Incremental Specialized and Specialized-Generalized Matrix Factorization Models based on Adaptive Learning Rate Optimizers

Antônio David Viniski[a,*], Jean Paul Barddal[a], Alceu de Souza Britto Jr.[a], Humberto Vinicius Aparecido de Campos[b]

[a] *Graduate Program in Informatics (PPGIa), Pontifícia Universidade Católica do Paraná (PUCPR), Rua Imaculada Conceição, 1155, 80215-901, Curitiba, Brazil*
[b] *Himarket, Avenida Visconde de Guarapuava, 2764, Curitiba, Brazil*

Abstract

Recommender systems suggest items that are likely to be preferred by a particular user based on historical behavior, actions, and feedback. In real-world applications, data on users and items are continuously generated at a fast pace, such as in e-commerce, social media, digital marketing, and content consumption applications. Since interactions occur over time, these scenarios can be formulated as a data stream where users' interests are potentially dynamic, i.e., they change over time. Given that changes are expected to occur, one of the current research challenges in streaming recommender systems is that models must adapt their parameters when changes occur to maintain performance. As such changes do not occur for all users and items in the stream at the same time, we consider adapting learning schemes to account for user or item identifiers and model individual parameters. Therefore, we used specialized parameters to adjust the step size for each dataset user or item. More specifically, this study proposes four specialized and specialized-generalized variants of four well-known adaptive learning rate optimizers and shows how they are combined with incremental matrix factorization methods. We tested our proposed optimization strategies on

*Corresponding author
Email addresses: `adviniski@ppgia.pucpr.br` (Antônio David Viniski), `jean.barddal@ppgia.pucpr.br` (Jean Paul Barddal), `alceu@ppgia.pucpr.br` (Alceu de Souza Britto Jr.), `humberto.campos@himarket.club` (Humberto Vinicius Aparecido de Campos)

different datasets and showed that one of the proposed specialized variants, that is, InAMSGradUser, improves the RECALL and NDCG rates by up to 11.1 and 7.5 percentage points, respectively, compared to the traditional stochastic gradient descent (SGD) optimizer.

Keywords:
Matrix factorization, Optimization algorithms, Collaborative filtering

1. Introduction

Given the increasing amount of data collected, recommendation systems became indispensable in several scenarios in which user profiling is essential. Modeling the user consumption patterns of products and services allows for personalized and relevant recommendations [1]. Consequently, recommender systems have been consistently explored by both researchers and practitioners because these are relevant marketing tools in a multitude of scenarios [2]. For instance, companies such as Amazon, Netflix, and Google News stand out for using efficient recommendation methods, which improve user experience and enhance user engagement towards products and services [3].

According to the available data and recommendation process characteristics, studies have classified recommendation models into three categories: content-based, collaborative, and hybrid filtering [2, 4]. In collaborative filtering systems, data must be in the form of a list of interactions between users and items, whereas in content-based filtering, characteristics that describe items are also required. Consequently, collaborative filtering is less restrictive and has been the subject of many studies over the years [4, 5, 6].

Most proposed recommendation techniques operate in a batch fashion. In such scenarios, given a training set comprising user-item interactions, a static model is generated and later used for recommendations [7]. However, the world is dynamic, and it is realistic to assume that interactions between users and items may become available over time [8]. Consequently, user preferences may change over time (concept drift), new items can be made available (cold start), or even the system rules can be modified (e.g., clearance sales and actions in periods of the year, month, or week) [6]. Thus, developers of such systems must also be aware of concept drift and cold start issues, which require recommender systems to work incrementally, continuously, and consistently to detect and adapt to changes in data so that performance is not jeopardized [9].

As changes in the data stream do not occur in all dataset users and items simultaneously, we consider adapting the recommender model learning schemes in a specialized manner. However, specialization can be ensured in addition to user or item parameter learning process. In practice, the definition of how specialization takes place includes analyzing the definition of particular step sizes for each user and item in the dataset and adjusting the parameters according to the recommender performance.

In this study, we propose adaptive optimization strategies for well-known incremental matrix factorization (MF) methods. Our approaches extend RMSprop [10], Adam [11], AMSGrad [12], and Nadam [13] optimizers, which are adaptive gradient descent-based algorithms often used in a batch fashion. Our proposed method differs from the MLF [14] optimizer in terms of the use of a single additional array, as we specify the target of personalization: the user or item. Furthermore, we apply personalization in an incremental scenario, in which models are adjusted according to changes in data and with positive-only feedback data. More specifically, our contribution is two-fold:

- The proposal of optimizer variants that include both user and item specialized and specialized-generalized incremental methods. These optimizers are also coupled with existing MF algorithms.
- An empirical analysis that demonstrates that adaptive learning rate optimizers induce better adjustment in the MF model's parameters and, consequently, more accurate recommendations.

The remainder of this paper is organized as follows. Section 2 describes related work on existing incremental positive-only MF models and adaptive learning rate optimizers. Section 3 details the proposed optimization methods and how they are coupled with the existing incremental MF models. Section 4 describes the experimental protocol used to perform the proposed analysis of recommender models. We discuss the obtained results in Section 5. Finally, Section 6 concludes the paper and describes future work.

2. Related Works

This section introduces related works on incremental recommender systems based on Matrix Factorization (MF) and presents existing adaptive learning rate optimizers available in the literature.

At this point, it is worthy to highlight recent works on latent factor and matrix factorization that tackle high-dimensional and sparse datasets (HiDS). First, the work of [15] proposes an ensemble of Alternating Least Squares (ALS) matrix factorization algorithm in estimating Quality-of-Service rates. Under different settings, the proposal overcame bayesian matrix factorization [16]. Another relevant work is latent factor analysis via particle swarm optimization (PSO) [17], in which SGD and Adam optimizers had the learning rates hyper-parameters adaptive tuned for HiDS applications. Next, authors in [18] proposed a combination of L_1 and L_2 norms to overcome data sparsity problems inherent to HiDS datasets. Experimentation showed that this approach overcomes traditional matrix factorization algorithms in terms of robustness and stability in different applications. More recently, authors in [19] proposed a graph-convolutional network approach for latent feature analysis in Quality-of-Service systems. The main singularity of this approach is that the graph-convolutional network allows the identification of non-linear relationship between users and item as well as amongst items. Finally, despite the recentness and relationship with our work, such works are tailored for HiDS batch scenarios and require adaptations to be used in streaming settings.

2.1. Matrix Factorization

Matrix Factorization (MF) is the most commonly used technique in recommendation system design [20, 5], and the most successful latent factor models are also based on MF [21]. MF models have been widely applied to different recommendation scenarios and outperform clustering and neighborhood methods in terms of predictive power, run time [22], and scalability [21].

The MF input is a relation matrix between users and items $R \in \mathbb{R}^{m \times n}$, where m denotes the number of users and n denotes the number of items [23]. Because of R's sparsity, MF models map users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space [21, 23]. The number of latent factors (f) is much smaller than the number of users and items, and the co-occurrence between users and items forms the basis for recommendations [20].

More formally, matrices $R \in \mathbb{R}^{m \times n}$, $\overrightarrow{A} \in \mathbb{R}^{m \times f}$ and $\overrightarrow{B} \in \mathbb{R}^{f \times n}$ represent users' ratings to items, users' latent vectors, and items' latent vectors, respectively. The $r_{u,i}$ entry in the u-th row and i-th column of R an it represents the rating that user u gives to item i. The u-th row vector ($\overrightarrow{A_u}$) of A and

i-th column vector ($\overrightarrow{B_i}$) of B are the user's u and item's i latent vectors, respectively [5]. Given this formulation, it is possible to compute the predicted rating between the u-th user and i-th item using the dot product depicted in Equation 1 [22].

$$\widehat{r}_{ui} = \overrightarrow{A_u} \cdot \overrightarrow{B_i}^T \tag{1}$$

MF is closely related to singular value decomposition (SVD), a well-established technique for information retrieval to identify latent features [24]. Applying SVD to collaborative filtering requires factoring in the user-item matrix. However, the conventional SVD is undefined when the rating matrix is incomplete. Given the high percentage of missing values, some methods use imputation techniques to fill in the missing values and make the rating matrix dense before applying SVD [25, 26]. However, these approaches can be costly and considerably distort the data owing to inaccurate imputation [21, 24]. Hence, more recent researches suggest direct modeling of the observed ratings while avoiding overfitting through a regularized model [21, 24]. Thus, the model was trained by minimizing L2-regularized squared error for known values of $\widehat{R}$ and the corresponding predicted ratings are denoted in Equation 2 [22].

$$\min_{\overrightarrow{A},\overrightarrow{B}} \sum_{(u,i)\in D} \left(r_{u,i} - \overrightarrow{A_u} \cdot \overrightarrow{B_i}^T\right)^2 + \lambda \left(\left\|\overrightarrow{A_u}\right\|^2 + \left\|\overrightarrow{B_i}\right\|^2\right) \tag{2}$$

In the formalization above, D is the set of (u, i) pairs for each known $r_{u,i}$ (training set), and λ is the regularization parameter for user $\overrightarrow{A_u}$ and item $\overrightarrow{B_i}$ latent vectors that are used to avoid overfitting. MF learns a model by fitting previously observed interactions. However, the goal is to generalize previously known user-item interactions to predict future, unobserved user preferences over items [21].

