Modeling Online Behavior in Recommender Systems: The Importance of Temporal Context

Milena Filipovic[1*], Blagoj Mitrevski[1*], Diego Antognini[1], Emma Lejal Glaude[2], Boi Faltings[1], Claudiu Musat[2]

[1]Ecole Polytechnique Fédérale de Lausanne, Switzerland
[2] Swisscom, Switzerland
firstname.lastname@{epfl.ch, swisscom.com}

Abstract

Simulating online recommender system performance is notoriously difficult and the discrepancy between the online and offline behaviors is typically not accounted for in offline evaluations. Recommender systems research tends to evaluate model performance on randomly sampled targets, yet the same systems are later used to predict user behavior sequentially from a fixed point in time. This disparity permits weaknesses to go unnoticed until the model is deployed in a production setting. We first demonstrate how omitting temporal context when evaluating recommender system performance leads to false confidence. To overcome this, we propose an offline evaluation protocol modeling the real-life use-case that simultaneously accounts for temporal context.

Next, we propose a training procedure to further embed the temporal context in existing models: we introduce it in a multi-objective approach to traditionally time-unaware recommender systems. We confirm the advantage of adding a temporal objective via the proposed evaluation protocol. Finally, we validate that the Pareto Fronts obtained with the added objective dominate those produced by state-of-the-art models that are only optimized for accuracy on three real-world publicly available datasets. The results show that including our temporal objective can improve recall@20 by up to 20%.

1 Introduction

The significance of recommender systems. In an increasingly digital world, institutions must accurately anticipate the next movie, song, product, or job that a user will interact with. From YouTube to Netflix, Spotify to Amazon, Facebook to Linkedin, recommender systems are a staple of our daily routines. They influence how we perceive our environment, from media content to human relationships.

Standard evaluation does not reflect real-life use-cases. Traditional methods of evaluation that entail random sampling over a long period of time are perfect if the system is designed to remain unchanged for an equally long and predefined period. However, if the system is to be used in a dynamic setting, the way it is evaluated must reflect that. Inadequate evaluation techniques can lead to a false confidence, which is especially detrimental in commercial settings.

Evaluating a recommender system can be done *online* or *offline*. *Online* evaluation entails deployment of the recommender in a commercial setting. While this may be the best way to measure the real-life impact of a system, it is also costly and therefore rarely used in research and benchmarking. *Offline* evaluation is far more common in recommender systems research. Here, the model is evaluated on historical data, by selecting some portion of the data to train on, while some other subset is used for performance testing.

Many existing recommenders ignore temporal information. Most recommender systems fall into one of the two main categories: content-based and collaborative filtering. Content-based filtering relies on recommended items having similar attributes to those that the user has previously interacted with. Collaborative filtering methods base the recommendation on items bought by similar users.

However, most models ignore temporal information, except a subtype of recommenders that focuses on the order of interactions. Time-aware recommender systems – called sequence-aware recommenders – introduce additional information to the interactions: the time at which the users' decisions were made. Consequently, there is a need to incorporate the temporal context into traditional recommenders.

In this work, we first focus on the importance of temporal dynamics in recommender system creation, training, and offline evaluation. While much effort is directed towards establishing the importance of proper evaluation design, it is generally focused on implementing relevant metrics to avoid under- or over-estimating real-world performance (Aggarwal et al. 2016). We draw attention to the lack of standardization in this domain, and the differences between research settings and the systems' ultimate applications. Then, we propose two temporal evaluation protocols and show how they attain a closer approximation of the real-life conditions in which recommender systems are deployed.

Second, we present a multi-objective approach to time-unaware recommender systems to incorporate the temporal context without any change in the model architecture. We demonstrate the advantages of such systems. Then, we introduce recency as an objective and as a means to include temporal dynamics in typically time-independent recommender systems. We also provide a measure of recency in the form of a performance metric.

Experiments on three real-world publicly available

[*]Work done while at EPFL and Swisscom.

datasets show both improvements in recency and relevance. Finally, we demonstrate that the Pareto Fronts obtained with the added objective dominate those produced by state-of-the-art models.

To the best of our knowledge, this is the first study quantifying the difference in recommender system performance when evaluated using methods that model real-world environments, as opposed to traditional techniques. After demonstrating the impact of recency, we show that a recommender system can be optimized for both the relevance and recency objectives simultaneously.

To summarize, the main contributions of this paper are as follows:

- We demonstrate how commonly used evaluation protocols do not provide adequate modeling of real-world deployment settings. To combat this, we propose two evaluation techniques to facilitate offline modeling of online production environments that inherently incorporate temporal dynamics;
- We introduce a recency function that can be utilized to create a recency objective. We show that optimizing for both recency and relevance (Milojkovic et al. 2019) leads to solutions that dominate those optimized just for relevance in both dimensions.

