probability ratio in PMI appears in an ideal classifier (6), a natural way to estimate these ratios is to fit a neural network to the same classification task as Section 5. If we use a network with a standard softmax output,

$$P(k \mid \{v^{(i)}\}, S_u) = \frac{\exp(f(v^{(k)}, S_u))}{\sum_{i=0}^{|C|} \exp(f(v^{(i)}, S_u))} \propto \exp(f(v^{(k)}, S_u)), \tag{7}$$

then comparing to (6), the ideal activations in the final layer are:

$$f(v, S_u) = \log \frac{P(v \mid S_u)}{P_C(v)} + \text{const.} \tag{8}$$

If we sample the candidate items with replacement in proportion to their general popularity, $P(v)$, we recover the PMI up to a constant.

It is also possible to create a binary classification task with a logistic sigmoid output layer instead of the above multiclass task with a softmax output. The binary classifier considers each of the $|C|+1$ candidates in isolation, and decides whether an item v was the user's item ($y=1$), or sampled from P_C as a 'negative example' ($y=0$). The ideal Bayes classifier for this modified task becomes:

$$P(y \mid v, S_u) \propto P(y)\, P(v \mid y, S_u) \propto \begin{cases} \frac{1}{|C|+1} P(v \mid S_u) & y = 1 \\ \frac{|C|}{|C|+1} P_C(v) & y = 0. \end{cases} \tag{9}$$

Normalizing and rearranging, we can write the ideal classifier output using the logistic sigmoid, σ:

$$P(y=1 \mid v, S_u) = \sigma\left(\log \frac{P(v \mid S_u))}{P_C(v)} - \log(|C|)\right), \quad \text{where } \sigma(a) = 1/(1+\exp(-a)). \tag{10}$$

Thus if we fit a binary classifier with a standard sigmoid output to this alternative task, with samples drawn in proportion to popularity $P(v)$, the activation of the classifier estimates $\text{PMI}(v, S_u) - \log(|C|)$.

An attraction of the classification approach is that each training update for a customer sequence S_u only involves computing and summing functions of the $|C|+1$ items in the classification task, whereas a next-item predictor requires computing a softmax probability vector over all possible items that could come next. Given enough data, the classification methods should be consistent for any number of candidates $|C|$. However, sampling-based strategies for training models with large softmaxes usually hurt performance (Ruder provides a nice review [27]). If $|C|$ is sufficiently large, we expect to perform reasonably on a HIT@k metric, but we don't necessarily expect the model to accurately estimate the probability ratios for all items in practice.

6.2 Embedded prior model

Another natural way to fit the probability ratio in (6) is to estimate both the customer-specific predictions $P(v \mid S_u)$ and the item popularity distribution $P(v)$ separately, and plug these into the ratio. However, in preliminary experiments using separate probability estimates gave poor HIT@k metrics, barely improving (less than a percent) on the model that estimates the conditional probability. We note that it isn't possible to estimate small probabilities reliably, and dividing by a small value can lead to a much larger relative error than in the value itself.

An alternative approach identifies that an ideal next-item predictor can be written as the popularity distribution over items $P(v)$, corrected by the PMI:

$$P(v \mid S_u) = \exp\left(\mathrm{PMI}(v, S_u) + \log P(v)\right). \tag{11}$$

$P(v)$ can be interpreted as the prior distribution over the next item that a user will interact with, because it is the distribution we would assign before seeing the user-specific data. Rather than correcting for the prior at the end, inspired by (11) we embed the prior into the parametric form of a next-item predictor model:

$$P(v \mid S_u) \propto \exp\left(f_w(v, S_u) + g_z(v)\right), \tag{12}$$

where neural network functions $f_w(v, S_u)$ and $g_z(v)$, with parameters w and z, are intended to model $\mathrm{PMI}(v, S_u)$ and $\log P(v)$ up to additive constants. We train the overall model to be a next-item predictor as usual, minimizing the negative log likelihood

$$\mathcal{L}_C = -\sum_u \log P(v_u \mid S_u), \tag{13}$$

where v_u is the next item user u interacted with. To ensure that g_z models the log-prior probabilities as intended, we use g_z in a standard 'softmax' model of $P(v)$, and compute a second loss to predict the items for each user without looking at their sequence:

$$\mathcal{L}_P = -\sum_u \log P(v_u), \quad \text{where } P(v_u) = \frac{\exp(g_z(v_u))}{\sum_{v' \in V} \exp(g_z(v'))}. \tag{14}$$

