Scheduling on a Budget: Avoiding Stale Recommendations with Timely Updates

Robin Verachtert[a,b,*], Olivier Jeunen[b,1], Bart Goethals[b,a,c]

[a] *Froomle NV, Posthofbrug 6-8, 2600 Antwerpen, Belgium*
[b] *University of Antwerp, Prinsstraat 13, 2000 Antwerpen, Belgium*
[c] *Monash University, Melbourne, VIC, Australia*

Abstract

Recommendation systems usually create static models from historical data. Due to concept drift and changes in the environment, such models are doomed to become stale, which causes their performance to degrade. In live production environments, models are therefore typically retrained at fixed time-intervals. Of course, every retraining comes at a significant computational cost, making very frequent model updates unrealistic in practice. In some cases, the cost is worth it, but in other cases an update could be redundant and the cost an unnecessary loss. The research question then consists of finding an acceptable update schedule for your recommendation system, given a limited budget. This work provides a pragmatic analysis of model staleness for a variety of collaborative filtering algorithms in news and retail domains, where concept drift is a known impediment. We highlight that the rate at which models become stale is highly dependent on the environment they perform in and that this property can be derived from data. These findings are corroborated by empirical observations from four large-scale online experiments. Instead of retraining at regular intervals, we propose an adaptive scheduling method that aims to maximise the accuracy of the recommendations within a fixed resource budget. Offline experiments show that our proposed approach improves recommendation performance

*Corresponding author
Email addresses: robin.verachtert@froomle.com (Robin Verachtert), jeunen@sharechat.co (Olivier Jeunen), bart.goethals@uantwerpen.be (Bart Goethals)
[1] Present address: ShareChat, New Bailey Street, Salford, M3 5FS, United Kingdom

while keeping the cost constant. Our findings can guide practitioners to spend their available resources more efficiently.

---

1. Introduction

Recommendation systems are deployed in production environments where they help users find relevant items in typically large catalogues. In modern-day settings, these users generate millions of interactions every day. This continuous stream of information creates two challenges for recommender systems: 1. the amount of data they need to process calls for efficient algorithms that can keep up with these ever-growing data streams, and 2. the dynamic nature of this data calls for frequent model updates (Gama et al., 2014; Al-Ghossein et al., 2018). As time passes, new items become available, others disappear, interests of the global population change, seasons make different items relevant, and interests of single users also change over time. These changes are crucial for the recommendation system to take into account.

Much research has concentrated on developing algorithms to better leverage large amounts of data and achieve greater accuracy. Additional improvements also account for concept drift in the data (Campos et al., 2014; Koren et al., 2009).

Recommendation methods are usually evaluated by splitting the dataset into a static training-, validation-, and test-set (Jeunen, 2019; Cañamares et al., 2020). While this is effective for evaluating the quality of the models, it does not address the dynamic nature of the data, due to which even the most accurate algorithms inevitably become stale.

Understanding how model staleness affects recommendation systems for specific environments is necessary to determine how frequently models should be updated (Breck et al., 2017). Therefore, we first study how recommender systems are impacted by model staleness as a result of concept drift in the environment. We study both the change that models undergo as new data becomes available and the resulting performance degradation. Our experiments in Fig-

ure 1 show that model staleness impacts quality very quickly in news domains, due to rapidly changing content and user interests. After just a few hours, the models will recommend mostly irrelevant items. In traditional retail settings, drift is much less pronounced, and as a result, models remain useful for longer periods of time.

The straightforward solution to avoid model staleness is to retrain the model. To the best of our knowledge, the natural question "When should the model be retrained?", remains unanswered in the scientific literature. In this paper, we attempt to formulate an answer to this often overlooked, yet important question.

A naïve answer is to continuously update the computed models. Once a model is trained and deployed, a new training cycle begins that recomputes a model on the updated dataset. In practice however, such a continuous training cycle incurs a significant cost of computational resources. Therefore, the go-to approach is to schedule model updates at regular intervals.

A more elegant solution is to create *incremental* models (Vinagre et al., 2014; Anyosa et al., 2018; Al-Ghossein et al., 2018; Jeunen et al., 2019; Vinagre et al., 2020; Jeunen et al., 2022). Instead of recomputing the model on the *entire* dataset, these models only process newly collected data and update a previously computed model with this new information. These methods have the advantage that the computational cost for each update is typically much lower than when the model is retrained on the full, potentially huge dataset. Still, the same research question holds, since even for incremental methods, a choice must be made on *when* each update is scheduled. This can be continuous, immediately as new data arrives, or more typically as the cost of computational resources can be high, in batches after a certain amount of time.

We challenge the implicit assumption made in regular interval scheduling that each update results in a similar increase in accuracy. To illustrate this, imagine a local news website. At night, much fewer interactions occur and usually no new articles become available. As a result, we can safely state that any updates scheduled during the night add very little to the quality of the model. Yet, they are roughly equal in cost to model updates that are being

computed at peak traffic hours (e.g. lunchtime) when drifting user interests and new items significantly impact the model. A clear need arises for a scheduling procedure that allows models to be trained at more opportune times. Such a scheduling procedure would increase the online performance of the model over the entire time period, whilst keeping computational costs constant. The day-and-night example serves to illustrate this phenomenon, but it should be clear that a scheduling algorithm that learns from the data is superior.

We posit that the goal of a scheduling procedure is to ensure that each update captures the same amount of new *information*. As such, updates should be scheduled more frequently in periods of high *information gain*, and less frequently in periods of lower gain. All this should happen whilst keeping a fixed budget into account. As we do not know the distribution of high-information periods over the next scheduling period, this adds a non-trivial complexity to the problem.

To overcome these challenges, we propose novel methods for scheduling updates based on summary statistics from previously observed batches of data. Our methods aim to detect *when* model updates benefit the system the most, considering either the informational value of the data or changes in the projected output of the model. We show that these methods improve on the widely used approach with regular intervals. Furthermore, our methods provide a solution to create an optimized schedule for a given budget, which is a common concern for practitioners. So far we intentionally did not mention the specific recommendation method used, as we wish to propose a method independent of the specific algorithm used. After all, even complex systems often rely on basic building blocks that need retraining.

The remainder of this paper is organized as follows: In Section 2, we highlight related work. In Section 3, we describe the methods used in the analysis of model staleness in a recommendation context, and describe the scheduling methods. In Section 4, the experimental results are presented and analysed. Finally, in Section 5, we present how our work provides valuable insights and guidance for practitioners that need to balance the trade-off between keeping a model up to

date and the computational cost that comes with it.