2.1.1. Incremental Stochastic Gradient Descent

The most common approach for minimizing Equation 2 in MF is the SGD optimization, in which the algorithm loops over all ratings in the training set. For each given training interaction, the system predicts r_{ui} and computes the associated prediction error (err_{ui}) using Equation 3 [21].

$$err_{ui} = r_{ui} - \widehat{r}_{ui} \tag{3}$$

In this process, both the user ($\overrightarrow{g_u}$) and item ($\overrightarrow{g_i}$) gradients of the error (Equation 4) are used, where λ is the regularization rate.

$$\begin{aligned}\overrightarrow{g_u} &\leftarrow err_{ui} \times \overrightarrow{B_i} - \lambda\overrightarrow{A_u} \\ \overrightarrow{g_i} &\leftarrow err_{ui} \times \overrightarrow{A_u} - \lambda\overrightarrow{B_i}\end{aligned} \tag{4}$$

Next, SGD modifies the parameters by a magnitude proportional to η in the inverse direction of the error's gradient according to Equation 5 [21], where η is the step size or learning rate, and λ is the regularization term for both user and item latent factors [22].

$$\begin{aligned}\overrightarrow{A_u} &\leftarrow \overrightarrow{A_u} + \eta \times \overrightarrow{g_u} \\ \overrightarrow{B_i} &\leftarrow \overrightarrow{B_i} + \eta \times \overrightarrow{g_i}\end{aligned} \tag{5}$$

In contrast, incremental SGD (ISGD) [22] was initially designed for positive-only feedback recommendations; thus, the interactions between users and items are represented as a Boolean matrix. Consequently, the prediction error is simplified, and its computation is given by Equation 6.

$$err_{ui} = 1 - \widehat{R}_{ui} \tag{6}$$

Similar to the traditional batch MF model, the ISGD updates the A_u and B_i vectors using Equation 5. A relevant difference between ISGD and its batch counterpart (SVD MF with batch SGD optimizer) is the order in which the method analyzes the data to generate the models [6]. The traditional batch SVD shuffles and performs multiple passes over the training data during preprocessing and model creation, whereas ISGD does not perform any data preprocessing and processes data according to their natural arrival order [22].

2.1.2. Probabilistic MF

Similarly to the traditional MF model, the Probabilistic MF (PMF) model decomposes a binary matrix into two lower-rank matrices to make predictions [27]. The main difference is related to the use of positive (ones, 1) and unknown (zeros, 0) observations during model training, that is, the matrix represents a user's likes and unknown/potential dislikes for items.

Equation 7 describes the PMF process, where σ represents a sigmoid function (σ), that applied to compute probabilistic predictions ($0 \leqslant \widehat{r_{ui}} \leqslant 1$). In the update process of the PMF's user ($\overrightarrow{A_u}$) and item ($\overrightarrow{B_i}$) latent factor vectors, ISGD update rules are used, except for error calculation, because

PMF uses both positive and negative (unknown) observations. In other words, if the instance is positive, then $err_{ui} = 1 - \widehat{r_{ui}}$. Otherwise, if it was a negative instance, the error was calculated using $err_{ui} = 0 - \widehat{r_{ui}}$.

$$\widehat{r_{ui}} \leftarrow \sigma\left(A_u \cdot B_i^T\right) \tag{7}$$

Furthermore, assuming T as the test set, when the user-item interaction is a binary outcome ($r_{u,i} \in \{0, 1\}$), it is natural to use cross entropy (Log-Loss, Equation 8) as the loss function for the optimization problem [27].

$$\text{Log-loss} = -\frac{1}{|T|} \sum_{u,i \in T} \left(R_{u,i} \log\left(\widehat{R}_{ui}\right) + (1 - R_{u,i}) \times \log(1 - \widehat{R}_{ui}) \right) \tag{8}$$

2.1.3. Bayesian Personalized Ranking for MF

Bayesian personalized Ranking for MF (BPRMF) is a generic optimization criterion for personalized ranking derived from a Bayesian problem analysis. The BPRMF model provides a learning BPR method based on SGD with bootstrap sampling [28]. In addition to the MF parameters, for each instance $(u; i)$ in the training set, BPRMF selects a negative item j (an item that the u-th user does not interact with). BPR optimization decomposes triplets in the $(u; i; j)$ format using the difference in the u-th user predictions w.r.t. items i and j, as shown in Equation 9.

$$\widehat{r}_{uij} = \widehat{r}_{ui} - \widehat{r}_{uj} \tag{9}$$

Next, the model applies a sigmoid function variant described in Equation 10 to the prediction $\widehat{r}_{uij}$ to add the Bayesian probabilistic characteristic to the model [29].

$$\sigma(\widehat{r}_{uij}) = \left(\frac{1}{1 + e^{-\widehat{r}_{uij}}}\right) \tag{10}$$

As the BPRMF has an additional term j, the model differs from ISGD when computing the gradients of the error for each term in the triplet $(u; i; j)$ (Equation 11).

$$\begin{aligned}\overrightarrow{g_u} &\leftarrow \sigma(\widehat{r}_{uij}) \times (B_i - B_j) - \lambda\overrightarrow{A_u} \\ \overrightarrow{g_i} &\leftarrow \sigma(\widehat{r}_{uij}) \times (A_u) - \lambda\overrightarrow{B_i} \\ \overrightarrow{g_j} &\leftarrow \sigma(\widehat{r}_{uij}) \times (-A_u) - \lambda\overrightarrow{B_j}\end{aligned} \tag{11}$$

Finally, for each triplet $(u;i;j)$, the latent factor vectors for user u, item i, and unobserved item j are updated using the traditional SGD optimization strategy described in Equation 12.

$$\begin{aligned}A_u &\leftarrow A_u + \eta \times \overrightarrow{g_u} \\ B_i &\leftarrow B_i + \eta \times \overrightarrow{g_i} \\ B_j &\leftarrow B_j + \eta \times \overrightarrow{g_j}\end{aligned} \tag{12}$$

2.1.4. Momentum-incorporated Latent Factor Model (MLF)

The Momentum-incorporated latent factor (MLF) algorithm is an optimization strategy introduced as part of the momentum-incorporated parallel SGD (MPSGD) model [14]. The main purpose of the MPSGD model is to paralyze the update of latent factors during the batch training step of the MF model [14]. In contrast, MLF is an SGD-based learning method designed to update the latent factors matrices of user A and item B, which can be integrated into an incremental learning scenario.