2 Related Work

2.1 Evaluating Recommender Systems

Traditional Recommender Systems. Inputs and outputs share similarities with classification and regression modeling: a class variable is predicted from a set of given features. Therefore, given that recommendation tasks can be seen as a generalization of these, some evaluation techniques used for classification are transferrable to recommender systems.

In collaborative filtering research, recommenders are generally evaluated either through *strong* or *weak generalization*, characterized by (Marlin 2004). In both approaches, models are trained on observed interactions and validated or tested on those that are held-out. *Weak generalization* is introduced in (Breese, Heckerman, and Kadie 1998), where the held-out set is created through random sampling of the available interactions. *Strong generalization* differs by taking disjoint sets of users for the training, validation, and testing sets. Following this, some interactions are held-out from the validation and test sets and then approximated using the recommender. Methods that encode user representation cannot apply *strong generalization*, as they cannot generate outputs for previously unseen users. An example of the *strong generalization* approach can be seen in (Liang et al. 2018), whereas (Ning and Karypis 2011; Wu et al. 2016; Rendle et al. 2012) all use *weak generalization*.

Several of these works emphasize that the application of their recommender system would be in predicting future user actions, yet all validation and testing is done with randomly selected interactions. This can break the time linearity as the knowledge of future interactions can help predict an anterior interaction.

Temporal Recommender Systems. They denote time-aware models (TARS), and incorporate time explicitly or implicitly. Temporal recommender systems include, but are not limited to, sequence-aware recommender systems (SARS).

As previously stated, SARS can be evaluated similarly to TARS. (Campos, Díez, and Cantador 2014) provide an extensive overview of possible evaluation techniques, which served as an inspiration and point of reference for this work.

While traditional evaluation protocols may be used on temporal recommenders, it is more representative to preserve the temporal ordering between interactions since this is something that the recommender aims to learn. By extension, train, validation, and test splits should also be ordered.

(Quadrana, Cremonesi, and Jannach 2018) state that they were unable to find a consensus among evaluation protocols used in recent sequence-aware recommender work, which is mirrored in our findings. Yet we did determine that most recent SARS focus only on next item prediction, meaning they output one recommendation. They also typically employ certain target item conditions to decrease computational cost (Campos, Díez, and Cantador 2014). The target item conditions determine the (sub)set of items for which a recommender should produce predictions and are specific for top-N recommendation evaluation. The reduction of the computational costs is generally done through conditions that rank one ground truth item against a set of other items false items. Examples can be found in (Sun et al. 2019; Kang and McAuley 2018; Hidasi and Karatzoglou 2018). We return to the problem of subsampling in Section 3.

2.2 Temporal Context in Recommender Systems

In this paper, we introduce the concept of recency. An important note is that there are multiple definitions of recency in recommender systems literature. In fact, this lack of consensus has persisted for years.

(Ding, Li, and Orlowska 2006; Vinagre, Jorge, and Gama 2015) treat the recency of an item as an attribute that is user-dependent. The value is determined by the last time the user interacted with a given item. (Chakraborty et al. 2017; Gabriel De Souza, Jannach, and Da Cunha 2019) also claim to incorporate recency into their research: when recommending news articles, they measure recency as the age of the item on the platform. Our analysis will follow the latter definition.

3 Proposed Evaluation Protocols

We propose that the temporal dimension should be considered when evaluating the performance of any recommender.

While random sampling may be an appropriate target selection technique for some classification or regression tasks, we argue that this is not the case when it comes to predicting a user's subsequent move.

Unlike the vast majority of evaluation methods applied to traditional recommenders, temporal recommender systems literature does model the passage of time. However, as stated above, the performance is often computed over a subset of the itemset and the user's true chosen item. The argument is that subsampling is necessary due to the complexity of

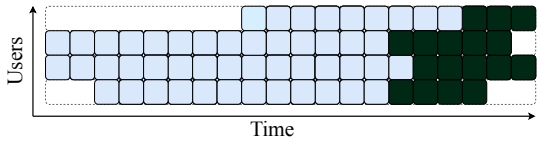


Figure 1: Proportional Temporal Selection.

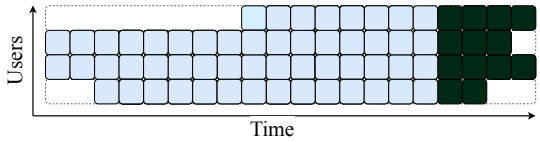


Figure 2: Strict Temporal Cutoff.

the ranking task. While this has some validity, itemsets of around 10,000 datapoints can be ranked highly efficiently, especially when taking into consideration recent advancements in machine learning libraries and GPU programming. Therefore, we do not utilize subsampling in our work.