2. Related Work

Detecting concept drift. This is a well studied problem in a variety of applications (Bifet and Gavalda, 2007; Klinkenberg and Renz, 1998; Gama et al., 2014; Qahtan et al., 2015; Widmer and Kubat, 1996; Yu and Abraham, 2017). Its goal is to detect *when* the underlying distribution of incoming observed data samples changes. Typically, these methods either: 1. measure changes in performance, 2. inspect changing properties of the model, or 3. inspect changing properties of the data (Klinkenberg and Renz, 1998).

Concept drift detection can be used to decide *when* to update a model, by simply coupling the detection of change with the action of a model update. In an environment where change is rapid, online performance measures, such as click-through-rate or offline metrics, such as precision and recall, are not suitable for such purposes, because they typically show high variance, and detecting changes in such metrics in short intervals is prone to false positives. Depending on the sensitivity of the model it is likely to either wait too long (for the confidence estimate to be smoothed by more samples), or act too soon (as each sudden change in the target value due to variance is considered a detected change in the underlying data).

The alternative of model introspection can provide a way forward in specific use-cases, when the model can be efficiently inspected. However, a generally applicable solution should preferably be model-agnostic. Because of these reasons, in this work we focus on the properties of the data, as this seems to be the most interesting avenue to decide on model updates in general yet highly dynamic recommendation scenarios.

Increasing the computational efficiency of model training. In virtually all practical recommendation applications, models that train faster are favourable. The reason for this is two-fold: 1. New data can be incorporated more quickly to

combat concept drift, and 2. The cost of keeping the model up-to-date is reduced.

Different approaches to improve the computation time of models exist. A first class of methods suggests a trade-off between the accuracy of the model and the computation time. Approximate models explicitly trade the exactness of the model for improved computational complexity (Arya et al., 1998; Li et al., 2019; Steck, 2019c). Forgetting mechanisms reduce computational complexity by reducing the amount of data used for training (Vinagre and Jorge, 2012). A second way to reduce computational cost is to adopt models that can be updated incrementally, using only the latest batch of information collected, rather than the complete historical data set. Incremental methods either compute an exact model, which is interchangeable with the traditional non-incremental one, while requiring less time to update (Jeunen et al., 2019; Vinagre et al., 2014), or use approximate methods (Zhang et al., 2020; Jeunen et al., 2022). Although both of these approaches to reduce computational cost study the cost-accuracy trade-off, they still lack a method for scheduling the updates. For example, in the extreme case where an update of the model is triggered with every new transaction, the model would be continuously recomputed. This introduces the high cost of a constantly running resource. More typically a fixed schedule is used, such that models are updated at regular intervals. Alternatively, a fixed window size is used, such that the model will be updated after a fixed number of transactions. In this work, we challenge both the assumption that every update is equally valuable and the assumption that every transaction carries the same amount of information.

Model staleness. In practical machine learning, model staleness is a fundamental pitfall that must be avoided to achieve satisfactory results consistently over time (Breck et al., 2017). In recommendation systems this notion is to the best of our knowledge under-explored. Jambor et al. (2012) describe an application of systems control theory to the problem of updating the recommendation system. In their method, they make the assumption that each transaction carries

the same information value, which we challenge in this work. In addition, the method relies on performance metrics to monitor the system, which introduces large variance when looking at very fine grained control at the granularity of minutes.

Zanardi and Capra (2011) present an approach using feed-forward control system theory to decide on updating a model. Based on the number of new users and items that have arrived in the system, it decides when to retrain. This approach manages to avoid waiting for the model to degrade by predicting the degradation based on summary statistics. Contrary to the previous method, this method does not assume a linear relationship between transactions and degradation, instead they assume a relationship between new users and items arriving in the system. However, if known users behave differently, this update strategy will not suggest a model update, and similarly, if no new items arrive, the model will also not be retrained, while users' interests in items might be changing.

Al-Ghossein et al. (2018) use an adaptive windowing technique (ADWIN) to decide when to update a Latent Dirichlet Allocation (LDA) topic model in production (Bifet and Gavalda, 2007; Blei et al., 2003). Their approach monitors the probability that a word occurs in a text, given the inferred LDA model. When this probability shifts significantly, the older data is discarded, and the model is updated using only the more recent data.

Our work differs from these approaches in that it is independent of a specific modelling technique, handles changing behaviour of users, and additionally takes into account the budget available for scheduling model updates, since that is often the variable that is constrained in a real-world scenario.

3. Methodology

Throughout this work, we assume the user-item interaction data to be binary and positive only, as this encompasses the most commonly encountered use-cases in practice (Verstrepen et al., 2017). A dataset D consists of user-item-timestamp triplets $(u, i, t) \in U \times I \times \mathbb{N}$. We assume the timestamp to be at

the granularity of seconds. User-item interactions can make up various different types of events (e.g. view, add-to-cart, click, purchase, ...), we will refer to them as "events" in general. A recommendation model ϕ maps a user representation to a vector of recommendation scores: $\phi : U \to \mathbb{R}^{|I|}$. For a user-item matrix $\boldsymbol{X} \in \mathbb{R}^{|U| \times |I|}$, $\phi(\boldsymbol{X})$ indicates a row-wise transformation from user histories to recommendation scores. For simplicity and without the loss of generalit, we do not consider more involved models that take into account contextual features or the sequence of the user history.

3.1. Measuring Model Staleness

We formalise the concept of "staleness" in two ways. First, a model is considered stale if its prediction accuracy is worse than a more recently updated model. Second, a model can be considered stale if its output is significantly different from that of a more recent model.

3.1.1. Accuracy

To measure changes in model accuracy over time, we evaluate models for consecutive, non-overlapping windows of w seconds on the test data. We will refer to these batches of test data as *slices*. As is typical in temporal evaluation, we split the data on timestamps t_{val} and t_0. The user-item interactions with timestamp $t < t_{\text{val}}$ are used for model training, those with $t_{\text{val}} \leq t < t_0$ for validation, and the remaining interactions with $t \geq t_0$ make up the test set.

We further divide the test data into n slices. This allows us to gain a more fine-grained view of model performance in the test set, since we can now obtain n performance estimates where $t_i = t_0 + (i \cdot \text{w})$ and slice i (0-indexed) consists of data $t_i \leq t < t_{i+1}$. Now, we evaluate the out-of-date model ϕ_0 trained on data up to t_0 and the up-to-date models ϕ_i trained on data up to t_i, and observe the impact of model recency on accuracy. This evaluation method boils down to the SW-EVAL method presented by Jeunen et al. (2018), where we additionally evaluate stale models on every test slice.

The decrease in accuracy as the model grows older is a manifestation of model staleness. For some datasets, the slope of this degradation is steep: a

couple of hours after training, the model achieves only 50% of the accuracy of an up-to-date model. For others, this degradation is far slower: after several days, the model still achieves 90% of the accuracy of an up-to-date model.