The momentum method is an adaptive optimization strategy for improving the convergence rate of an SGD-based model. The momentum algorithm improves the model convergence by adding a fraction γ of the previous interaction updates to the current interaction update. Thus, the learning update in the current interaction considers the direction trend from the previous updates, reducing oscillations during the learning process and making the resultant model converge faster [30, 14]. Equation 13 presents the update rules of the momentum optimizer, where $\overrightarrow{v_t}$ stores the history of the previous update, γ is the momentum term (also known as the decay rate), $\overrightarrow{g}$ represents the gradients of the error, and $\overrightarrow{\theta}$ represents the latent factor vector (A_u, B_i).

$$\begin{aligned}\overrightarrow{v_t} &\leftarrow \gamma \times \overrightarrow{v_{t-1}} + \eta \times \overrightarrow{g} \\ \overrightarrow{\theta} &\leftarrow \overrightarrow{\theta} - \overrightarrow{v_t}\end{aligned} \tag{13}$$

MLF incorporates momentum optimization by introducing $V_{(A)}$ and $V_{(B)}$ auxiliary arrays. These arrays store the history of previous updates for each user u and item i in the dataset. Consequently, to update the user latent factor vector $\overrightarrow{A_u}$ with the MLF model, the update rules depicted in Equation 14 must be used.

$$\begin{aligned} \overrightarrow{v_{u(t)}} &\leftarrow \gamma \times \overrightarrow{v_{u(t-1)}} + \eta \times \overrightarrow{g_u} \\ \overrightarrow{A_u} &\leftarrow \overrightarrow{A_u} - \overrightarrow{v_{u(t)}} \end{aligned} \tag{14}$$

Similarly, for the item latent factor vector $\overrightarrow{B_i}$ updates, we used the rules in Equation 15.

$$\begin{aligned} \overrightarrow{v_{i(t)}} &\leftarrow \gamma \times \overrightarrow{v_{i(t-1)}} + \eta \times \overrightarrow{g_i} \\ \overrightarrow{B_i} &\leftarrow \overrightarrow{B_i} - \overrightarrow{v_{i(t)}} \end{aligned} \tag{15}$$

2.2. Adaptive Learning Rate Optimizers

Most machine-learning methods can be formulated as optimization problems to determine the extremum of an objective function. Accordingly, the three vital steps of a machine learning algorithm are (i) building a model hypothesis, (ii) defining the objective function, and (iii) solving the maximum or minimum of the objective function to determine the model parameters. In this sense, the first two steps are modeling problems, and the third step solves the problem using optimization algorithms [31].

Therefore, optimization is a mathematical discipline used to solve decision-making problem. The basic idea of optimization is to determine the best solution among the various available options in the given objective function [32]. Traditionally, optimization research is divided into first-order, high-order, and derivative-free methods [31, 32]. First-order optimization methods, known as gradient-based optimization, are primarily based on first-order derivatives or gradient descent algorithms. High-order methods, in turn, are used to address the problem in which an objective function exhibits highly non linear and ill-conditioned behavior. Finally, we use derivative-free methods in real-world optimization problems, where the derivative of the objective and constraint functions may not exist or is difficult to calculate.

In this study, we focus on traditional MF models, in which the most frequently used optimization strategy to minimize the objective function is gradient descent-based methods, such as RMSProp, Adam, AMSGrad, Nadam, Momentum, Adadelta, and Adagrad optimizers. Therefore, the following sections introduce the RMSProp, Adam, AMSGrad, Nadam, and gradient descent optimization strategies selected to replace the traditional

SGD optimizer in the MF model. We selected RMSProp, Adam, AMSGrad, and Nadam optimizers because they exhibited superior and more robust performance in many works when compared to other optimization techniques [33, 34, 35].

2.2.1. RMSprop optimizer

RMSprop [10] is an adaptive learning rate method that is similar to the SGD algorithm with momentum. It was developed as a stochastic technique for mini-batch learning, which is suitable for online and non-stationary settings. RMSprop and gradient descent differ in gradient computation. RMSProp uses the exponential decaying average $\overrightarrow{v_t}$ (Equation 16) method to discard the squared values of the gradients distant from the current time step. The algorithm converges rapidly after obtaining convex structure. In RMSprop's update rules depicted in Equation 17, the gradients ($\overrightarrow{g}$) are divided by the running average of their recent magnitudes ($\overrightarrow{v_t}$) [10]. The γ parameter is the momentum (also called the decay rate) and is usually set to 0.9 [10]. Considering that the value of $\overrightarrow{v_t}$ may converge to 0, weights can "blow up". To prevent the gradients from blowing up, RMSprop includes a padding factor ϵ in the denominator, often $\epsilon = 10^{-8}$, to avoid inconsistent computation [11].

$$\overrightarrow{v_t} \leftarrow \gamma \times \overrightarrow{v_{t-1}} + \overrightarrow{g}^2(1-\gamma) \tag{16}$$

$$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \overrightarrow{g}\frac{\eta}{\sqrt{\overrightarrow{v_t}} + \epsilon} \tag{17}$$

RMSprop deals with gradient vanishing and explosion issues using a moving average of past squared gradients, normalizing the gradient accordingly. This normalization adapts the learning rate, decreasing it for large gradients to avoid exploding and increasing it for small gradients to avoid vanishing [10, 36].

2.2.2. Adaptive Moment Estimation - Adam

Adaptive moment estimation (Adam) computes adaptive learning rates for each model parameter [11]. Similar to RMSprop, Adam stores an exponentially decaying average of the past squared gradients $\overrightarrow{v_t}$ (Equation 18).

$$\overrightarrow{v_t} \leftarrow \beta_2\overrightarrow{v_{t-1}} + (1-\beta_2)\overrightarrow{g}^2 \tag{18}$$

Adam also mantains an exponentially decaying average of past gradients m_t, similar to momentum (Equation 19).

$$\overrightarrow{m_t} \leftarrow \beta_1 \overrightarrow{m_{t-1}} + (1 - \beta_1) \overrightarrow{g} \tag{19}$$

The $\overrightarrow{m_t}$ and $\overrightarrow{v_t}$ parameters are estimates of the first (the mean) and the second moments (the uncentered variance) of the gradients, respectively; hence the method is named accordingly. As $\overrightarrow{m_t}$ and $\overrightarrow{v_t}$ are initialized as vectors of 0s (zeroes), they are biased towards zero, especially during the initial time steps and especially when the decay rates are low (i.e., β_1 and β_2 are close to 1, propose default values of 0.9 and 0.999, respectively). Thus, to counteract these biases, Adam computes the bias-corrected first and second moment estimates using Equations 20 and 21, respectively, where t is the current time step (epoch).

$$\overrightarrow{\widehat{m}_t} \leftarrow \frac{\overrightarrow{m_t}}{(1 - \beta_1)^t} \tag{20}$$

$$\overrightarrow{\widehat{v}_t} \leftarrow \frac{\overrightarrow{v_t}}{(1 - \beta_2)^t} \tag{21}$$

Finally, Adam's optimization update rule in Equation 22 uses the first (Equation 20) and second-moment (Equation 21) estimations as follows:

$$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \frac{\overrightarrow{\widehat{m}_t}}{\sqrt{\overrightarrow{\widehat{v}_t}} + \epsilon} \tag{22}$$

2.2.3. AMSGrad Optimizer

The AMSGrad [12] algorithm overcomes the problem in which Adam cannot converge to the optimal solution. The AMSGrad optimizer proposes a new update method that uses the maximum of the past squared gradients $\max(\overrightarrow{v_t})$ (Equation 23) rather than the exponential average ($\overrightarrow{v_t}$) directly to update the parameters (Equation 24). AMSGrad also calculates the exponentially decaying average of past squared gradients ($\overrightarrow{v_t}$ - Equation 18) and the exponentially decaying average of past gradients ($\overrightarrow{m_t}$ - Equation 19).

$$\max(\overrightarrow{v_t}) \leftarrow \max\left(\max(\overrightarrow{v_t}), \overrightarrow{v_t}\right) \tag{23}$$

$$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \frac{\overrightarrow{m_t}}{\sqrt{\max(\overrightarrow{v_t})} + \epsilon} \tag{24}$$

2.2.4. Nesterov-accelerated Adaptive Moment Estimation (Nadam)

The Nesterov-accelerated adaptive moment estimation (Nadam) [13] algorithm combines the Adam and Nesterov-accelerated gradient (NAG). To incorporate NAG into Adam, the authors modified its momentum term $\overrightarrow{m_t}$, thus defining the adjusted momentum ($\overrightarrow{\widehat{m}_{a(t)}}$) and similarly replacing the previous momentum vector with the current momentum vector. To simplify the adjusted moment term ($\overrightarrow{\widehat{m}_{a(t)}}$), Nadam used the corrected bias ($\overrightarrow{\widehat{m}_t}$ - Equation 20) of the current moment term.

$$\overrightarrow{\widehat{m}_{a(t)}} \leftarrow \beta_1 \overrightarrow{\widehat{m}_t} + \frac{(1-\beta_1)\,\overrightarrow{g_t}}{1-\beta_1^t} \tag{25}$$

This algorithm increases or decreases the decay factor β_1 over time. Nadam's update rule is given in Equation 26.

$$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \frac{\overrightarrow{\widehat{m}_{a(t)}}}{\sqrt{\overrightarrow{\widehat{v}_t}} + \epsilon} \tag{26}$$

3. Proposed Methods

This section introduces the proposed Adam, AMSGrad, Nadam, and RMSProp variants that leverage adaptive learning rate optimizers previously introduced in Section 2.2. The main purpose of our proposed optimizer variants is to consider the user or item in the data-stream instance to update the optimizer parameters. Such an analysis enables the learning of personalized optimizer terms for each user or item in the data stream. In contrast to the MLF optimizer described in Section 2, our proposed method uses a single auxiliary array (per user or item) to personalize optimizer parameters.