The adoption of a recommender system in real scenarios has two major phases. The first, called the development phase, is purely offline and theoretical. In this part, three separate sets of data must be created: a training set that the model will use to learn item and user representations, a validation set for hyperparameter tuning, and a test set to evaluate how well the model performs. The second, called the deployment-ready phase, include interactions with end-users. The maximum amount of data is leveraged to train a model with as much information as possible, evaluate its performance, and then deploy it into production. In this case, only two sets are needed: a training and a validation set.

One downside of collaborative filtering methods is that most models are incapable of incorporating new items without retraining. While ways to alleviate this problem have been explored (Luo, Xia, and Zhu 2012), the issue remains widespread and worthy of more study, but lies outside the scope of this paper.

Therefore, we assume an industry-like environment: the recommender system will be retrained regularly and will be exposed to clients for a relatively short period, ranging from a couple of days to a maximum of a few months. We postulate that the performance of the recommender on the last portion of historically available data is most indicative of how it will behave when deployed.

Our protocols focus on set creation. When selecting the target values in a validation set, we take two possible approaches. The first, *proportional selection*, depicted in Figure 1, selects the final $X\%$ of each user's interactions and uses these to create target items. Here we preserve the time ordering of the input and target interactions, maintaining similarity with the real-life use-case. However, there is no strict time cutoff, as is the case when we train a system on data available to a certain point and then deploy it.

The second approach, shown in Figure 2, is precisely based on a *strict time cutoff* to select the target items of the validation set. This method is even closer to the real-world use case. However, it does suffer from certain drawbacks as user interactions are not necessarily evenly distributed through time, leading to some users being more represented than others in the target set. While these are similar to the suggestions developed in (Campos, Díez, and Cantador 2014), we underline that these approaches should not be limited to evaluating TARS. It is crucial to approximate with maximum precision the performance of a model when developing a novel system, before it is released into production. The second approach directly models the real-world context and contains user-item interaction sequences created after a specific strict time cutoff.

4 Recency to Improve Recommendation

The main task of a recommender system is to anticipate users' future desires and suggest content that they would find relevant. However, just recommending the most relevant items does not always satisfy all the concerns of those building the system. The *relevance* objective is the one that is most commonly found in recommender systems literature and accounts for the accuracy or correctness. It actively focuses the recommender on selecting the item(s) with which the users will most likely interact.

However, relevance is not the only objective used in practice. We distinguish two types of objectives: *correlated* and *uncorrelated* to relevance. The former ones correspond to those whose optimization is linked to the relevance objective. Examples are novelty (Vargas and Castells 2011), serendipity (Ge, Delgado-Battenfeld, and Jannach 2010), and utility-based objectives, such as revenue. The latter, *not correlated to relevance*, can be diversity and fairness.

We introduce such a utility-based objective used to inject temporal information alongside the relevance objective. While the exploration of uncorrelated objectives is essential for the future of recommender systems, we leave it for future work.

4.1 Adding Temporal Context

Based on the our experience with real-life use-cases, we discovered that users seem to gravitate towards purchasing content that had more recently been added to a given platform. Building on these findings, and works such as (Chakraborty et al. 2017) and (Gabriel De Souza, Jannach, and Da Cunha 2019), we decided to explore the effects of incorporating recency as an objective during the learning phase. Given an item x with a timestamp t_x, we further define the recency function f as:

$$f(x) = \begin{cases} 1 & \frac{t_x - t_{min}}{t_{max} - t_{min}} \geq 0.8 \\ 0.3^{(0.8 - \frac{t_x - t_{min}}{t_{max} - t_{min}}) \times \frac{10}{3}} & otherwise \end{cases} \quad (1)$$

where t_{max} and t_{min} are the maximum (most recent) and minimum (oldest) timestamps over the itemset. In f, we first scale all timestamps to $[0, 1]$ using the min-max scaler, and then apply a transformation inspired by (Huang et al. 2013). A plot of the function is shown in Figure 3.

The recency objective is formulated as a loss that stimulates the recommendation of recent items. Each item in the itemset is assigned a recency weight, based on the recency function. The vector is then used to weigh item importance

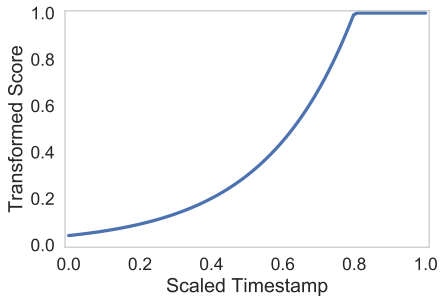


Figure 3: Our proposed recency function (Equation 1).

Algorithm 1 SMSGDA with Gradient Normalization.