3.1.2. Output correlation

Two models might attain the same level of recommendation accuracy in a very different manner (i.e. being correct on non-overlapping sets of users and items). As such, one could argue in favour of focusing on the recommendations that these models generate. That is, we now define model change as differences in model *output*. To avoid biases present in real user histories, we choose to generate k psuedo-user histories using the following procedure. For each psuedo-user, we first generate a uniformly distributed random integer N between N_{min} and N_{max}, which defines the number of items in this user's history. Second, we generate this pseudo-user's history by sampling N items from the dataset without replacement. During sampling each item is given a uniform probability of getting chosen.

This produces a binary matrix $X \in \mathbb{N}^{k \times |I|}$, where $X_{ui} = 1$ when the item i was sampled for the pseudo-user u. For example, when the dataset contains 5 items and we generate $k = 3$ users with $N_{min} = 2$ and $N_{max} = 4$, we might get the following matrix.

$$X = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

This matrix X will be used as input to the recommendation models for which we want to evaluate staleness.

Let the updated model at slice s be ϕ_s. For each slice, we compute recommendation scores using an up-to-date model ϕ_s, and an outdated model ϕ_0. Let $P = \phi_s(X)$ and $Q = \phi_0(X)$. To calculate the change between models, we compute the correlation of the recommendation ranking implied by these two output matrices. We use the Kendall Tau b statistic (Kendall, 1945) to account for ties in scores. This uses the number of *concordant*, *discordant* and *tied* pairs

in the models' outputs. Intuitively, a high Kendall Tau value in a comparison between two model outputs indicates that the models agree on the ranking of most items, and therefore using them in the top-N recommendation scenarios would result in similar outcomes. A low Kendall Tau value indicates that the updated model disagrees with the ranking of the old model. So staleness occurs in ϕ_0, as using ϕ_s in top-N recommendation results in significantly different recommendations.

For each row i we consider all pairs of items (j, k) for which $j < k$ and items j and k have received a non-zero score in either P_i or Q_i.

3.2. Estimating information gain

The basis of our scheduling method is that we estimate the amount of information gained from events collected since the last update.

Number of events (EV). We assign an equal information value to each event collected since the previous update. So we compute the amount of new information at time t_i since the last update timestamp tl as

$$\text{information}_{\text{EV}} = \sum_{(u,i,t) \in D : t_l < t \leq t_i} 1 = |\{(u, i, t) \in D | t_l < t \leq t_i\}| \qquad (1)$$

Inverse predicted relevance (IPR). The assumption that each event contains the same information for the recommender system is often an oversimplification. Unexpected interactions are more valuable than expected ones because they indicate the inability of the model to predict a user's behaviour. To encode the informational value of an event to the model, we propose to use the inverted normalised recommendation score. High recommendation scores will give rise to lower information values and vice versa. Intuitively, if the recommendation model already *expected* the new user-item interaction, it holds less information than when it is *unexpected*. Formally, let $\phi_t : U \to \mathbb{R}^{|I|}$ be the model deployed at time t, and let $\phi_t(u, i) \equiv [\phi_t(u)]_i$ denote the i^{th} output (i.e. the recommendation score for item i). We additionally assume recommendation scores to be normalised to the unit interval. Then, the information value

collected at time t_i since the last update timestamp t_l is computed as:

$$\text{information}_{\text{IPR}} = \sum_{(u,i,t) \in D : t_l < t \leq t_i} \frac{1}{\phi_{t_l}(u,i)} \tag{2}$$

Output correlation (CORR). Rather than estimate the value of an event, we can also look at the effect the collected events had on a model. If a model computed with the new events would recommend significantly different items to similar users, we can surmise that a large amount of information has been gathered. Conversely if the output is closely correlated, we can assume that the new interactions did not carry much information. Let M_{l_l} be the model computed at the time of the last update (t_l), M_{t_i} the model computed at t_i and U' a fixed sample of users. We compute CORR = correlation$(M_l(U'), M_i(U'))$, as described in Section 3.1. Ideally we would like to compute the change of the model deployed in production, however this would require updating that model, which is exactly the action we are aiming to avoid doing unless necessary. Instead, we use a more cost-efficient but less accurate proxy-model that can be updated frequently without high cost. In our, experiments we use a "recently popular" model to compute correlations, but more sophisticated, efficiently computable, models can be used in exactly the same way.

3.3. Scheduling Model Updates

At each discrete timestamp t, our scheduling method needs to decide whether or not to schedule an update of the model. The scheduling method should most efficiently use an allowed budget to update the model such that performance over the whole period is maximized. For the first two methods EV and IPR, if the amount of collected information since the last update at timestamp t_l is above a threshold δ, an update will be scheduled. For CORR an update is scheduled once the model correlation falls below the threshold δ.

The "*optimal*" value for δ yields the highest model accuracy, within a specified update budget. As such, to find the optimal value for the threshold, our proposed methods should disregard any values that would introduce too frequent updates.

The allowed update frequency f_{update} depends on the available budget for model computations and the computational complexity of the model m used.

$$f_{\text{update}} = \frac{\text{budget}}{\Delta t \cdot \text{cost}(m)}.$$

Where $\frac{\text{budget}}{\Delta t}$ is the available budget for a certain period, and $\text{cost}(m)$ the monetary cost of training the model once. Evidently, models that require more resources, can be updated less frequently.

Using EV the value for δ_{EV} is the number of events before scheduling an update. Given a requested number of updates, this can be computed exactly. If f_{EV} is the average frequency of events (events / day) and f_{update} the requested update frequency (updates / day), then δ_{EV} is computed as:

$$\delta_{\text{EV}} = \frac{\text{information}_{EV}}{\text{update}} = \frac{f_{\text{EV}}}{f_{\text{update}}} \tag{3}$$

For the two other methods, we cannot compute an exact value, and so offline optimisation on a validation dataset is required. That is, for a range of values for δ, we calculate the number of updates that *would* have been scheduled by IPR or CORR. Then, we pick the maximal value of δ that remains within the prespecified budget.

4. Experimental Results

4.1. Datasets

In order to validate the proposed methods in a variety of domains, we use publicly available datasets for both news and retail use-cases, as well as two proprietary industrial datasets. For the news use-case, we use the Adressa (Gulla et al., 2017) and GLOBO (de Souza Pereira Moreira et al., 2018) datasets and the proprietary "*News*" dataset. For the retail use-case, we use the CosmeticsShop Kaggle dataset, as well as the proprietary "*Retail*" dataset. [2] [3]

As is common in the literature – we pruned users and items with very low numbers of interactions from the dataset (Beel and Brunel, 2019). In doing

[2] https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop
[3] https://rees46.com/

Dataset	|D|	|U|	|I|	Period
Adressa	2 526 497	226 795	2 643	7d
Globo.com	2 720 889	218 081	9 668	17d
Industry *News*	3 323 941	239 149	3 378	7d
CosmeticsShop	7 187 824	361 775	22 932	152d
Industry *Retail*	12 066 513	995 651	15 712	46d

Table 1: Properties for the datasets used in the offline experiments.

so, we significantly decrease the computational complexity of model training without significantly impacting the results obtained. As our experimental setup consists of an iterative model training and evaluation process, this benefits the reproducibility of our work with reasonable computational resources. We require users to have interacted with at least 3 items, and 10/50 interactions per item for the news/retail datasets respectively.