In this paper, $P(v)$ is a single fixed discrete distribution, so we could also consider fitting it using counts. However, training it as a neural network is convenient, and would allow us to easily condition on other information about the general context of the recommendation (discussed further in Section 8). As standard for neural network training, we will use a gradient-based optimizer (Adam [28]), but there is a choice of how to fit the two functions:

(1) We can fit the prior g_z in an initial training loop, freeze the parameters z and then fit f_w.
(2) We can fit the models together in one training loop by minimizing the sum of the two loss functions with respect to all of the parameters. The parameters z for the prior appear in both terms, whereas the parameters for the PMI appear only in the user-specific loss.

Table 1. Dataset statistics after pre-processing

Dataset	# users	# movies	# actions	Avg. length
ML-1M	6,040	3,416	1m	163.5
ML-20M	138,493	26,744	20m	144.4

(3) We can use the same objective as 1. in a single training loop, by doing 2. except ignoring the user-specific loss $\mathcal{L}_C$ when computing the gradients for the parameters of the prior. (Most deep learning frameworks provide a way to stop, block, or detach gradient computations.)

In preliminary experiments 1 and 3 had equivalent performance and tended to perform slightly better than 2. As it's more convenient to have a single training loop, we report the results for 3 in our experiments.

After training, we don't need to use two neural network functions at inference time. Instead, to estimate the PMI we can discard g_z, and only need to evaluate f_w, which directly gives us an estimate of PMI up to an additive constant. The constant is independent of the item v, and so is not required when ranking items to compute HIT@k.

7 EXPERIMENTS

We design experiments in order to answer the following research questions:

RQ.1 Does probability ratio estimation improve the established popularity-sampled HIT@k metric?

RQ.2 How does the established popularity-sampled HIT@k performance vary as we change the candidate sample size in training, and sample with and without replacement?

RQ.3 How does the performance of the proposed methods vary as we change the number of samples in the HIT@k when sampling with replacement?

RQ.4 Does probability ratio estimation recommend fewer popular items?

7.1 Datasets and Experimental Setup

We evaluate the recommender models on a popular benchmark dataset which contains movie ratings collected from a non-commercial movie recommendation website MovieLens[29]. We choose two frequently used versions, MovieLens 1M (**ML-1M**)[3] and MovieLens 20M (**ML-20M**)[4], which have 1 million and 20 million ratings respectively. We follow the standard preprocessing steps used in previous papers [5, 6, 11]. The existence of a rating is treated as a positive action, and rating scores are not used. The actions are grouped by users and ordered by timestamps, so each user has a sequence of movies they have rated. Previous papers [5, 6] claim that the users and the movies with fewer than 5 interactions have been discarded. However, to reproduce the statistics reported, we had to skip the filtering step for ML-20M and only filtered out movies with fewer than 5 ratings in ML-1M. See Table 1 for the dataset summary statistics after pre-processing.

As described in Section 3, each user's final interaction is taken as a test item, which the trained models rank relative to other candidate items given the user's previous interactions. To evaluate the correctness of the ranked lists, we look at HIT@k metrics combined with different candidate sampling strategies. Additionally, we also look at the average index of top k recommendations (Avg. Index@k). In our models, we assign an index to the items according to their popularity.

[3] https://grouplens.org/datasets/movielens/1m/
[4] https://grouplens.org/datasets/movielens/20m/

The most popular item will get assigned index one while the least popular will be assigned index $|V|$. Therefore, the average index is a measure of how far into the tail of the item distribution our recommendations explore on average. The higher it is the more items from the tail we are recommending.

7.2 Model Architectures and Implementation Details

To test the effectiveness of estimating PMI during training, we use SasRec [5] and Bert4Rec [6], two state-of-the-art sequential recommender systems based on Transformers [18]. These models are designed to estimate the conditional probability of the next item a user will interact with, given a sequence of previously interacted items, see (1). We modify SasRec with the method described in Section 6.1 and Bert4Rec with the methods described in Section 6.1 and Section 6.2 so they estimate PMI's probability ratios instead of the conditional probabilities.