The rationale for proposing optimizer variants is to consider the dynamic characteristics of streaming recommendation scenarios. Such dynamism occurs because the incidence of cold-start and concept drift problems, which requires recommendation models to adapt their learning schemes to maintain performance. As changes in the data behavior are not observed in the entire dataset and are only for a few users or items, we propose updating the optimizer learning steps individually. The idea here is not to implement an explicit drift detector or treat the new users and new items differently because of the cold-start incidence but, instead, to make the model adaptive according to the observed performance.

As both optimizers have as parameters the squared decaying average v_t and/or the decaying average m_t, we show the implementation of the AMSGrad optimizer variants owing to its additional parameter $\texttt{max}(v_t)$. Thus, the following algorithms are prototypes for other adaptive learning rate optimizers. Therefore, to implement RMSProp, Adam, and Nadam variants, their update rules and operations are changed accordingly. Algorithm 1 shows the implementation of the original AMSGrad version, which receives as inputs the latent factors vector $\overrightarrow{\theta}$, current gradient of the error $\overrightarrow{g}$, learning rate η, decay rates β_1 and β_2, padding factor to prevent divisions by zero ϵ, squared decaying average $\overrightarrow{v_{(t)}}$, decaying average $\overrightarrow{m_{(t)}}$, and maximum squared decaying average $\max(\overrightarrow{v_{(t)}})$ of the past gradients g. As output, the AMSGrad algorithm returns the updated latent factors vector $\overrightarrow{\theta}$. Algorithm 1 is responsible for inplace updating of the AMSGrad parameters based on the gradient of the error $\overrightarrow{g}$ and according to their rules in lines 1, 2, and 3 (described in Section 2.2). Finally, line 4 updates the latent factor vector $\overrightarrow{\theta}$ according to the AMSGrad update rules, and line 5 returns the updated vector.

Section 3.1 shows the characteristics of the specialized optimization strategy, in which we learn the personalized terms for each user or item in the data stream. In Section 3.2, we present specialized-generalized (SG) variants, which combine traditional and specialized optimizers to update the latent factor vector.

3.1. Specialized Optimizer

In the specialized version, we propose learning personalized optimizer parameters for each user u or item i to update MF models. In this sense, we update latent factor vector of user $\overrightarrow{A_u}$ and item $\overrightarrow{B_i}$ considering a specific optimizer term learned for the current user (item-specialized version) or item (item-specialized version) in the data stream.

Algorithm 2 presents a specialized version of the AMSGrad optimizer. The specialized AMSGrad optimizer has as inputs the latent factors vector $\overrightarrow{\theta}$, current gradient of the error $\overrightarrow{g}$, and personalization target p (which

Algorithm 1: Original AMSGrad Optimizer.

```
Input: $\overrightarrow{\theta}$: latent factors vector, $\overrightarrow{g}$: gradients of the error, $\eta$:
       learning rate, $\beta_1$ and $\beta_2$: decay rates, $\epsilon$: padding factor to
       prevent inconsistent computations, $\overrightarrow{v_{(t)}}$: squared decaying
       average, $\overrightarrow{m_{(t)}}$: decaying average, $\max(\overrightarrow{v_{(t)}})$: maximum squared
       decaying average.
Output: $\overrightarrow{\theta}$: updated latent factors vector.
1  $\overrightarrow{v_{(t)}} \leftarrow \beta_2 \times \overrightarrow{v_{(t-1)}} + (1-\beta_2) \times \overrightarrow{g}^2$
2  $\overrightarrow{m_{(t)}} \leftarrow \beta_1 \times \overrightarrow{m_{(t-1)}} + (1-\beta_1) \times \overrightarrow{g}$
3  $\max(\overrightarrow{v_{(t)}}) \leftarrow \max(\max(\overrightarrow{v_{(t)}}), \overrightarrow{v_{(t)}})$
4  $\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \left( \frac{\overrightarrow{m_{(t)}}}{\sqrt{\max(\overrightarrow{v_{(t)}})+\epsilon}} \right)$
5  return $\overrightarrow{\theta}$
```

indicates the identifiers of user u or item i, depending on the specialized variant selected). The specialized version also receives the learning rate η, decay rate β_1, decay rate β_2, padding factor ϵ (to prevent divisions by zero), and AMSGrad parameters, that is, specialized squared decaying average $\overrightarrow{v_{p(t)}}$, specialized decaying average $\overrightarrow{m_{p(t)}}$, and specialized maximum squared decaying average $\max(\overrightarrow{v_{p(t)}})$ of the past gradients g. Algorithm 2 then updates the AMSGrad personalized optimizer parameters $v_{p(t)}$ (Line 1), $m_{p(t)}$(Line 2), and `max`$(v_{p(t)})$ (Line 3) based on the gradients of the error $\overrightarrow{g}$. Finally, the latent factor vector $\overrightarrow{\theta}$ (line 4) is updated using the AMSGrad update rules (Equation 24) and the updated vector is returned (line 5).

If the user-specialized version is selected, both the latent factor vectors of user $\overrightarrow{A_u}$ and item $\overrightarrow{B_i}$ are updated using the personalized optimizer parameters learned for user u in the instance. Suppose we have a function called `update_model` that receives the specialized AMSGrad inputs, then user and item latent factors vectors are updated using Equation 27, where $\overrightarrow{v_{u(t)}}$, $\overrightarrow{m_{u(t)}}$, and $\max(\overrightarrow{v_{u(t)}})$ represents the specialized AMSGrad parameters learned for the user u.

$$\begin{aligned} \overrightarrow{A_u} &\leftarrow \texttt{update\_model}(A_u, g_u, u, \eta, \beta_1, \beta_2, \epsilon, \overrightarrow{v_{u(t)}}, \overrightarrow{m_{u(t)}}, \max(\overrightarrow{v_{u(t)}})) \\ \overrightarrow{B_i} &\leftarrow \texttt{update\_model}(B_i, g_i, u, \eta, \beta_1, \beta_2, \epsilon, \overrightarrow{v_{u(t)}}, \overrightarrow{m_{u(t)}}, \max(\overrightarrow{v_{u(t)}})) \end{aligned} \tag{27}$$

Algorithm 2: Incremental Specialized AMSGrad Optimizer

```
Input: $\overrightarrow{\theta}$: latent factors vector, $\overrightarrow{g}$: gradients of the error, $p$: user
       or item identifier based on specialized variant, $\eta$: learning
       rate, $\beta_1$ and $\beta_2$: decay rates, $\epsilon$: padding factor to prevent
       inconsistent computations, $\overrightarrow{v_{p(t)}}$: specialized squared decaying
       average, $\overrightarrow{m_{p(t)}}$: specialized decaying average, $\max(\overrightarrow{v_{p(t)}})$:
       specialized maximum squared decaying average.
Output: $\overrightarrow{\theta}$: updated latent factors vector.
1 $\overrightarrow{v_{p(t)}} \leftarrow \beta_2 \times \overrightarrow{v_{p(t-1)}} + (1-\beta_2) \times \overrightarrow{g}^2$
2 $\overrightarrow{m_{p(t)}} \leftarrow \beta_1 \times \overrightarrow{m_{p(t-1)}} + (1-\beta_1) \times \overrightarrow{g}$
3 $\max(\overrightarrow{v_{p(t)}}) \leftarrow \max(\max(\overrightarrow{v_{p(t)}}), \overrightarrow{v_{p(t)}})$
4 $\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \left( \frac{\overrightarrow{m_{p(t)}}}{\sqrt{\max(\overrightarrow{v_{p(t)}})+\epsilon}} \right)$
5 return $\overrightarrow{\theta}$
```