Table 1 shows summary statistics of the pre-processed datasets. The news datasets contain fewer items, and are collected over much shorter periods of time compared to the retail datasets. The train-test timestamp t_0 was kept constant per dataset for each experiment. For Adressa and the proprietary News dataset, we used the last two days of the data as test data. For Globo, we used the last five days. For CosmeticsShop we used the final month as test data. For the proprietary Retail dataset, we used the last two weeks. When evaluating algorithmic performance, we used sliding windows of 15 minutes for the news datasets, and 6 hours for the retail settings. This significantly reduced granularity for retail datasets is justified by our findings that models for retail take longer to change compared to news models.

4.2. Recommendation Algorithms

Table 2 lists the recommendation algorithms we considered for comparison in our offline experiments. Although we acknowledge that real-life industrial recommender systems are much more complex than any of these algorithms, we do believe that they are prototypical and can represent such systems for which the same principles of drift and scheduling apply. The cost per update is the estimated cost in USD for training the model once on a dedicated virtual

Dataset	Algorithm	Runtime (s)	Memory (GB)	Monthly Cost (USD)
Adressa	ItemKNN	35	1	0.47
	EASEr	44	1.2	0.60
	Mult-VAE	2120	3	695
	BPR-MF	7538	3	1180
Globo	ItemKNN	34	3	0.46
	EASEr	302	8	4.1
	Mult-VAE	3275	4	1074
	BPR-MF	7627	3	1080
CosmeticsShop	ItemKNN	94	11	2.6
	EASEr	3490	32	330
	Mult-VAE	15800	32	1346
	BPR-MF	-	> 60	-

Table 2: Computational requirements for public datasets and algorithms. Monthly cost assumes computing the model every hour. If training takes longer than one hour we make the assumption that training is only started once the previous model is done computing. For Mult-VAE we used the default parameters for hidden and bottleneck layer sizes (600 and 200). BPR-MF models were trained with embedding size = 100. Mult-VAE and BPR-MF were both run for 50 epochs. The BPR-MF computation for CosmeticsShop dataset ran out of memory on our testing machine with 60 GB RAM.

machine (VM) on Google Cloud Services [4]. The approaches that benefit from dedicated GPUs (BPR-MF (Rendle et al., 2009) and Mult-VAE (Liang et al., 2018)), were given a single NVIDIA Tesla P100. The computation time was computed on a machine with 14 virtual cores and 60 GB of RAM memory. As BPR-MF required more RAM than available on the CosmeticsShop dataset in our test setup, it is left blank in Table 2. For deep learning approaches, the introduction of a GPU increases costs dramatically, as well as their relatively higher run times. These costs are based on the training times of these models on the available data. In real world applications, the data available usually spans a longer period, more users and more items than we have available in the offline datasets. Because of this, we can expect practical estimates of the cost per update to be multiple times higher than the values reported here. Nevertheless, our method is unaffected by this scaling factor. By adjusting the cost estimate to the data at hand, we can still schedule updates on a given budget. Parameters used in the experiments were tuned using grid search on the validation set.

[4] https://cloud.google.com/products/calculator/

Because of the long training times for Mult-VAE and BPR-MF, our experiments that rely on iterative model retraining to simulate online behaviour would require several weeks of computation time. To improve reproducibility of the work whilst remaining relevant for state-of-the-art methods, we therefore focus on two scalable and highly competitive item-based collaborative filtering algorithms:

Item-kNN is a well-known and often used baseline algorithm for neighbourhood-based collaborative filtering (Sarwar et al., 2001). The model consists of a single matrix multiplication with an item-item matrix $\boldsymbol{S} \in \mathbb{R}^{|I|\times|I|}$: $\phi(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{S}$. Here, $\boldsymbol{S}_{i,j}$ holds the cosine similarity between items i and j, for which efficient algorithms exist (Jeunen et al., 2019). As $\boldsymbol{X}$ and $\boldsymbol{S}$ will often be sparse in nature, computing recommendation scores $\phi(\boldsymbol{X})$ is also fast. Recent work on neural news recommendation highlights the remarkable competitiveness of simple neighbourhood-based methods compared to more complex alternatives (Ludewig and Jannach, 2018; Moreira et al., 2019).

EASE$^{\text{r}}$ was recently proposed as an extension of the well-known SLIM method (Ning and Karypis, 2011; Steck, 2019b). In EASE$^{\text{r}}$, the item-item matrix $\boldsymbol{S}$ is found through a least-squares optimisation problem that allows for closed-form computation. This makes the model much more efficient to compute than various complex neural alternatives, whilst yielding highly competitive results. As the optimisation requires inverting the Gramian item-item matrix, EASE$^{\text{r}}$ becomes more costly to update as the size of the item catalogue grows. We refer the interested reader to the work of Steck (2019a) for further information on their approach.

We will release all source code to reproduce the results on the public data presented throughout the following sections upon acceptance.

4.3. Model Staleness

In this section we investigate why it is important to update recommendation models. First we show that models grow stale when they are not updated. In a second experiment, we show that (frequent) model updates mitigate model

staleness and are necessary to achieve adequate performance.

4.3.1. Existence of Model Staleness

We investigate the existence of model staleness by looking at how a models performance evolves as it gets older and how much the output changes between models trained at various points in the timeline. We follow the methodology defined in Section 3.1.

In Figure 1, we show how the accuracy of a model degrades over time for each of the datasets. To reduce the variance, and impact of time of day in evaluation, we compute the degradation for 12 different starting timestamps, each separated by one hour. As expected, models for news datasets suffer from staleness after only a couple of hours, while models in retail contexts remain similarly performant for a longer time. Interestingly, models on Globo and Adressa behave differently. On Adressa the degradation is much stronger than for Globo. The rate of model staleness is specific to the dataset, and thus it should be taken into account when choosing a training schedule in production.

To investigate the change in model output, we use $l_{\min} = 4$, $l_{\max} = 10$ such that the sampled histories contain between 4 and 10 interactions and $k = 5000$ samples for Adresssa, $k = 10000$ for Globo, and $k = 20000$ for CosmeticsShop.