1: $initialize()$
2: **for** $i \in 1, ..., n$ **do**
3: $\quad empirical\_loss_i = \mathcal{L}_i(w)$
4: **end for**
5: **for** $epoch \in 1, ..., M$ **do**
6: $\quad$ **for** $batch \in 1, ..., B$ **do**
7: $\quad\quad do\_forward\_pass()$
8: $\quad\quad$ **for** $i \in 1, ..., n$ **do**
9: $\quad\quad\quad calculate\, loss\;\; \mathcal{L}_i(w)$
10: $\quad\quad\quad calculate\, gradient\;\; \nabla_w \mathcal{L}_i(w)$
11: $\quad\quad\quad normalize\, gradient\;\; \nabla_w \hat{\mathcal{L}}_i(w) = \frac{\nabla_w \mathcal{L}_i(w)}{empirical\_loss_i}$
12: $\quad\quad$ **end for**
13: $\quad\quad \alpha_1, \ldots, \alpha_n = QCOPSolver\left(\nabla_w \hat{\mathcal{L}}_1(w), \ldots, \nabla_w \hat{\mathcal{L}}_n(w)\right)$
14: $\quad\quad \nabla_w \mathcal{L}(w) = \sum_{i=1}^{n} \alpha_i \nabla_w \hat{\mathcal{L}}_i(w)$
15: $\quad\quad w = w - \eta \nabla_w \mathcal{L}(w)$
16: $\quad$ **end for**
17: $\quad evaluate\_model()$
18: $\quad update\_pareto\_set()$
19: **end for**

when calculating the loss. Adding weights into a traditional loss does not affect the differentiability of the function.

To illustrate how our temporal objective can be easily integrated into a traditionally time-unaware recommender system, we take as a use-case the state-of-the-art variational autoencoder Mult-VAE^{PR} of (Liang et al. 2018). For the sake of brevity, we refer the reader to (Liang et al. 2018) for more details about the model.

We thus propose an extension of Mult-VAE^{PR}, where the loss function for user u is modified to:

$$\mathcal{L}_\beta(x_u; \theta, \phi) = \mathbb{E}_{q_\phi(z_u|x_u)}\left[\log p_\theta(f(x_u) * x_u | z_u)\right] - \beta \cdot \mathrm{KL}(q_\phi(z_u|x_u)||p(z_u))$$

where the expected negative log-likelihood is modified to include the element-wise multiplication of input vector x_u by $f(x_u)$, which corresponds to the recency scores of the given items in x_u. β controls how much importance is given to the KL term, z_u is a variational parameter of the variational distribution θ and ϕ are model parameters.

4.2 Multi-Objective Optimization

Optimizing a recommender on multiple objectives is non-trivial. Thanks to the recent work of (Milojkovic et al. 2019), we employ the proposed multi-gradient descent algorithm for multiple objectives to train our recommenders. Additionally, the authors show that the algorithm is efficient and does not impact on training time, as it can be seen in Algorithm 1, where n is the number of objectives, M is the number of epochs, and B the number of batches. After a standard forward pass (Line 7), the loss and gradient are computed for each objective (Line 8-10). Then, weights of the gradients are computing as a Quadratic Constrained Optimization Problem (Désidéri 2012), which can be solved analytically for two objectives, or solved as a constrained optimization problem as proposed in (Sener and Koltun 2018) for more than two objectives. Solving it allows us to obtain the common descent vector and update the parameters (Line 14-15).

This training procedure enables us to incorporate both our temporal context and the relevance objectives to retrieve time-aware recommendations. The algorithm adapts the weight repartition between the two objectives in an advanced manner to optimize both during training.

	ML-20M	**Steam**	**Netflix≥4**
# of users	46,295	257,775	471,457
# of items	9479	13,018	13,995
# of interactions	3.76M	3.14M	38.87M
% of interactions	0.86%	0.09%	0.59%

Table 1: Statistics of the datasets, after preprocessing.

5 Experiments

5.1 Datasets

We study the performance of various models on three real-world publicly available datasets. The characteristics of the preprocessed datasets are summarized in Table 1.

MovieLens-20M contains about 20 million ratings[1], with values between 1 and 5. To transform it into implicit feedback, we binarize the user-item interaction matrix, keeping ratings of 4 and above as positive feedback. We filter out all users with less than five ratings, and all movies rated by less than five users. Since this dataset contains entries from 1999 up to 2015, we chose to focus on the last ten years of available data.

Steam has review information from the gaming platform Steam[2]. We converted user-item interactions into a positive feedback signals. The dataset contains reviews from 2010 to 2018; however, the platform only sees an uptick in review activity after 2014, which is why we select the last four years available for further analysis.