- **Self-attentive Sequential Recommendation Model (SasRec) [5]** uses a unidirectional Transformer encoder that only learns left-to-right relationships in the sequence. At training time, every item in a sequence, except the first item, is treated as a prediction target. The model uses single-headed self-attention layers to encode the subsequence up to the current time into a vector. The dot product of that user sequence vector with item v gives an activation r_v. For each target, an alternative candidate or 'negative' item v' is uniformly-sampled from the set of all items that are not in the input subsequence or equal to the target, and an alternative activation $r_{v'}$ is computed. The contribution to the loss for an item and negative item pair is[5]

$$\mathcal{L} = -\log(\sigma(r_v) - \log(1 - \sigma(r_{v'}))). \tag{15}$$

  This contribution corresponds to the loss for a positive ($y=1$) example and a negative ($y=0$) example for the binary classifier (10) for probability ratio estimation. Because the sample was drawn uniformly, we identify that the activations correspond to $P(v \mid S_u)$, the log probabilities for a next-item predictor. To make this theoretical connection, the negative sample should have been drawn with replacement, but there is little difference: the negative item would rarely equal the true item if it were sampled with replacement.[6]
- **Bert4Rec [6]** has a similar model architecture to SasRec. The main difference is that it uses a bidirectional multi-headed Transformer to encode the sequences, i.e. relationships between right-to-left item pairs are also learned. Bert4Rec is also trained on a different task than SasRec. As its name suggests, it is trained on the Cloze task, like in the BERT language model [30]. Some items in a sequence are randomly masked and the model predicts these masked items by their surrounding contexts. In the output layer, Bert4Rec calculates the probability distribution of the masked item by applying a softmax on the dot product between the final hidden state of the masked item and the entire item embedding table. It is then trained to minimize the negative log probability of the true labels. Computing a softmax distribution over all possible items is more expensive than the sigmoid output for two items in SasRec. However, Bert4Rec only needs to make these predictions for masked items. At test time we can use Bert4Rec as a next-item predictor, estimating $P(v \mid S_u)$.

To better align the training objective with the popularity-sampled HIT@k metrics, we propose to modify the loss functions from estimating conditional probability $P(v \mid S_u)$ to PMI's probability ratio $P(v \mid S_u)/P(v)$.

[5]This sampling procedure and loss corresponds to the official code for the SasRec paper, https://github.com/kang205/SASRec/. The paper [5] described a different cross entropy loss (Section III.E, top of p5), which for each target sums over all possible negative examples (far more expensive). The paper probably intended to divide this sum by the number of possible negatives, because then the code's sampled loss would be an unbiased Monte Carlo estimate of that corrected loss.

[6]Another detail is that the negative examples can't be items from the user's past sequence. In this dataset, items in the sequence don't repeat, so if we saw such an item, we know its probability is zero. Excluding these uninteresting candidates implicitly renormalizes the prior over the feasible set, which just changes the remaining non-zero probability ratios by a constant.

Table 2. Hyperparameters used in model training and reproduced HIT@10 results in comparison to results reported in [6, 11]. The HIT@10 is evaluated on the positive target along with 100 negative items sampled by popularity. # blocks are the number of transformer blocks. # heads are the number of attention heads.

Model	Dataset	Batch size	Emb. dim.	Hidden dim.	# blocks	# heads	Dropout	[6]	[11]	ours
SasRec	ML-1M	128	64	64	2	1	0.2	0.663	0.656	0.658
	ML-20M	128	256	256	2	1	0.2	0.714	0.706	0.719
Bert4Rec	ML-1M	256	64	256	2	2	0.3	0.697	0.663	0.664
	ML-20M	256	256	1024	2	8	0.1	0.747	0.732	0.720

The smallest change to SasRec is to sample its negative items in proportion to their popularity. Because it uses a binary classification task with $C=1$ candidate, the activations should now correspond to PMI (10). Given the large expense of predicting every item, rather than just the final item, we did not try the embedded prior approach (Section 6.2), which requires a large softmax computation for each prediction.

Bert4Rec already uses large softmaxes so we did apply the method of Section 6.2, adding a prior model to the activations at training time. We also tried reducing the size of the softmaxes by randomly sampling candidates, which can also give us PMI's probability ratios, as described in Section 6.1.