The correlation results in Figure 2 extend the results of our analysis on accuracy. Based on their output, the models for all datasets grow stale. However, this change is not directly related to the chosen accuracy metrics. For example, a week-old ItemKNN model for CosmeticsShop still has an accuracy of 95% compared to the up-to-date model, despite their outputs only having a correlation of 0.7. Therefore, we remark that the model changes faster than it becomes stale, and this change could affect metrics such as diversity or fairness. This highlights that the number of required updates depends strongly on the chosen success metric as well. In this work, we limit the scope of our study to accuracy metrics.

Both decreases in accuracy and correlation of output between models confirm for recommendation models that, if we want high performance, we need to make

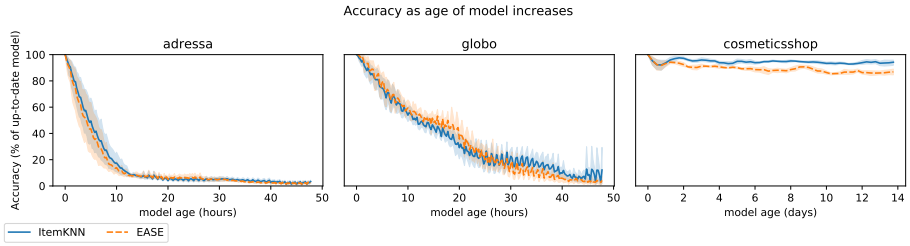

Figure 1: The y-axis shows the accuracy of the model as a percentage of the accuracy of the up to date model. Accuracy decreases much faster for news datasets, compared to the mostly stable performance on the retail dataset, both for EASE$^{\text{r}}$ (Dotted line) and ItemKNN (full line) models. The shaded area shows the 90% confidence interval.

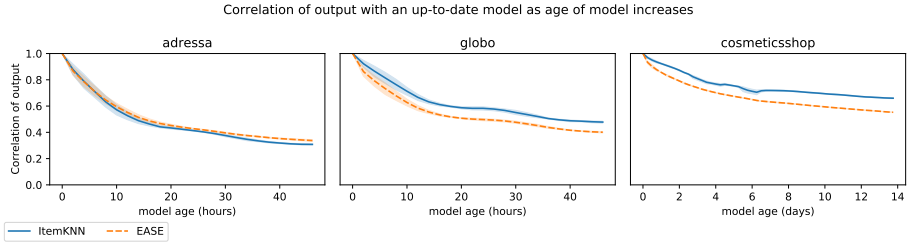

Figure 2: The y-axis shows the correlation between the output of an out-of-date model, and the up-to-date model. The x-axis shows the age of the out-of-date model. The shaded area shows the 90% confidence interval.

sure the model is up-to-date (Breck et al., 2017).

4.3.2. Impact of model staleness on model performance

We investigate how scheduling can mitigate the staleness effect, by evaluating how well a traditional regular interval scheduler mitigates the staleness of models. Specifically, we vary the update frequency to investigate the benefit gained from more frequent updates, as well as the costs associated with them.

Based on the results in Section 4.3.1, we expect more frequent updates to result in higher accuracy. Figure 3 reflects this expectation. For both news datasets, infrequent updates lead to lower performance than the more frequently updated models. However, we do notice a diminishing return on the number of updates, illustrated by the sections of the plots with zero gradient. For example, updating twice per hour (96 updates) to three times per hour (144 updates) is not significant. Updating from a single update every four hours, to two updates per hour does show a significant increase ($p < 0.05$) in performance.

For the retail dataset, CosmeticsShop, the stable performance of out-of-

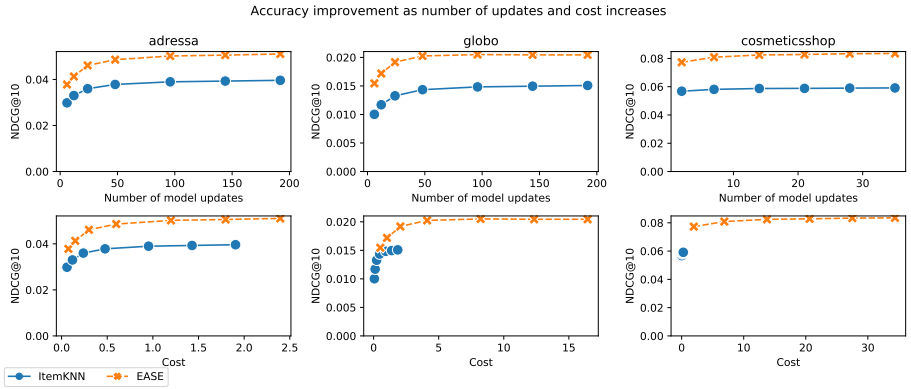

Figure 3: The y-axis shows NDCG@10 of the model, the x-axis shows the number of times the model was updated and the cost associated with these updates. Similar results are obtained with Recall@10, which we omit here for brevity. Costs for ItemKNN are far lower, as the model trains much faster, especially when the number of items grows. On cosmeticsshop the cost difference between ItemKNN and EASE is so big, that all ItemKNN point are around the same (low) cost.

date models shown in Figure 2 suggests that the benefit of increased updates is low. This expectation is confirmed here as well, as further increasing the number of updates past 14 for the period of 14 days yields only minimal accuracy improvement.

For both retail and news datasets, there is a significant performance gap between the ItemKNN results and the EASE^{r} accuracy scores. While it is not the goal of this paper to investigate the relative performance of these models, it is interesting to remark that more complex models that have a higher accuracy, also require a higher computational cost. Therefore, it is necessary to consider the estimated cost of each update in addition to the number of times a model is updated.

Given the estimated costs per update in Table 2, the second row of plots in Figure 3 shows the cost associated with the updates and resulting model performance. Costs scale linearly with the amount of updates, but vary strongly from model to model. For example, computing the EASE^{r} model for CosmeticsShop once is more expensive than computing the ItemKNN model 35 times. When deploying a recommender system in production, it is necessary to take this cost into account. Neural network approaches are popular in the literature, and, as we have shown, they often require high computation costs to train. In many

use-cases, simpler up-to-date models can perform better within tight budget constraints. Additionally, if we can decrease the cost of a single model computation, we can update more frequently and increase accuracy as a consequence. This highlights the importance of incremental models.

Similarly, faster-to-compute approximate approaches can also increase accuracy compared to their exact counterparts when the budget is constrained – despite losing accuracy in static evaluation scenarios. As an example, in their paper on "Markov Random Fields for Collaborative Filtering", Steck (2019c) reports that the approximate version of the algorithm requires five times less computation time, while only losing 0.8% accuracy. This means that the cost per update for the approximation is at least five times lower than that of the exact model. As such, at constant cost, the approximate model can be trained five times more often. This compensates the theoretical loss in accuracy in many production settings. For the remainder of this work, we return to an abstract notion of cost to focus on the impact of well-timed model updates on accuracy.