Netflix is the well-known Netflix Prize Competition dataset[3]. It consists of over 100 million ratings. The ratings are on a scale from 1 to 5 and were collected between 1998 and 2005. We filter these ratings in the same way as the MovieLens ratings, and take the last two years of activity. Because of low performance on certain baselines, we denote two variants for the implicit feedback: one with threshold

[1] https://grouplens.org/datasets/movielens/20m/.
[2] https://cseweb.ucsd.edu/~jmcauley/datasets.html.
[3] https://www.kaggle.com/netflix-inc/netflix-prize-data.

of 4 and above (Netflix≥4), the other one with a threshold of 5 (Netflix≥5).

5.2 Recommendation Techniques

For all models, we ensured that the items that the user had previously interacted with were removed from the output before the top-k results were selected for metric calculation. All models were trained with the Adam optimizer, with a learning rate of 0.001.

Mult-VAE^{PR}: All experiments with the Mult-VAE^{PR} (Liang et al. 2018) were conducted using the implementation from the MAMO framework[4]. We used the same setup as in the original paper.

SVD: We utilize the PyTorch implementation[5] of the Singular Value Decomposition (Sarwar et al. 2000), taking only the top 100 dimensions.

NCF: For Neural Collaborative Filtering (He et al. 2017), we take the implementation from [6], sample four negative instances for every existing user-item interaction, set the predictive factor of 64, and the number of hidden layers for the multilayer perceptron (MLP) to three. We do not present results obtained using pre-trained NeuMF, as they exhibited the same patterns as generalized matrix factorizaion (GMF) and MLP, but did not give a significant improvement.

To resolve the difficulties to obtain good results with the *Netflix≥4* dataset for GMF and MLP models, we used instead the *Netflix≥5* dataset. The main difference being that only ratings of five and above are considered as positive.

BERT4Rec: This sequence-aware recommender system was introduced in (Sun et al. 2019). We implemented it in PyTorch and integrated it with the MAMO framework. Most of the hyperparameters used were taken from the original paper. The number of transformer layers is set to 2, the head number is 4, head dimensionality is 64, and the dropout is 0.1. We use a sequence length of 100, while the proportion of masked inputs is 0.2. The model is trained using the Adam optimizer with a learning rate of 1e-4.

BERT4Rec Extension: We propose an extension to BERT4Rec. BERT4Rec consumes sequences of items. The positions in the sequence that the user wishes to predict are filled with a special mask identifier. When predicting the next item(s) in a sequence, (Sun et al. 2019) place a mask on the last position in the sequence, and then take the top-k items from the probability distribution of this position, as generated by the model. Instead of selecting the top-k items from the last (masked) position in the sequence, we suggest to select them from the last-p positions, all of which are masked in the input sequence. From each position in p we select the top-$\lfloor \frac{k}{p} \rfloor$ items, making sure that there are no repeated items. If k is not divisible by p, the leftover elements are selected from the first masked position.

5.3 Experimental Setup

The experiments conducted show how an inadequate manner of creating the validation sets in the *deployment-ready* phase leads to false confidence in the performance of the evaluated model. In the *deployment-ready* phase, what we call the validation set is not necessarily used for hyperparameter tuning, but to assess the performance of the model before it is deployed. There are minor differences in the datasets used for the models with and without user representation. Models without user representation require some input interactions to be able to predict targets, while those without simply need to be passed a user identifier.

We divide our experiments into three sets, corresponding to the type of evaluation.

Traditional Evaluation. Similarly to (Liang et al. 2018), we divide the data into a train set with 80% of users, validation set with 10% and test set with 10%. The target user-item interactions are selected by randomly sampling 20% of the user-item interactions in the validation and test sets.

We show that if a model is evaluated on and later applied to a task that entails predicting randomly held-out interactions, the performance achieved on both validation and test sets is comparable. This traditional approach is typically used to report model performance.

We then contrast performance on randomly held-out interactions in the validation set against temporally held-out interactions in the test set. We take 5% of the users from the train set to create the validation set and hold-out 20% of their interactions. The test set contains the interactions and users from the train and validation sets as inputs, and the temporally held-out interactions are targets.

Temporal Evaluation. We show that when evaluated with either a proportional or hard temporal cutoff, the model's performance is closer to what would be observed in a real-life setting. However, it is important to note the ideal evaluation technique is heavily domain dependent.

First, we hold out the last 20% of user-item interactions from each user in the validation set. In the second approach, we hold out the last couple of months of activity and evaluate the model's ability to predict these interactions. We create the validation and test sets as before.

Temporal Evaluation with Added Temporal Context. We introduce temporal context into the traditionally time-independent Mult-VAE^{PR} by using the work from (Milojkovic et al. 2019) to optimize the model for accuracy and recency. To calculate the recency score we must determine a timestamp for every item in the itemset. We take the timestamp of the moment that the item first became available, or the first recorded instance of any user interacting with the given item. The strict temporal cutoff validation set is utilized, as well as the temporal test set described previously.