Otherwise, if an item-specialized version is selected, $\overrightarrow{A_u}$ and $\overrightarrow{B_i}$ are updated using the personalized optimizer parameters learned for the item in the instance using Equation 28, where $\overrightarrow{v_{i(t)}}$, $\overrightarrow{m_{i(t)}}$, and $\max(\overrightarrow{v_{i(t)}})$ represents the specialized AMSGrad parameters learned for the item i.

$$\begin{aligned} \overrightarrow{A_u} &\leftarrow \texttt{update\_model}(A_u, g_u, i, \eta, \beta_1, \beta_2, \epsilon, \overrightarrow{v_{i(t)}}, \overrightarrow{m_{i(t)}}, \max(\overrightarrow{v_{i(t)}})) \\ \overrightarrow{B_i} &\leftarrow \texttt{update\_model}(B_i, g_i, i, \eta, \beta_1, \beta_2, \epsilon, \overrightarrow{v_{i(t)}}, \overrightarrow{m_{i(t)}}, \max(\overrightarrow{v_{i(t)}})) \end{aligned} \tag{28}$$

Considering that we are analyzing the RMSProp, Adam, AMSGrad, and Nadam optimizers, we have two specializations for each version. As our focus is on incremental scenario, we named the variants with the prefix "In" and used the user or item suffix to represent the specialized version. In this sense, the traditional variants are the InRMSProp, InAdam, InAMSGrad, and InNadam optimizers. The user-specialized variants assume the "User" suffix with InRMSPropUser, InAdamUser, InAMSGradUser, and InNadamUser. Finally, the item-specialized variants, which have "Item" suffix, are InRMSPropItem, InAdamItem, InAMSGradItem, and InNadamItem. Table 1 presents the update rules of Adam, AMSGrad, Nadam, and RMSProp optimizer, where p indicates the specialization target, which assumes the user or item identifiers during the model updates.

3.2. Specialized-Generalized Optimizer

The proposed specialized-generalized variant combines the original optimizer version with a specialized version. Accordingly, it is necessary to learn both optimizer parameters instead of learning only the general or specialized versions. We combine the optimizer's general and specialized variants by joining their respective update rules. Therefore, we use the sum of the particular parameters of both variants to generate the specialized-generalized version. Equation 29 shows the specialized-generalized optimizer update rule in which $\overrightarrow{\theta}$ is the user $\overrightarrow{A_u}$ or item $\overrightarrow{B_i}$ latent factor vector, η is step size, and P_o and P_s are parameters that represent the original and specialized optimizers, respectively. The combination of these parameters enables the gradient to store both the local and global latent factor changes.

$$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \left(P_o + P_s\right) \tag{29}$$

Algorithm 3 presents the AMSGrad specialized-generalized optimizer variant. The input arguments ($\overrightarrow{\theta}, \overrightarrow{g}, p, \eta, \beta_1, \beta_2, \epsilon$) and the output ($\overrightarrow{\theta}$) are the same as those of the original and specialized versions. However, the specialized-generalized variant stores both general and specialized squared decaying averages ($\overrightarrow{v_{(t)}}$ and $\overrightarrow{v_{p(t)}}$), decaying averages ($\overrightarrow{m_{(t)}}$ and $\overrightarrow{m_{p(t)}}$), and maximum squared decaying averages ($\texttt{max}(\overrightarrow{v_{(t)}})$ and $\texttt{max}(\overrightarrow{v_{p(t)}})$). However, we must update both the general (Algorithm 3 - lines 1, 2, and 3) and specialized (Algorithm 3 - lines 4, 5, and 6) parameters according to AMSGrad update rule. Finally, to update the latent factor vector $\overrightarrow{\theta}$, the specialized-generalized variant combines general and specialized AMSGrad update rules (line 7), in which the main difference is between the specialized and specialized-generalized versions.

To distinguish the optimizers' variants, we named the specialized-generalized versions with the "SGI" prefix and the "User" or "Item" suffix to represent the specialized part of the optimizer. The user-specialized-generalized variants are SGIRMSPropUser, SGIAdamUser, SGIAMSGradUser, and SGINadamUser while the item-specialized variants are SGIRMSPropItem, SGIAdamItem, SGIAMSGradItem, and SGINadamItem. Algorithms 1, 2, and 3 present the execution flow of the variants of the AMSGrad optimizer. Accordingly, if we want to select other optimizers, we have to use their update rule presented in Table 1, which shows the latent factor vector $\overrightarrow{\theta}$ update for the specialized and specialized-generalized variants of the Adam, AMSGrad, Nadam, and

Algorithm 3: Incremental Specialized-Generalized AMSGrad Optimizer.

```
Input: $\overrightarrow{\theta}$: latent factors vector, $\overrightarrow{g}$: gradients of the error, $p$: user
       or item identifier based on specialized variant, $\eta$: learning
       rate, $\beta_1$ and $\beta_2$: decay rates, $\epsilon$: padding factor to prevent
       inconsistent computations, $\overrightarrow{v_{(t)}}$: squared decaying average,
       $\overrightarrow{m_{(t)}}$: decaying average, $\max(\overrightarrow{v_{(t)}})$: maximum squared
       decaying average $\overrightarrow{v_{p(t)}}$: specialized squared decaying average,
       $\overrightarrow{m_{p(t)}}$: specialized decaying average, $\max(\overrightarrow{v_{p(t)}})$: specialized
       maximum squared decaying average.
Output: $\overrightarrow{\theta}$: updated latent factors vector
1  $\overrightarrow{v_{(t)}} \leftarrow \beta_2 \times \overrightarrow{v_{(t-1)}} + (1-\beta_2) \times \overrightarrow{g}^2$
2  $\overrightarrow{m_{(t)}} \leftarrow \beta_1 \times \overrightarrow{m_{(t-1)}} + (1-\beta_1) \times \overrightarrow{g}$
3  $\max(\overrightarrow{v_{(t)}}) \leftarrow \max(\max(\overrightarrow{v_{(t)}}), \overrightarrow{v_{(t)}})$
4  $\overrightarrow{v_{p(t)}} \leftarrow \beta_2 \times \overrightarrow{v_{p(t-1)}} + (1-\beta_2) \times \overrightarrow{g}^2$
5  $\overrightarrow{m_{p(t)}} \leftarrow \beta_1 \times \overrightarrow{m_{p(t-1)}} + (1-\beta_1) \times \overrightarrow{g}$
6  $\max(\overrightarrow{v_{p(t)}}) \leftarrow \max(\max(\overrightarrow{v_{p(t)}}), \overrightarrow{v_{p(t)}})$
7  $\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \left( \frac{\overrightarrow{m_{(t)}}}{\sqrt{\max(\overrightarrow{v_{(t)}})+\epsilon}} + \frac{\overrightarrow{m_{p(t)}}}{\sqrt{\max(\overrightarrow{v_{p(t)}})+\epsilon}} \right)$
8  return $\overrightarrow{\theta}$
```

RMSProp optimizers. Additional operations may be necessary based on each optimizer step, which are presented in Section 2.2.

Table 1: Optimizers update rules based on specialized and specialized-generalized versions.