4.4. Comparing scheduling methods

In order to evaluate which of the methods for estimating the information gain described in Section 3.2 performs the best, we compare them for Item-kNN and EASE^{r} on all datasets.

For each of our two types of datasets, we use two different update frequencies. For the news datasets (evaluated over a period of 48 hours), we use schedules with 24 and 48 updates ($f_{update} = 12$/day and 24/day). For the retail dataset (evaluated over 14 days), we use schedules with 7 and 14 updates ($f_{update} = 0.5$/day and 1/day). We choose these numbers of updates, such that the amount of model updates does not yet counteract staleness entirely, as visualised in Figure 3. Thus, leaving room for smarter scheduling to make a measurable difference. Furthermore, these update frequencies are typical in production settings.

In order to make results between schedulers comparable, we ensured that all schedulers updated the model exactly as often as requested. Simulating

a situation where the threshold selection on the validation dataset results in exactly the same amount of updates on the test dataset. To do this, we started by computing the threshold as defined in Section 3.3, and then slightly increased or decreased it until the right amount of updates are scheduled during the test period.

While this procedure goes against good practices for optimising parameters, we show in Figure 4 that given more data than available in the offline datasets, we would be able to accurately estimate the correct threshold value that schedules exactly the right amount of updates for a given frequency. We visualise how accurate our estimates of the thresholds really are, by plotting the amount of updates scheduled by each of the methods, compared to the amount requested. We only present the results for all schedulers for the Adressa dataset, due to run-time constraints. Compared to the equal time schedule, which by design schedules the right amount of updates, the fitted parameters mostly schedule the expected amount of updates. The EV method is the strongest outlier, scheduling fewer updates than expected. This is a direct consequence of the event frequency that changes significantly throughout the dataset, showing that the EV method is more susceptible to variations in amount of traffic than the other two. In order to validate that in practice the right amount of updates would be scheduled, we also show the results on a real-world news dataset. With two months of validation data and 14 days of test data, the results confirm that given more data, the day-to-day variance is removed, such that in realistic scenarios the expected amount of updates will be scheduled during the test period, and our proposed methods compute the right thresholds. It is therefore an effect of the size of the offline datasets, that we need to modify our thresholds after computing them on the validation dataset to obtain a fair comparison.

The results of experiments comparing scheduling methods, are presenteed in Tables 3 and 4.

Key empirical observations from these experiments are as follows:

For the large majority of settings, at least one of the proposed novel methods outperforms the widely used “equal time” baseline. This confirms that smart

	ItemKNN			
	NDCG	**Recall**	**NDCG**	**Recall**
Number of updates	24		48	
Adressa				
equal time	0.0360	0.0682	0.0378	0.0721
EV	**0.0367**	0.0698	**0.0383**	0.0730
CORR	0.0353	0.0674	0.0376	0.0726
IPR	0.0364	**0.0700**	0.0382	**0.0738**
Globo				
equal time	0.0133	0.0214	0.0143	0.0233
EV	0.0136	0.0221	0.0144	0.0235
CORR	0.0136	0.0221	0.0144	0.0235
IPR	**0.0138**	**0.0223**	**0.0145**	**0.0236**
Industry news dataset				
equal time	0.0379	0.0700	0.0404	0.0749
EV	**0.0391**	**0.0725**	**0.0411**	**0.0763**
CORR	0.0377	0.0698	0.0398	0.0735
IPR	0.0388	0.0720	0.0410	0.0761
Number of updates	7		14	
CosmeticsShop				
equal time	0.0581	0.0813	**0.0587**	0.0821
EV	0.0579	0.0812	0.0584	0.0817
CORR	**0.0582**	**0.0918**	0.0585	0.0923
IPR	0.0579	0.0913	0.0586	**0.0924**
Industry retail dataset				
equal time	0.1217	0.1563	0.1222	0.1569
EV	0.1216	0.1562	**0.1223**	**0.1570**
CORR	0.1210	0.1553	0.1222	0.1568
IPR	**0.1218**	**0.1565**	**0.1223**	0.1569

Table 3: The NDCG@10 and Recall@10 results with the ItemKNN algorithm for all considered datasets. Experiments on news datasets were evaluated over a period of 48 hours, on retail datasets over a period of 14 days. The results for the best scheduling methods are shown in bold per configuration.

	EASE^r			
	NDCG	**Recall**	**NDCG**	**Recall**
Number of updates	24		48	
Adressa				
equal time	0.0461	0.0858	0.0486	0.0906
EV	0.0469	0.0877	0.0492	0.0920
CORR	0.0465	0.0878	0.0491	0.0929
IPR	**0.0473**	**0.0893**	**0.0496**	**0.0940**
Globo				
equal time	0.0192	0.0318	**0.0203**	**0.0338**
EV	0.0192	0.0320	0.0202	0.0337
CORR	**0.0197**	0.0326	0.0202	0.0336
IPR	**0.0197**	**0.0328**	0.0202	**0.0338**
Industry news dataset				
equal time	0.0461	0.0808	0.0494	0.0875
EV	**0.0477**	**0.0841**	**0.0504**	**0.0894**
CORR	0.0455	0.0797	0.0483	0.0851
IPR	0.0473	0.0832	0.0501	0.0889
Number of updates	7		14	
CosmeticsShop				
equal time	0.0809	0.1099	0.0824	0.1121
EV	0.0803	0.1094	0.0820	0.1115
CORR	**0.0812**	**0.1232**	**0.0825**	0.1253
IPR	0.0810	0.1231	**0.0825**	**0.1255**
Industry retail dataset				
equal time	**0.1479**	0.1849	0.1486	0.1861
EV	0.1478	0.1849	**0.1487**	0.1862
CORR	0.1471	0.1838	0.1486	0.1861
IPR	**0.1479**	**0.1850**	**0.1487**	**0.1863**

Table 4: The NDCG@10 and Recall@10 results for EASEr algorithm on all considered datasets. Experiments on news datasets were evaluated over a period of 48 hours, on retail datasets over a period of 14 days. The results for the best scheduling methods are shown in bold per configuration.