5.4 Evaluation Metrics

We evaluate models using three ranking metrics, as recommenders can often only show a predefined number of recommendations.

- **Precision@K:** calculates how many of the recommended items are relevant to the user;
- **Recall@K:** quantifies the proportion of relevant items in the top-k recommended items by calculating how many of

[4]https://github.com/swisscom/ai-research-mamo-framework.

[5]https://pytorch.org/docs/stable/generated/torch.svd.html.

[6]https://github.com/guoyang9/NCF.

Dataset	Model	Val^{trad}	Val^{prop}	Val^{cutoff}	Test^{temp}
ML-20M	Mult-VAE^{PR}	0.32 / 0.18	0.26 / 0.13	0.11 / 0.06	0.11 / 0.07
	SVD	0.25 / 0.22	0.14 / 0.11	0.07 / 0.03	0.11 / 0.07
	GMF	0.25 / 0.22	0.11/ 0.10	0.08 / 0.03	0.10 / 0.07
	MLP	0.25 / 0.23	0.12 / 0.10	0.07 / 0.03	0.11 / 0.07
Steam	Mult-VAE^{PR}	0.20 / 0.02	0.14 / 0.02	0.11 / 0.01	0.13 / 0.01
	SVD	0.10 / 0.02	0.10 / 0.02	0.09 / 0.01	0.08 / 0.01
Netflix$\geq$4	Mult-VAE^{PR}	0.35 / 0.18	0.22 / 0.10	0.12 / 0.05	0.10 / 0.05
	SVD	0.23 / 0.16	0.23 / 0.16	0.09 / 0.05	0.07 / 0.04
Netflix$\geq$5	SVD	0.23 / 0.10	0.23 / 0.11	0.12 / 0.05	0.09 / 0.03
	GMF	0.31 / 0.14	0.30 / 0.14	0.14 / 0.05	0.12 / 0.04
	MLP	0.31 / 0.14	0.30 / 0.14	0.14 / 0.05	0.12 / 0.04

Table 2: Results of the Mult-VAE^{PR}, SVD, GMF, and MLP evaluated on a traditional, proportionally selected temporal, and strict cutoff validation set, as well as on a temporally shifted test set. We report Recall / Precision at $k = 20$.

Dataset	Val^{trad}	Test^{trad}
ML-20M	0.31 / 0.17	0.31 / 0.17
Steam	0.20 / 0.02	0.20 / 0.02
Netflix$\geq$4	0.35 / 0.19	0.35 / 0.19

Table 3: Results of initial Mult-VAE^{PR} experiments, evaluated on a traditional evaluation protocol. We report Recall / Precision at $k = 20$.

the desirable items are are suggested to the end-user. We take our definition from (Liang et al. 2018);

- **Recency@K:** assigns a recency score to each item, calculating the rating of the top-k recommended and relevant items. For user u with relevant items I_u we define $\omega(k)$ as the item at rank k, where $\mathbb{I}$ is the indicator function:

$$Recency@K(u, \omega, f) = \sum_{k=1}^{K} \mathbb{I}[\omega(k) \in I_u] \times f(\omega(k)) \tag{2}$$

6 Results

Traditional Evaluation. This experiment aims to show that the traditional way of evaluation recommender systems, shown in Table 3, is not a faithful representation of the environments in which they are actually deployed. The good performance achieved by evaluating in this way can provide a false sense of security.

Our claim is supported by the values highlighted by Table 2. Even though the validation sets are not identical to the ones before, the performance observed is very similar. However, it degrades on the time delayed test set, or to be more precise, when it simulates what would happen in a production setting. Drops in performance of -65.63%, -35.00%, and -71.43% can be observed, on the Recall@20 values.

We postulate that this discrepancy leads to significant dissonance between the results of certain recommenders as reported in literature, and those observed in their real-life application.

Temporal Evaluation. The results shown in Table 2 depict what happens when using traditional validation as apposed to our proposed evaluation sets. The table illustrates how the strict cutoff validation set approximates the *deployment* behavior. For all datasets, this approach seems to be a closer estimation of the "real-life" performance. For example, the drop in performance is reduced from -71.43% to -16.67% on the Netflix$\geq$4 dataset. The proportionally selected validation sets seems to work well for the Steam dataset, and we know from industry experience that it can be good on others. However, it seems to be highly dataset specific and is something that should be kept in mind.