Specialized	Specialized-Generalized
Adam	
$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \frac{\overrightarrow{\widehat{m}_{p(t)}}}{\sqrt{\overrightarrow{\widehat{v}_{p(t)}}} + \epsilon}$	$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \left(\frac{\overrightarrow{\widehat{m}_{(t)}}}{\sqrt{\overrightarrow{\widehat{v}_{(t)}}} + \epsilon} + \frac{\overrightarrow{\widehat{m}_{p(t)}}}{\sqrt{\overrightarrow{\widehat{v}_{p(t)}}} + \epsilon} \right)$
AMSGrad	
$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \times \frac{\overrightarrow{m_{p(t)}}}{\sqrt{\max(\overrightarrow{v_{p(t)}})} + \epsilon}$	$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \times \left(\frac{\overrightarrow{m_{(t)}}}{\sqrt{\max(\overrightarrow{v_{(t)}})} + \epsilon} + \frac{\overrightarrow{m_{p(t)}}}{\sqrt{\max(\overrightarrow{v_{p(t)}})} + \epsilon} \right)$
Nadam	
$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \times \frac{\overrightarrow{\widehat{m}_{pa(t)}}}{\sqrt{\overrightarrow{\widehat{v}_{p(t)}}} + \epsilon}$	$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \times \left(\frac{\overrightarrow{\widehat{m}_{a(t)}}}{\sqrt{\overrightarrow{\widehat{v}_{(t)}}} + \epsilon} + \frac{\overrightarrow{\widehat{m}_{pa(t)}}}{\sqrt{\overrightarrow{\widehat{v}_{p(t)}}} + \epsilon} \right)$
RMSProp	
$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \overrightarrow{g} \frac{\eta}{\sqrt{v_{p(t)}} + \epsilon}$	$\overrightarrow{\theta} \leftarrow \overrightarrow{\theta} + \eta \times \overrightarrow{g} \times \left(\frac{1}{\sqrt{\overrightarrow{v_{(t)}}} + \epsilon} + \frac{1}{\sqrt{\overrightarrow{v_{p(t)}}} + \epsilon} \right)$

4. Experimental Setup

This section reports the experimental protocol adopted for assessing the proposed methods.

4.1. Datasets

We selected four real-world datasets for experimentation: Amazon Books [37], Movie Lens 1M [38], Movie Tweetings [39], and SMDI-200UE [6]. The Amazon Books, Movie Lens 1M, and Movie Tweetings datasets provide explicit user feedback. To analyze the effectiveness of our proposed models in the one-class collaborative filtering scenario, we transformed the explicit feedback into positive-only feedback. We considered positive feedback only for ratings higher than 3.5. Table 2 depicts the main characteristics of these datasets, including the number of interactions, users, items, and sparsity (Equation 30), where $|D|$ is the number of unique interactions, m the number of users, and n the number of items.

$$\text{Sparsity (\%)} = 100 \times \frac{1 - |D|}{m \times n} \tag{30}$$

Table 2: Overview of the datasets used on this study.

Datasets	**Interactions**	**Users**	**Items**	**Sparsity**
Amazon Books	769991	21675	22223	99.84%
Movie Lens 1M	575280	6038	3533	97.30%
Movie Tweetings	658700	65513	28149	99.96%
SMDI-200UE	447391	9472	6924	99.59%

4.2. Algorithms and Parametrization

Since we are dealing with a streaming scenario in which the order of the events is significant, we split our datasets according to the temporally-aware protocol suggested in [40]. Figure 1 presents the training and update processes of the recommender model used in this study to evaluate the proposed optimizers. We used the first 20% of each dataset for batch training (Recommender Model Training - Figure 1), 30% for batch testing and incremental training (Performance Validation - Figure 1), and the remaining 50% as the test set (Incremental Update Phase - Figure 1). The test set is used in a test-then-train fashion, meaning that each user-item interaction is used for model evaluation (Evaluate Model and Compute Statistics - Figure 1) and updating sequentially (Update Model - Figure 1).

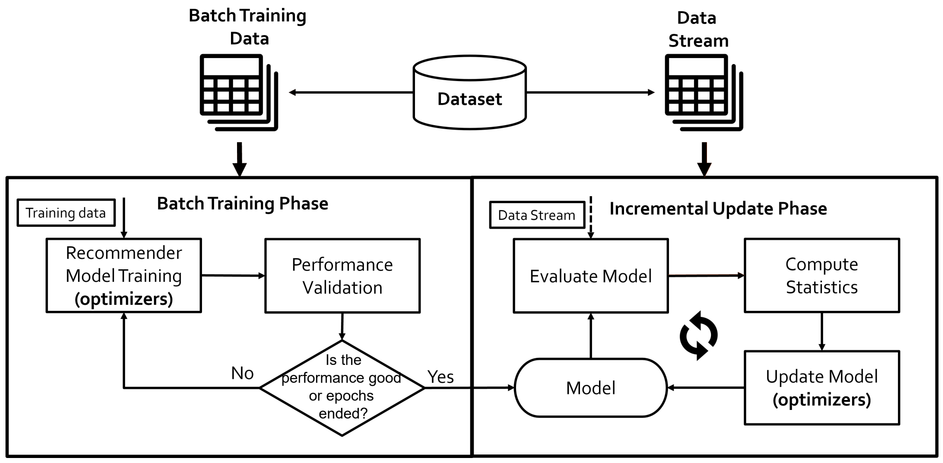


Figure 1: Streaming protocol used in our experimentation.

We compare our proposed MF optimization strategies with and against the SGD [22] and InMLF [14] algorithms, as previously introduced in Section 2. We also used PMF [27] and BPRMF [16] recommender models as baselines. Additionally, we tested the non-machine learning methods *Top Popular* (TopPop) [41] and random [42] for comparison. TopPop consists of recommending items with the best degree of success among all users, instead of modeling the user item relationship using a machine learning method. In contrast, the random approach randomly selects items from a set of unobserved items in the recommendation phase. The incremental MF recommendation model uses the SGD optimizer in its original version, which is the most frequently used method for minimizing the L2-regularized squared error (Equation 2). Therefore, we compared the proposed adaptive optimizer variants as replacements for the traditional SGD optimizer.

We tested the following hyper-parameter values in the MF incremental recommender model: optimizer $\in$ {`SGD`, `InMLF`, `InAdam`, `InAdamUser`, `InAdamItem`, `SGIAdamUser`, `SGIAdamItem`, `InAMSGrad`, `InAMSGradUser`, `InAMSGradItem`, `SGIAMSGradUser`, `SGIAMSGradItem`, `InNadam`, `InNadamUser`, `InNadamItem`, `SGINadamUser`, `SGINadamItem`, `InRMSprop`, `InRMSpropUser`, `InRMSpropItem`, `SGIRMSpropUser`, and `SGIRMSpropItem`}, learning rate $\in \{0.01, 0.001, 0.0001\}$, regularization rate equal to 0.01, latent factors $\in \{10, 20, 40, 60, 80\}$, batch training epochs equal to 10. We also used these hyper-parameters, except for the optimizer, in the PMF and BPRMF recommendation models.

We chose the best hyper-parameters based on a grid search, which exhaustively generates candidates from the specified grid of parameter values. The entire dataset was used for each combination. In this sense, parameter tuning is not a part of the processing time of the methods. We replicated each best experimental setting 10 times in this study, thus, the results depict the average and standard deviation of the recall values. We selected a single random seed for each replication and used it with all optimizers to enable further paired comparisons and application of the statistical significance test.

Finally, we incorporated hypothesis testing to determine whether one optimization significantly outperforms others method. As we have many optimization strategies for comparison, we execute each experiment several times if we envision finding any statistically significant difference. Therefore, we selected each optimizer's best variant and applied non-parametric tests. We followed the protocol reported in [43] by combining the Friedman [44] and Nemenyi post-hoc [45] statistical tests.

4.3. Evaluation Metrics

We express the goodness-of-fit of the models using `RECALL@K` and normalized discounted cumulative gain (`NDCG@K`) metrics with $K = 10$, which are the most commonly used recall thresholds. For each instance (u, i) in the test set (T), we selected a candidate list of 1000 unknown items for user u, and the known (relevant) item i was appended to this candidate list. We ordered candidate items by descending proximity to a value of 1 (as we are dealing with a positive-only scenario) using the function $f_{ui} = |1 - \widehat{R}_{ui}|$, according to the non-Boolean predicted scores $\widehat{R}_{ui}$ obtained by the recommender models.

The `RECALL@K` metric, described in Equation 31, measures the average (across all users) of the proportion of recommended items that appear among the top K positions of the ranked list [46], where $|T|$ is the test set size.

$$\text{RECALL@K} = \frac{1}{|T|} \sum_{(u,i)\in T} \text{hit@K}(u, i) \tag{31}$$

For each instance $\langle u, i\rangle$, $\text{hit@K}(u, i) = 1$ is said to happen when i is ranked among the top K items, and $\text{hit@K}(u, i) = 0$ otherwise.