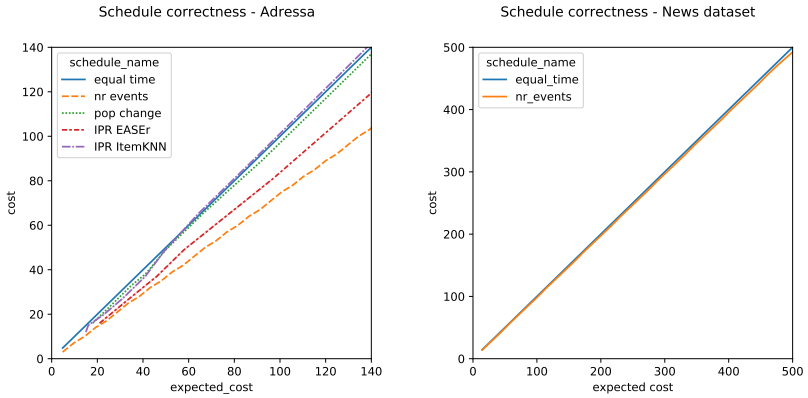

Figure 4: The x-axis shows the number of updates requested during optimisation, the y-axis shows number of updates effectively scheduled during testing. Results are shown for the Adressa dataset, using 2 days of validation data to select the threshold parameters. The right plot presents the schedule correctness for the number of events scheduling threshold. We perform this only for this schedule, as it is the one with highest variance in the experiments on the small dataset.

scheduling regimes can improve recommendation performance at constant cost.

The IPR method generalises the best to all datasets, either performing superior or close second. This confirms that the method is able to detect the right signals to decide on updating the model, regardless of the context in which it is used.

Since both IPR and CORR improve over EV in some settings, we confirm our hypothesis that not all events carry the same amount of information. Depending on the context, it is necessary to take this into account to find the optimal schedule.

Each of the three methods is model-agnostic, and can be used with any model that generates recommendation scores. The EV and CORR methods do not depend on the deployed model for choosing their updates. IPR conversely does depend on the relevance scores predicted by the recommendation model, but the results for ItemKNN and EASE show that it generalises to different models.

Improvements with novel methods are most notable on the Recall@10 metric, rather than on the NDCG@10 metric. This can be explained by the fact that new items typically get recommended at the bottom of the top-K. Recall is

much more sensitive to these lower ranked, but correct new items, compared to NDCG, which focuses on ranking the correct items at the top of the list.

4.5. Online Experiments

In order to validate the offline results in this article, we also performed several online experiments on both news and retail websites.

For the news use-case we got the opportunity to use a Dutch-language, local, newspaper website. The staleness rates in the offline experiments showed patterns matching the Adressa dataset, suggesting that models will grow stale quickly when not retrained.

For the retail use-case we used two very different online webshops. A "traditional" retail webshop, showing similar staleness patterns as CosmeticsShop and a "flash" retail webshop, where items are only available for a limited time, whose offline staleness patterns showed behaviour between the news and retail datasets. Models do grow stale, but not as quickly as they do for news.

Model staleness. To confirm that the results for our model staleness hypotheses presented in Section 4.3.2 also hold in an online setting, we performed three randomised controlled trials, colloquially known as an A/B-test, with control/treatment corresponding to the same recommendation model updated at different intervals. In these trials, we do not yet make use of the new scheduling methods we proposed, but instead use the traditional fixed schedule.

In the first of these trials we verified that model staleness impacts performance quickly in a fast moving news context, such that faster updates result in better performance. For the control group, we updated the model every 30 minutes and the treatment group had its model updated at an increased rate of once every 15 minutes. Based on our offline experiments, we expected the treatment group to outperform the control group due to the reduced staleness in the serving models. This was confirmed after a 2-week trial, in which the treatment group showed a 2% relative increase in click-through-rate (CTR) compared to the control group.

In a second trial on the traditional webshop, we verified that fewer updates did not reduce recommendation performance. For the control group in this trial the model is updated every 3 hours, while for the treatment group it is updated once per day. We expected no significant difference in CTR between the two groups, given the offline results. After a trial that ran for 4 weeks with more than 1.5 million recommendation opportunities for each of the groups, indeed no significant difference was found between the two schedules. This is an important insight: indeed, reducing the number of model updates by a factor of 8 implies reducing the computational costs for this model by a factor of 8.

The final trial was held on the flash-retail website. Our expectation was that increasing the amount of model updates from 4 per day (control group) to 8 per day (treatment group) would prove beneficial on both CTR and conversion. This was confirmed after the 10-day trial, with the treatment group showing a significant improvement in both CTR (+20%) and conversion (+25%) over the control group.

Smart scheduler. To extend our evaluation of the smart scheduling methods proposed, we performed two trials where a smart scheduling approach was compared to a baseline fixed schedule approach. During these trials we opted to use the scheduler based on the CORR information metric, computed on a "recently popular" model, for practical reasons. While the IPR scheduler performed better in many of the offline settings, it was much harder to integrate into the recommendation pipeline.

For the first trial, on the news website, we used the winning treatment from our previous test (identifying model staleness) as the control group in this test. The models are updated according to a fixed schedule, with an update every 15 minutes. The treatment group used the CORR scheduler, whose threshold was fitted to schedule 48 updates per day using a validation dataset. Note that the treatment will perform half as many updates as the control group, which without the scheduler was significantly worse in the previous test. This trial was carried out on 3 different lists of recommendations on the website, each

shown on different pages. After a trial of one week, no significant difference was found between the two groups for two out of three lists. On one of the lists, the control group performed significantly better than the treatment group, though the improvement was small ($< 1\%$) especially given that it required twice the cost. As such, this test confirms that the CORR scheduler manages to schedule its updates at better timepoints, so it requires only half of the updates—and half of the budget— to get the improved performance we previously only found using 96 updates per day.

The second trial was held on the flash retail website. During this trial the control group received recommendations from a model updated 8 times per day, and the models for the treatment group were scheduled using the CORR scheduler whose threshold was fitted to also schedule 8 models per day. Over the evaluation period of 2 weeks, the CORR scheduler method showed a significant ($p < 0.05$) improvement of 2% in CTR (relative) over the control group.

Given the positive online results of the CORR scheduler, and the better offline performance of IPR, we are considering it future work to also implement the IPR scheduler online, and verify its offline superiority over the popularity change scheduler in online trials.

5. Conclusions

In this article we have demonstrated the effect of retraining and model staleness in different environments and for a variety of collaborative filtering approaches, and we have studied the cost-accuracy trade-off. Achieving higher accuracy requires increased computational costs up to some asymptotic point, after which more updates have no further benefit. In real world applications of recommendation systems we cannot ignore this cost of computing models and therefore it is paramount to find an optimal balance between cost and accuracy. Our results indicate the importance of models that can be efficiently and incrementally trained for real-world applications – because they allow frequent updates while keeping costs low.

We have proposed a generic method to create a smart schedule for retraining or updating models, which results in higher accuracy than retraining at fixed intervals given the same resources. Our scheduling approach uses a heuristic to comply to the budget constraint. Applying this budget constraint to control methods and drift detection methods is an interesting avenue to further improve the use of resources in practice.

In our experiments, we have hinted at the fact that up-to-date models also have an impact on other metrics, such as fairness or diversity. It remains interesting future work to investigate this impact thoroughly.

Acknowledgment

This work has been supported by Agentschap Innoveren en Ondernemen, Belgium [HBC.2019.2178].