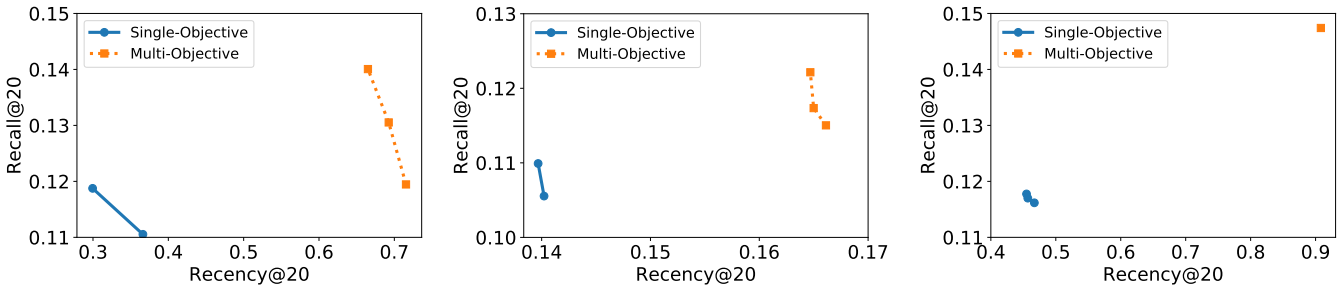
Table 2 also shows that this phenomenon is not isolated to the Mult-VAE^{PR}, but can be repeated with the SVD, GMF, and MLP models. As mentioned before, we were unable to conduct experiments on Netflix$\geq$4 with the GMF and MLP models; therefore we report their results on Netflix$\geq$5.

The most severe drop in performance is in the case of traditional evaluation on the GMF and MLP on the Netflix$\geq$5 dataset, where the Recall@20 decreases by -61.29%.

It is important to note that simpler methods, especially those based on matrix factorization, do not deal well with the Steam dataset. This is the sparsest dataset that we work with, as shown in Table 1, and this seems to make it difficult to learn anything meaningful. Following this conclusion, we exclude the Steam dataset results for GMF and MLP. However, we keep the results for SVD.

We strongly recommend that these evaluation methods be taken into account when presenting novel achievements in the field. When feasible, we recommend to apply both evaluation protocols.

Temporal Evaluation and Temporal Models. The results presented so far were achieved using traditional recommender architectures, with no way of learning temporal dynamics. Our next contribution is to integrate the temporal dynamic in the training process. Table 4 shows the results obtained with BERT4Rec and BERT4Rec$^{(5)}$. We apply this extension in the testing phase only. The results show that while it boosts the predictive power of the model in some



(a) ML20m dataset. (b) Steam dataset. (c) Netflix≥4 dataset.

Figure 4: Pareto Fronts obtained through optimizing on one objective (accuracy), and two objectives (accuracy and recency).

Dataset	Model	Val$^{\text{cutoff}}$	Test$^{\text{temp}}$
ML-20M	BERT4Rec	0.20 / 0.09	0.15 / 0.08
	BERT4Rec$^{(5)}$	-	0.15 / 0.08
Steam	BERT4Rec	0.21 / 0.02	0.17 / 0.02
	BERT4Rec$^{(5)}$	-	0.18 / 0.02
Netflix≥4	BERT4Rec	0.24 / 0.13	0.20 / 0.05
	BERT4Rec$^{(5)}$	-	0.21 / 0.06

Table 4: Results of BERT4Rec and BERT4Rec$^{(5)}$ evaluated on a strict cutoff validation set and a time delayed test set. BERT4Rec$^{(5)}$ is only applied on the test set. We report Recall / Precision at $k = 20$.

Dataset	Model	R	P	Re
ML-20M	Mult-VAE$^{\text{PR}}$	0.11	0.07	0.23
	MOREVAE	**0.13**	**0.08**	**0.47**
Steam	Mult-VAE$^{\text{PR}}$	**0.13**	**0.01**	0.15
	MOREVAE	**0.13**	**0.01**	**0.18**
Netflix≥4	Mult-VAE$^{\text{PR}}$	0.10	0.04	0.34
	MOREVAE	**0.12**	**0.05**	**0.66**

Table 5: Comparison of Mult-VAE$^{\text{PR}}$ and MOREVAE results obtained on temporally shifted test sets. We report **R**ecall, **P**recision, and **Re**cency at $k = 20$.

cases, the benefits vary from case to case.

By comparing Table 4 and Table 2 (that contain results achieved with traditional, time-independent recommenders), BERT4Rec achieves the best performance on the test set. This confirms our hypothesis that temporal dynamics should be accounted for in both evaluation design and model architecture in order to attain the best possible recommenders.

Temporal Evaluation with Added Temporal Context. To further integrate the temporal context, our following contribution has the recency included as an objective influencing the optimization. We refer to the multi-objective Mult-VAE$^{\text{PR}}$ as the Multi-Objective Recency Enriched mult-VAE$^{\text{PR}}$(MOREVAE).