The `NDCG@K` metric, described in Equation 32, indicates whether the ground-truth items are ranked higher than others by accounting for the position of hit [47].

$$\text{NDCG@K} = \begin{cases} 0, & \text{if rank(i)} \leq \text{K} \\ \frac{1}{log_2(rank+1)}, & \text{otherwise} \end{cases} \tag{32}$$

The scores reported in the following section for `RECALL@K` and `NDCG@K` were obtained in the test portion of the experiments. As we select the best set of parameters, which include the number of latent features, we do not consider the processing time of the recommender models in this study. The greater the number of latent factors f, the longer is the processing time of the recommender models. Furthermore, the proposed specialized and specialized-generalized optimizer variants are more time and memory consuming than the traditional SGD optimizer because of the storage of the parameters for all users or items of the system.

5. Results and Analysis

In this section, we present results of experiments using four real-world datasets. The results obtained by incremental MF models are presented in Tables 3 and 4. These tables depict the RECALL@K and NDCG@K (with $K = 10$) obtained by each optimizer along with the analyzed datasets. We compared our proposed optimizers against the MF-SGD, MF-InMLF, BPRMF, PMF, TopPop, and Random baselines and marked the best results per dataset in bold. We report the statistical test results obtained by combining the Friedman and Nemenyi tests, assuming a 95% confidence level throughout our discussion. The p-values for each tested dataset are shown in Figure 3. We also observed significant differences in computing the critical distance between the analyzed optimizers and baselines (Figure 3).

Regarding the baseline methods (Tables 3 and 4), the MF with InMLF [14] optimizer provide the best RECALL@10 and NDCG@10 values in the Amazon Books dataset. Considering the Movie Lens 1M dataset, the MF-SGD and BPRMF models provided the best baseline results. By contrast, for the Movie Tweetings dataset, the MF-SGD model obtained the highest RECALL values. Finally, in the supermarket dataset (SMDI-200UE), the PMF model showed the best performance.

Tables 3 and 4 summarize the performances of the different optimizers analyzed in terms of the RECALL@10 and NDCG@10 values. The In-

Table 3: RECALL@10 and NDCG@10 values obtained by the incremental MF recommender model for the Amazon Books and Movie Lens 1M datasets.

Models	**Amazon Books**		**Movie Lens 1M**	
	RECALL@10	**NDCG@10**	**RECALL@10**	**NDCG@10**
Baselines				
TopPop	0.0040 ± 0.0000	0.0020 ± 0.0000	0.0390 ± 0.0000	0.0190 ± 0.0000
Random	0.0010 ± 0.0000	0.0000 ± 0.0000	0.0030 ± 0.0002	0.0010 ± 0.0001
MF-SGD	0.0130 ± 0.0001	0.0060 ± 0.0001	0.1200 ± 0.0003	0.0600 ± 0.0001
MF-InMLF	0.0180 ± 0.0002	0.0090 ± 0.0001	0.1070 ± 0.0003	0.0520 ± 0.0002
BPRMF	0.0150 ± 0.0003	0.0080 ± 0.0002	0.1140 ± 0.0008	0.056 ± 0.0004
PMF	0.0050 ± 0.0001	0.0020 ± 0.0001	0.1210 ± 0.0050	0.0600 ± 0.0028
MF with Adam Optimizers variants				
InAdam	0.0080 ± 0.0001	0.0040 ± 0.0001	0.1300 ± 0.0002	0.065 ± 0.0001
InAdamUser	0.0210 ± 0.0025	0.0100 ± 0.0012	0.1310 ± 0.0002	0.066 ± 0.0001
InAdamItem	0.0210 ± 0.0005	0.0100 ± 0.0002	0.0870 ± 0.0012	0.045 ± 0.0006
SGIAdamUser	0.0160 ± 0.0015	0.0080 ± 0.0007	0.1250 ± 0.0003	0.063 ± 0.0002
SGIAdamItem	0.0190 ± 0.0004	0.0090 ± 0.0002	0.1060 ± 0.0017	0.053 ± 0.0009
MF with AMSGrad Optimizers variants				
InAMSGrad	0.0050 ± 0.0002	0.0020 ± 0.0001	0.1310 ± 0.0003	0.0660 ± 0.0002
InAMSGradUser	**0.0360 ± 0.0040**	**0.0170 ± 0.0020**	**0.1320 ± 0.0003**	**0.0660 ± 0.0002**
InAMSGradItem	0.0110 ± 0.0005	0.0050 ± 0.0002	0.0850 ± 0.0013	0.0430 ± 0.0007
SGIAMSGradUser	0.0270 ± 0.0045	0.0130 ± 0.0023	**0.1320 ± 0.0003**	**0.0660 ± 0.0001**
SGIAMSGradItem	0.0110 ± 0.0005	0.0050 ± 0.0002	0.1020 ± 0.0012	0.0510 ± 0.0006
MF with Nadam Optimizers variants				
InNadam	0.0080 ± 0.0001	0.0040 ± 0.0001	0.1310 ± 0.0002	0.0660 ± 0.0001
InNadamUser	0.0190 ± 0.0020	0.0090 ± 0.0009	0.1310 ± 0.0003	0.0660 ± 0.0002
InNadamItem	0.0210 ± 0.0006	0.0100 ± 0.0003	0.0910 ± 0.0013	0.0460 ± 0.0006
SGINadamUser	0.0140 ± 0.0009	0.0060 ± 0.0005	0.1300 ± 0.0004	0.0650 ± 0.0003
SGINadamItem	0.0180 ± 0.0004	0.0090 ± 0.0002	0.1140 ± 0.0024	0.0570 ± 0.0012
MF with RMSProp Optimizers variants				
InRMSProp	0.0170 ± 0.0004	0.0080 ± 0.0002	0.1260 ± 0.0004	0.0630 ± 0.0002
InRMSPropUser	0.0170 ± 0.0008	0.0080 ± 0.0004	0.1280 ± 0.0003	0.0640 ± 0.0002
InRMSPropItem	0.0160 ± 0.0016	0.0080 ± 0.0007	0.1260 ± 0.0002	0.0630 ± 0.0001
SGIRMSPropUser	0.0090 ± 0.0006	0.0040 ± 0.0003	0.1300 ± 0.0003	0.0650 ± 0.0001
SGIRMSPropItem	0.0080 ± 0.0003	0.0040 ± 0.0001	0.1290 ± 0.0005	0.0640 ± 0.0002

AMSGradUser optimizer obtained the best results across all tested datasets. Considering the RECALL@10 values of InAMSGradUser, the adaptive optimizer increases by up to 2.3 percentage points in the Amazon Books, 1.2 percentage points for the Movie Lens 1M (Table 3), 8.2 percentage points for the Movie Tweetings, and 11.1 percentage points for the SMDI-200UE dataset (Table 4), compared with the traditional SGD baseline.

Furthermore, compared with adaptive InMLF optimization, the InAMSGradUser optimizer provided an increase by up to 1.8, 2.5, 13.1, and 14.7 percentage points for the Amazon Books, Movie Lens 1M, Movie Tweetings, and SMDI-200UE datasets, respectively. Concerning the NDCG@10 metric, the results showed a lower increase, which is natural because the NDCG

Table 4: RECALL@10 and NDCG@10 values obtained by the incremental MF recommender model for the Movie Tweetings and SMDI-200UE datasets.