Hyperparameters We reimplemented SasRec and Bert4Rec in PyTorch [31]. We could not reproduce the reported results with the hyperparameters provided by [6], so performed our own hyperparameter search and ended up using the following configuration. We trained the models using Adam optimizer [28] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and l_2 weight decay of 1e-7. We used a linear learning rate scheduler with maximum learning rate set to 1e-3. Max sequence length is set to 200 for both ML-1M and ML-20M. The remaining hyperparameters are model-dependent and can be seen in Table 2. We evaluated the models on HIT@10 with $|C| = 100$ popularity-sampled negative candidates. Our implementation of SasRec achieves very similar results to those reported by [6, 11]. Our results for Bert4Rec are slightly worse than the original implementation of [6], but close to the results reported by [11]. The small difference might be caused by the different choices of hyperparameters and default settings in neural network frameworks.

7.3 Conditional Probability vs. Probability Ratio

In this section we compare fitting conditional probability and probability ratio for SasRec and Bert4Rec to answer **RQ.1**. The original objective of SasRec and Bert4Rec are both to estimate the conditional probability distribution of the next item a user is going to interact with given their interaction sequence. In order to improve their performance on popularity-sampled HIT@k metrics, we modify their training objectives to estimate the ratio between the conditional probability and the prior probability. The results show that estimating the probability ratio consistently improves the performance on popularity-sampled HIT@k, which verifies our hypothesis that fitting probability ratio is better aligned with popularity-sampled metrics. Table 3 shows the experiment results. We observe that fitting probability ratio leads to 4.90% and 7.00% higher HIT@10 than original SasRec on ML-1M and ML-20M, while Bert4Rec was reported 4.15% and 4.72% higher HIT@10 in [6]. For Bert4Rec, both the classification and embedded prior approach improve the performance by a substantial amount. The embedded prior approach has a small advantage over the classification approach, which might be due to the small candidate set used to evaluate HIT@k. We show in later sections that calculating the full softmax is more beneficial when the item set is large.

Table 3. Popularity-sampled HIT@k results of fitting conditional probability and probability ratio for SasRec and Bert4Rec on ML-1M and ML-20M datasets. We fit probability ratio for Bert4Rec using two different approaches, i.e. classification and embedded prior. The classification approach uses a sampled loss, while the embedded prior approach uses a full softmax. In the classification approach, we report the results with $|C| = 100$ candidates for both ML-1M and ML-20M. We include HIT@k results for $k = \{1, 5, 10\}$. For all k, we sample $|C| = 100$ negative items without replacement according to popularity distribution. For each base model and each dataset, the best HIT@k value is marked as bold. We also include the percentage improvements on HIT@10 results by fitting probability ratios instead of the baseline that predicts the next item (conditional probability (1))).

Model	Dataset	Objective	Loss	HIT@1	HIT@5	HIT@10	HIT@10 Improv.
SasRec	ml-1m	Conditional	Sampled	0.2270	0.5252	0.6575	—
		Prob. ratio	Sampled	**0.2454**	**0.5623**	**0.6897**	4.90%
SasRec	ml-20m	Conditional	Sampled	0.2751	0.5856	0.7190	—
		Prob. ratio	Sampled	**0.3277**	**0.6451**	**0.7693**	7.00%
Bert4Rec	ml-1m	Conditional	Full softmax	0.2483	0.5382	0.6644	—
		Prob. ratio	Full softmax	**0.3007**	0.5917	0.7045	6.03%
		Prob. ratio	Sampled	0.2916	**0.5927**	**0.7081**	6.58%
Bert4Rec	ml-20m	Conditional	Full softmax	0.3067	0.5968	0.7200	—
		Prob. ratio	Full softmax	0.3794	0.6695	0.7817	8.56%
		Prob. ratio	Sampled	**0.3936**	**0.6933**	**0.8018**	11.37%

7.4 Number of samples in classification task

The plot in Fig. 2 sheds light on **RQ.2**: increasing the number of samples of the classification task improves the performances on the popularity-sampled HIT@k metric. Changing the sampling from uniform to popularity has a large effect on the metric, while sampling with or without replacement at training time does not seem to have a large effect. When the sample size is around a tenth of the number of items, $\frac{|C|}{|V|} \sim 0.1$, the classification task has similar performance to the embedded prior model and it is significantly cheaper to train.

We also observe that training the model with 1 or 10 candidates sampled uniformly at random does not change the performances very much; we think that it is an indication that this task is very simple when the number of samples is small and might lead to models that don't generalize very well when used for other tasks.