We present both the Pareto Fronts obtained during training and the results of the best models on the test sets. Those results were obtained through more intense training than those shown in the previous sections. The Pareto Fronts were generated by evaluating on the strict cutoff validation sets during training, and the best models were chosen by selecting those with the highest Recall@20 and applying them to the time delayed test sets. Figure 4 shows that the multi-objective approach not only dominates the single objective one in terms of recency, but that optimizing for recency also increases the relevance of the recommendations, validating our initial intuition. The results of the best models over the test sets are shown in Table 5. The improvements obtained are 18.18%, 0.00%, and 20% for Recall@20; 14.29%, 0.00%, and 25.00% for Precision@20. The improvements seen in Recency@20 are 104.35%, 20.00%, and 94.12%.

7 Conclusion

Following standard offline recommendation evaluations during development, based on random sampling of user-item interactions as held-out data, leads to false confidence when deploying models in real-life scenarios. Previous research generally focused on developing better metrics to reflect real-world performance, but still omitted temporal context. We highlighted the lack of standardization and proposed two temporal evaluation protocols that empirically better approximate real-life conditions.

Our second contribution is a novel recency objective, that can be used to integrate temporal information in existing time-unaware recommenders. We propose to leverage a multi-objective approach and train models on relevance and recency simultaneously. Experiments on three real-world publicly available datasets showed that our method produced solutions that strictly dominate those obtained with a model trained on a single-objective optimization.

We explored datasets that are frequently used in recommender systems research, all related to digital media content. Digital media content is consumed frequently and generally without much repetition. The importance of recency and capturing transient behavioral trends may not be equivalent in other recommender systems applications, such as grocery or clothes shopping. The influence of temporal dynamics on these sectors is an exciting topic, and we leave it to future academic and commercial research.

References

Aggarwal, C. C.; et al. 2016. *Recommender systems*, volume 1. Springer.

Breese, J. S.; Heckerman, D.; and Kadie, C. 1998. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, 43–52. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 155860555X.

Campos, P. G.; Díez, F.; and Cantador, I. 2014. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction* 24(1-2): 67–119.

Chakraborty, A.; Ghosh, S.; Ganguly, N.; and Gummadi, K. P. 2017. Optimizing the recency-relevancy trade-off in online news recommendations. In *Proceedings of the 26th International Conference on World Wide Web*, 837–846.

Désidéri, J.-A. 2012. Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. *Comptes Rendus Mathematique* 350(5-6): 313–318.

Ding, Y.; Li, X.; and Orlowska, M. E. 2006. Recency-based collaborative filtering. In *Proceedings of the 17th Australasian Database Conference-Volume 49*, 99–107.

Gabriel De Souza, P. M.; Jannach, D.; and Da Cunha, A. M. 2019. Contextual hybrid session-based news recommendation with recurrent neural networks. *IEEE Access* 7: 169185–169203.

Ge, M.; Delgado-Battenfeld, C.; and Jannach, D. 2010. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, 257–260.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, 173–182.

Hidasi, B.; and Karatzoglou, A. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 843–852.

Huang, C.-L.; Chen, M.-C.; Huang, W.-C.; and Huang, S.-H. 2013. Incorporating frequency, recency and profit in sequential pattern based recommender systems. *Intelligent Data Analysis* 17(5): 899–916.

Kang, W.-C.; and McAuley, J. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, 197–206. IEEE.

Liang, D.; Krishnan, R. G.; Hoffman, M. D.; and Jebara, T. 2018. Variational Autoencoders for Collaborative Filtering.

Luo, X.; Xia, Y.; and Zhu, Q. 2012. Incremental collaborative filtering recommender based on regularized matrix factorization. *Knowledge-Based Systems* 27: 271–280.

Marlin, B. 2004. *Collaborative filtering: A machine learning perspective*. University of Toronto Toronto.

Milojkovic, N.; Antognini, D.; Bergamin, G.; Faltings, B.; and Musat, C. 2019. Multi-Gradient Descent for Multi-Objective Recommender Systems. *arXiv preprint arXiv:2001.00846* .

Ning, X.; and Karypis, G. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, 497–506. IEEE.

Quadrana, M.; Cremonesi, P.; and Jannach, D. 2018. Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)* 51(4): 1–36.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* .

Sarwar, B.; Karypis, G.; Konstan, J.; and Riedl, J. 2000. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science.

Sener, O.; and Koltun, V. 2018. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, 527–538.

Sun, F.; Liu, J.; Wu, J.; Pei, C.; Lin, X.; Ou, W.; and Jiang, P. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 1441–1450.

Vargas, S.; and Castells, P. 2011. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, 109–116.

Vinagre, J.; Jorge, A. M.; and Gama, J. 2015. Collaborative filtering with recency-based negative feedback. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 963–965.

Wu, Y.; DuBois, C.; Zheng, A. X.; and Ester, M. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, 153–162.