References

Al-Ghossein, M., Murena, P., Abdessalem, T., Barré, A., Cornuéjols, A., 2018. Adaptive collaborative topic modeling for online recommendation, in: Proc. of the 12th ACM Conference on Recommender Systems, ACM. p. 338–346.

Anyosa, S.C., Vinagre, J., Jorge, A.M., 2018. Incremental matrix co-factorization for recommender systems with implicit feedback, in: Companion Proceedings of the The Web Conference 2018, International World Wide Web Conferences Steering Committee. p. 1413–1418.

Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y., 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. Journal of the ACM (JACM) 45, 891–923.

Beel, J., Brunel, V., 2019. Data pruning in recommender systems research: Best practice or malpractice?, in: Proc. of the 13th ACM Conference on Recommender Systems, ACM.

Bifet, A., Gavalda, R., 2007. Learning from time-changing data with adaptive windowing, in: Proc. of the 2007 SIAM international conference on data mining, SIAM. pp. 443–448.

Blei, D.M., Ng, A.Y., Jordan, M.I., 2003. Latent dirichlet allocation. the Journal of machine Learning research 3, 993–1022.

Breck, E., Cai, S., Nielsen, E., Salib, M., Sculley, D., 2017. The ml test score: A rubric for ml production readiness and technical debt reduction, in: Proc. of the 2017 IEEE International Conference on Big Data, IEEE. pp. 1123–1132.

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

Cañamares, R., Castells, P., Moffat, A., 2020. Offline evaluation options for recommender systems. Information Retrieval Journal 23, 387–410.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Comput. Surv. 46.

Gulla, J.A., Zhang, L., Liu, P., Özgöbek, O., Su, X., 2017. The adressa dataset for news recommendation, in: Proc. of the International Conference on Web Intelligence, ACM. p. 1042–1048.

Jambor, T., Wang, J., Lathia, N., 2012. Using control theory for stable and efficient recommender systems, in: Proc. of the 21st international conference on World Wide Web, pp. 11–20.

Jeunen, O., 2019. Revisiting offline evaluation for implicit-feedback recommender systems, in: Proc. of the 13th ACM Conference on Recommender Systems, ACM. p. 596–600.

Jeunen, O., Van Balen, J., Goethals, B., 2022. Embarrassingly shallow autoencoders for dynamic collaborative filtering. User Modeling and User-Adapted Interaction .

Jeunen, O., Verstrepen, K., Goethals, B., 2018. Fair offline evaluation methodologies for implicit-feedback recommender systems with MNAR data, in: Proc. of the ACM RecSys Workshop on Offline Evaluation for Recommender Systems.

Jeunen, O., Verstrepen, K., Goethals, B., 2019. Efficient similarity computation for collaborative filtering in dynamic environments, in: Proc. of the 13th ACM Conference on Recommender Systems, ACM. p. 251–259.

Kendall, M.G., 1945. The treatment of ties in ranking problems. Biometrika 33, 239–251.

Klinkenberg, R., Renz, I., 1998. Adaptive information filtering: Learning in the presence of concept drifts. Learning for Text Categorization , 33–40.

Koren, Y., Bell, R., Volinsky, C., 2009. Matrix factorization techniques for recommender systems. Computer 42, 30–37.

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., Lin, X., 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. IEEE Transactions on Knowledge and Data Engineering 32, 1475–1488.

Liang, D., Krishnan, R.G., Hoffman, M.D., Jebara, T., 2018. Variational autoencoders for collaborative filtering, in: Proc. of the World Wide Web Conference, ACM. pp. 689–698.

Ludewig, M., Jannach, D., 2018. Evaluation of session-based recommendation algorithms. User Modeling and User-Adapted Interaction 28, 331–390.

Moreira, G.D.S.P., Jannach, D., da Cunha, A.M., 2019. Contextual hybrid session-based news recommendation with recurrent neural networks. IEEE Access 7, 169185–169203.

Ning, X., Karypis, G., 2011. Slim: Sparse linear methods for top-n recommender systems, in: Proc. of the 2011 IEEE 11th International Conference on Data Mining, IEEE Computer Society. pp. 497–506.

Qahtan, A.A., Alharbi, B., Wang, S., Zhang, X., 2015. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams, in: Proc. of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM. p. 935–944.

Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L., 2009. BPR: Bayesian personalized ranking from implicit feedback, in: Proc. of the 25th Conference on Uncertainty in Artificial Intelligence, AUAI Press. pp. 452–461.

Sarwar, B., Karypis, G., Konstan, J., Riedl, J., 2001. Item-based collaborative filtering recommendation algorithms, in: Proc. of the 10th International Conference on World Wide Web, ACM. pp. 285–295.

de Souza Pereira Moreira, G., Ferreira, F., da Cunha, A.M., 2018. News session-based recommendations using deep neural networks, in: Proc. of the 3rd Workshop on Deep Learning for Recommender Systems, pp. 15–23.

Steck, H., 2019a. Collaborative filtering via high-dimensional regression. CoRR abs/1904.13033. arXiv:1904.13033.

Steck, H., 2019b. Embarrassingly shallow autoencoders for sparse data, in: Proc. of the World Wide Web Conference, p. 3251–3257.

Steck, H., 2019c. Markov random fields for collaborative filtering, in: Advances in Neural Information Processing Systems 32, pp. 5473–5484.

Verstrepen, K., Bhaduriy, K., Cule, B., Goethals, B., 2017. Collaborative filtering for binary, positiveonly data. SIGKDD Explor. Newsl. 19, 1–21.

Vinagre, J., Jorge, A., 2012. Forgetting mechanisms for scalable collaborative filtering. Journal of the Brazilian Computer Society 18, 271–282.

Vinagre, J., Jorge, A.M., Al-Ghossein, M., Bifet, A., 2020. ORSUM - Workshop on Online Recommender Systems and User Modeling. ACM. p. 619–620.

Vinagre, J., Jorge, A.M., Gama, J., 2014. Fast incremental matrix factorization for recommendation with positive-only feedback, in: User Modeling, Adaptation, and Personalization, Springer International Publishing, Cham. pp. 459–470.

Widmer, G., Kubat, M., 1996. Learning in the presence of concept drift and hidden contexts. Machine Learning 23, 69–101.

Yu, S., Abraham, Z., 2017. Concept drift detection with hierarchical hypothesis testing, in: Proc. of the 2017 SIAM International Conference on Data Mining (SDM), pp. 768–776.

Zanardi, V., Capra, L., 2011. Dynamic updating of online recommender systems via feed-forward controllers, in: Proc. of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 11–19.

Zhang, Y., Feng, F., Wang, C., He, X., Wang, M., Li, Y., Zhang, Y., 2020. How to retrain recommender system? a sequential meta-learning method, in: Proc. of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM. p. 1479–1488.