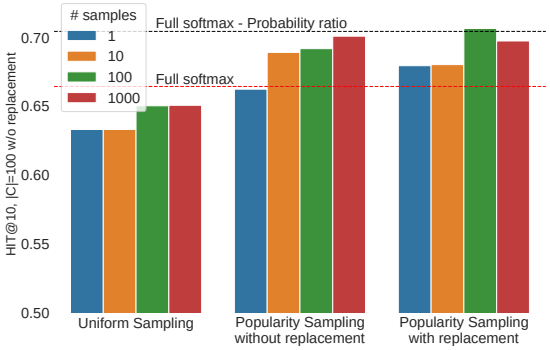


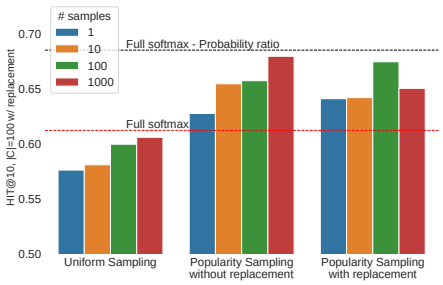
Fig. 1. Popularity-sampled HIT@10 results for Bert4Rec model on ML-1M for different sampling methods during training. HIT@10 metric is sampled without replacement and $|C| = 100$. y-axis starts from 0.5.

Table 4. Popularity-sampled HIT@10 results for Bert4Rec model on ML-1M. When training on the classification task, we compare different sampling methods and numbers of samples. The models are evaluated on HIT@10 metrics that also differ in sampling methods and numbers of samples. Best results of each metric is marked as bold.

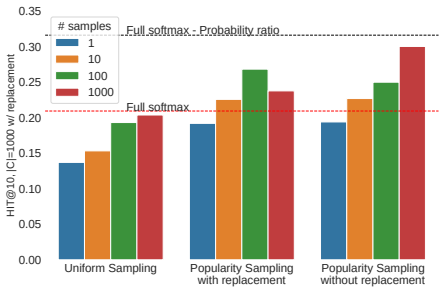
Objective	Loss	Num. samples in loss	Replacement in loss	HIT@10 |C|=100 w/o replacement	HIT@10 |C|=100 w/ replacement	HIT@10 |C|=1000 w/ replacement
Conditional	Full softmax	—	—	0.6644	0.6123	0.2094
Prob. ratio	Full softmax	—	—	0.7045	**0.6853**	**0.3162**
Conditional	Sampled	1	—	0.6331	0.5762	0.1371
Conditional	Sampled	10	Yes	0.6331	0.5810	0.1533
Conditional	Sampled	100	Yes	0.6503	0.5997	0.1932
Conditional	Sampled	1000	Yes	0.6507	0.6060	0.2038
Prob. ratio	Sampled	1	—	0.6795	0.6411	0.1921
Prob. ratio	Sampled	10	Yes	0.6803	0.6421	0.2258
Prob. ratio	Sampled	100	Yes	**0.7081**	0.6747	0.2684
Prob. ratio	Sampled	1000	Yes	0.6975	0.6505	0.2377
Prob. ratio	Sampled	10	No	0.6892	0.6546	0.2270
Prob. ratio	Sampled	100	No	0.6919	0.6575	0.2500
Prob. ratio	Sampled	1000	No	0.7010	0.6796	0.3003

7.5 Number of samples in HIT@k with replacement

In this section we discuss the results relevant to **RQ.3**. We are only considering HIT@k with replacement (Section 5), because when we increase the number of candidates $|C|$ in the metric without replacement the task becomes more and more similar to the uniformly sampled metric. In Fig. 2a it is apparent that when the number of samples in the metric is small compared to the number of products, $|C| \ll |V|$, the classification task performs as well or slightly better than the embedded prior model. However when the number of samples increases to be of the same order of magnitude as the number of products, the embedded prior outperforms the classification task (Fig. 2b). We therefore recommend using the embedded prior model when the recommender system needs to pick a few candidates from a large set of items.



(a) HIT@10, $|C| = 100$ sampled with replacement.



(b) HIT@10, $|C| = 1000$ sampled with replacement.

Fig. 2. Popularity-sampled HIT@10 results for Bert4Rec model on ML-1M for different sampling methods during training. Note that y-axis starts at 0.5 in 2a and 0 in 2b.

Table 5. Average index of the top 10 recommendations for test set samples produced by Bert4Rec models trained on different objectives for ML-1M dataset. The highest average index is marked in bold.

Objective	Loss	Avg. Index@10
Conditional	Full softmax	528.0
Prob. ratio	Full softmax	972.6
Prob. ratio	Sampled	**1082.3**

Table 6. Example recommendations for a random user in ML-1M dataset. On the left are the top 10 movies recommended by the original Bert4Rec model that estimates conditional probability. On the right are the top 10 movies recommended by the modified Bert4Rec that estimates probability ratio by training with an embedded prior model. Count refers to the number of a movie's occurrence in the dataset.

Conditional			Prob. ratio		
Movie title	Genre	Count	Movie title	Genre	Count
Indiana Jones and the Temple of Doom (1984)	Action|Adventure	1127	No Way Out (1987)	Thriller	355
Lethal Weapon (1987)	Action|Comedy|Crime|Drama	1627	Last Temptation of Christ, The (1988)	Drama	254
Star Trek: The Wrath of Khan (1982)	Action|Adventure|Sci-Fi	1448	For Your Eyes Only (1981)	Action	425
Batman (1989)	Action|Adventure|Crime|Drama	1431	Steel Magnolias (1989)	Drama	408
Predator (1987)	Action|Sci-Fi|Thriller	1297	Children of a Lesser God (1986)	Drama	331
Top Gun (1986)	Action|Romance	1010	Man with the Golden Gun, The (1974)	Action	461
Star Trek IV: The Voyage Home (1986)	Action|Adventure|Sci-Fi	1126	F/X (1986)	Action|Crime|Thriller	701
Abyss, The (1989)	Action|Adventure|Sci-Fi|Thriller	1715	Shaft (2000)	Action|Crime	421
Untouchables, The (1987)	Action|Crime|Drama	1127	Shaft (1971)	Action|Crime	222
F/X (1986)	Action|Crime|Thriller	701	Dead Men Don't Wear Plaid (1982)	Comedy|Crime|Thriller	367

7.6 Popularity of recommended items

In this section we would like to verify that estimating PMI does lead to less popular items in the recommendations produced (**RQ.4**). In Table 5 we can see that both approaches of estimating probability ratio result in a much higher average index of the top 10 recommended movies than the original Bert4Rec model, which means our models explore more tail movies. From a qualitative perspective, we examine the recommendations produced by the unmodified Bert4Rec model and our proposed model in Table 6. We observe that our method recommends less popular movies in general, e.g. the movie counts are smaller than those recommended by the original Bert4Rec models. Also, the movies genres are more diverse, whereas Bert4Rec mainly recommends Action movies. Therefore, we conclude that estimating probability ratio does help recommend fewer popular items.

8 CONCLUSIONS AND FUTURE WORK

In this work we have drawn a connection between the probability ratio in PMI and HIT@k metrics where the candidates are sampled in proportion to their popularity. Extensive results on MovieLens 1MM and MovieLens 20MM show that training recommender systems to estimate PMI, rather than as next-item predictors, outperforms state-of-the-art baselines. The change of objective function can be more important than architectural changes.

We also compared three different methods to estimate the PMI:

M.1 Estimate the conditional distribution and the prior separately

M.2 Ratio estimation by classification

M.3 Embedded prior model

Our experiments shows that **M.2** and **M.3** outperform **M.1**, while **M.2** performs as well as **M.3** and is more computationally efficient when the number of samples is small. If the recommender will be used in production to select recommendations among a very large number of items we recommend **M.3** which is the one that performs the best when the number of candidates is large.

We evaluated our models on HIT@k (Table 4) where the candidates are sampled according to the popularity distribution with replacement. We argue that the popularity sampled metric is more appropriate than the randomly sampled metric for strongly personalised models because it is more closely related to PMI which avoids recommending an item to a user if the user is not more interested in the item than the general population.

The authors of [22] pointed out that it is very difficult to choose a score to rank items that works in all settings. They discuss some alternatives, but said they had to resort to controlled online experiment to pick the best one for a specific use case. While we think that using the PMI is a better choice than the conditional probability for a strongly personalized recommender we don't recommend to use it for every use case. Given no information about a user, (6) reverts to uniform scores. We could raise the prior to an inverse-temperature β so that the popularity of items retains some influence given no other information:

$$\frac{P(v^{(k)} \mid S_u)}{P(v^{(k)})^{\beta}}, \quad 0 \leq \beta \leq 1. \tag{16}$$

The algorithms in this paper and metrics still apply, just using a different prior distribution.

We note that the same ideas still apply if we wanted to use more complex prior models in (6). For example during Christmas, some movies are generally more popular, and we might want to avoid recommending obvious Christmas movies to our users. One approach might be to condition on time, t, in both the numerator and the denominator of (6), scoring items v for customer sequence S_u using $P(v^{(k)} \mid S_u, t)/P_C(v^{(k)} \mid t)$. This ideas in this paper have shown how to fit models to estimate these scores, and that a suitable metric for development is HIT@k with candidate items sampled from a time-specific distribution. We believe that there is plenty of future work in exploring the space of possible probability ratio scores, and more complex prior models.

9 BROADER IMPACT

This work proposes training objectives and methods for recommender systems to create strongly-personalized systems that don't recommend obvious popular products. We also encourage the use of existing metrics that reward systems trained in these ways, and discuss variants of the metrics. If these methods and metrics become prevalent, recommender systems will become more personalized and it will benefit users with niche interests and vendors of items from the tail of the distribution that are less likely to be recommended by current state-of-the-art systems. On the other hand, recommender systems have many fewer instances of tail items to learn from, so the estimates of the probability ratio for the tail items are noisy. If our proposed method becomes mainstream there will be an increased risk to show items that look irrelevant or strange to the customer.

We do not believe that there exist a single correct metric for all recommender systems, on-line A/B tests are necessary to find the most appropriate metric for a specific use case. However, we hope that this work brings clarity to the link between metrics and training methods, and will encourage more experimentation with different types of metrics that align better with the specific use cases of different recommender systems.

REFERENCES

[1] Steve Noble Ian MacKenzie, Chris Meyer. How retailers can keep up with consumers, 2013.

[2] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2016.

[3] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery.

[4] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. Behavior sequence transformer for e-commerce recommendation in alibaba. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, DLP-KDD '19, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206, 2018.

[6] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 1441–1450, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. *Transformers4Rec: Bridging the Gap between NLP and Sequential / Session-Based Recommendation*, page 143–153. Association for Computing Machinery, New York, NY, USA, 2021.

[8] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 241–248, New York, NY, USA, 2016. Association for Computing Machinery.

[9] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y. Chang. Improving sequential recommendation with knowledge-enhanced memory networks. In *The 41st International ACM SIGIR Conference on Research amp; Development in Information Retrieval*, SIGIR '18, page 505–514, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 843–852, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Alexander Dallmann, Daniel Zoller, and Andreas Hotho. *A Case Study on Sampling Strategies for Evaluating Neural Sequential Item Recommendation Models*, page 505–514. Association for Computing Machinery, New York, NY, USA, 2021.

[12] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 173–182, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[13] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[14] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 565–573, New York, NY, USA, 2018. Association for Computing Machinery.

[15] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. Sequential user-based recurrent neural network recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 152–160, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 1419–1428, New York, NY, USA, 2017. Association for Computing Machinery.

[17] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, aug 2017.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008, 2017.

[19] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations. *ACM Trans. Inf. Syst.*, 39(1), nov 2020.

[20] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 447–456, New York, NY, USA, 2009. Association for Computing Machinery.

[21] Gideon Dror, Yehuda Koren, and Markus Weimer, editors. *Proceedings of KDD Cup 2011*, volume 18 of *Proceedings of Machine Learning Research*. PMLR.

[22] Brent Smith and Greg Linden. Two decades of recommender systems at amazon.com. *IEEE Internet Computing*, 21(3):12–18, 2017.

[23] H. B. Barlow. Cerebral cortex as model builder. In D. Rose and V. G. Dobson, editors, *Models of the Visual Cortex*. John Wiley & Sons Ltd., 1985.

[24] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, 1990.

[25] Ziwei Zhu, Yun He, Xing Zhao, Yin Zhang, Jianling Wang, and James Caverlee. Popularity-opportunity bias in collaborative filtering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 85–93, New York, NY, USA, 2021. Association for Computing Machinery.

[26] M. Sugiyama, T. Suzuki, and T. Kanamori. *Density ratio estimation in machine learning*. Cambridge University Press, 2012.

[27] Sebastian Ruder. On word embeddings – part 2: approximating the softmax, 2016. http://ruder.io/word-embeddings-softmax.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2015. arXiv:1412.6980.

[29] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.