Models	**Movie Tweetings**		**SMDI-200UE**	
	RECALL@10	**NDCG@10**	**RECALL@10**	**NDCG@10**
Baselines				
TopPop	0.0070 ± 0.0000	0.0030 ± 0.0000	0.1500 ± 0.0000	0.0870 ± 0.0000
Random	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0020 ± 0.0001	0.0010 ± 0.0000
SGD	0.1510 ± 0.0032	0.0630 ± 0.0014	0.2290 ± 0.0004	0.1330 ± 0.0002
InMLF	0.1010 ± 0.0020	0.0430 ± 0.0008	0.1930 ± 0.0023	0.1080 ± 0.0012
BPRMF	0.0580 ± 0.0004	0.0280 ± 0.0003	0.2380 ± 0.0013	0.1380 ± 0.0008
PMF	0.0090 ± 0.0015	0.0050 ± 0.0009	0.3390 ± 0.0058	**0.2250 ± 0.0030**
MF with Adam Optimizers variants				
InAdam	0.1660 ± 0.0013	0.074 ± 0.0006	0.3210 ± 0.0004	0.2030 ± 0.0003
InAdamUser	0.2250 ± 0.0045	0.1110 ± 0.0027	0.3370 ± 0.0004	0.2010 ± 0.0017
InAdamItem	0.2070 ± 0.0016	0.0990 ± 0.0012	0.1950 ± 0.0026	0.1210 ± 0.0021
SGIAdamUser	0.2060 ± 0.0064	0.1010 ± 0.0034	0.3290 ± 0.0011	0.1840 ± 0.0014
SGIAdamItem	0.2120 ± 0.0010	0.1000 ± 0.0007	0.2100 ± 0.0042	0.1290 ± 0.0023
MF with AMSGrad Optimizers variants				
InAMSGrad	0.1510 ± 0.0029	0.0720 ± 0.0014	0.3140 ± 0.0011	0.1820 ± 0.0011
InAMSGradUser	**0.2320 ± 0.0028**	**0.1150 ± 0.0017**	**0.3400 ± 0.0005**	0.2080 ± 0.0007
InAMSGradItem	0.2090 ± 0.0023	0.1 ± 0.0013	0.2350 ± 0.0039	0.1560 ± 0.0028
SGIAMSGradUser	0.2090 ± 0.0025	0.102 ± 0.0013	0.3380 ± 0.0003	0.2010 ± 0.0003
SGIAMSGradItem	0.2090 ± 0.0011	0.101 ± 0.0007	0.2550 ± 0.0030	0.1660 ± 0.0025
MF with Nadam Optimizers variants				
InNadam	0.1570 ± 0.0004	0.0700 ± 0.0002	0.3260 ± 0.0006	0.1970 ± 0.0004
InNadamUser	0.2280 ± 0.0033	0.1120 ± 0.0018	**0.3400 ± 0.0006**	0.2010 ± 0.0018
InNadamItem	0.2190 ± 0.0012	0.1050 ± 0.0009	0.2050 ± 0.0051	0.1280 ± 0.0029
SGINadamUser	0.2040 ± 0.0048	0.0990 ± 0.0026	0.3280 ± 0.0015	0.1820 ± 0.0014
SGINadamItem	0.2220 ± 0.0012	0.1060 ± 0.0009	0.2240 ± 0.0073	0.1360 ± 0.0043
MF with RMSProp Optimizers variants				
InRMSProp	0.1780 ± 0.0024	0.0910 ± 0.0015	0.1960 ± 0.0004	0.1170 ± 0.0004
InRMSPropUser	0.1770 ± 0.0021	0.0930 ± 0.0015	0.1940 ± 0.0010	0.1190 ± 0.0005
InRMSPropItem	0.1710 ± 0.0043	0.0880 ± 0.0023	0.2030 ± 0.0032	0.1220 ± 0.0016
SGIRMSPropUser	0.1880 ± 0.0026	0.0960 ± 0.0014	0.3130 ± 0.0013	0.1610 ± 0.0016
SGIRMSPropItem	0.1650 ± 0.0012	0.0850 ± 0.0007	0.3120 ± 0.0014	0.1640 ± 0.0015

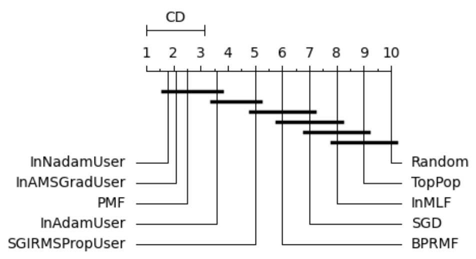
evaluates the item ranking in the TOP@10 ranked list. Regarding the InAMSGradUser and SGD NDCG@10 values, we observed increases of up to 1.1, 0.6, 5.2, and 7.5 percentage points for the Amazon Books, Movie Lens 1M, Movie Tweetings, and SMDI-200UE datasets, respectively. However, although the increase in NDCG values was smaller than that in RECALL, there were also significant differences between the results.

The results showed that the PMF model provided competitive results for the Movie Lens and SMDI-200UE datasets. Moreover, considering the Amazon Books and Movie Tweeting datasets, which are the largest amog the tested datasets, the results are not significantly different from the TopPop baseline. As observed in the incremental MF model results (Tables 3 and 4), considering the optimizer variants, the user-specialized version provided superior performance in most cases. In all the analyzed datasets, the InAdamUser, InAMSGradUser, and InNadamUser variants showed the best results compared to the other variants of each optimizer. For the RMSProp optimizer, the user specialized-generalized version presented the best results. In contrast, in most experiments item-specialized and item-specialized-generalized variants did not significantly differ from the SGD baseline. We explain this behavior by searching for datasets of distinct users and items, where the number of users is higher than the number of items. Additionally, as we are working in a collaborative filtering scenario, where the primary purpose is to model and learn the relationship between users, it makes sense to learn individualized optimizer parameters for each user, because the optimizer also uses these parameters to update the latent factor vector of the items.

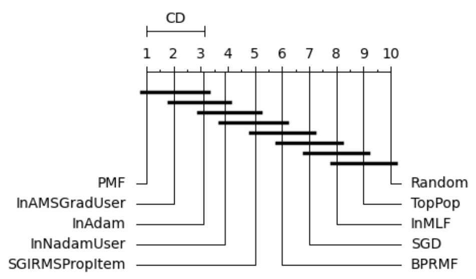
Considering the statistical significance test, although the InAMSGradUser rates were greater, some optimizers did not show significant differences considering the Nemenyi test with a 95% confidence level. In the Movie Lens 1M dataset, the InAMSGradUser optimizer did not show a significant difference from the SGIAMSGradUser, InNadamUser, InAdamUser, InAMSGrad, InNadam, and InAdam optimizers. Considering the Movie Tweetings dataset, the optimizers without significant differences from InAMSGradUser are InNadamUser, InAdamUser, SGINadamItem, InNadamItem, and SGINadamItem. Finally, in the SMDI-200UE dataset, the InAMSGradUser optimizer does not differ from InNadamUser, SGIAMSGradUser, InAdamUser, SGIAdamUser, SGINadamUser, and InNadam.

Figure 2 shows the critical distance (CD) of the best variants of each optimizer, considering the obtained results for all datasets. The general results concerning the RECALL@10 values demonstrate the best performance of user-specialized variants. In addition, by comparing the RECALL@10 values across all the analyzed datasets, we confirm that the AMSGradUser and InNadamUser optimizers presented the best performance in the experiments, showing significant differences from the other optimizers, except for the PMF model in the SMDI-200UE dataset, which presented competitive results. Furthermore, considering the NDCG@10 values, the PMF model outperformed the other optimizers. On the other hand, in contrast to what was observed in the RECALL@10 analysis, regarding the NDCG@10 values, the InAdam and SGIRMSPropItem optimizers outperform the InAdamUser and SGIRMSPropIUser variants.

Figure 3 presents the results of Friedman and Nemenyi's statistical signif-



(a) RECALL@10.



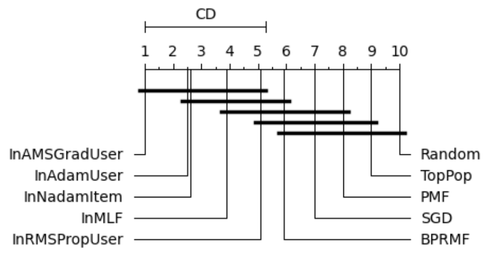
(b) NDCG@10.

Figure 2: Nemenyi test results on RECALL@10 and NDCG@10 values in the analysed datasets.

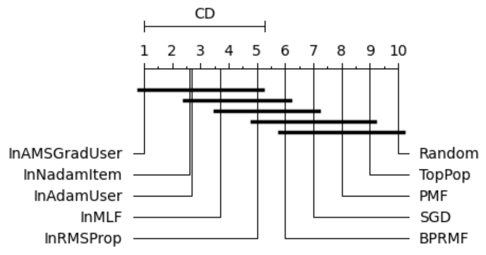
icance test, considering the experiments of all techniques in each analyzed dataset. The legends present the Friedman p-values and show significant differences, assuming a 95% (p-value < 0.05) confidence level for the results for all datasets. Therefore, the graph of the critical distance obtained by the Nemenyi test shows the rankings and significant differences of each method. We observe in the statistical significance tests in Figure 3 that the behavior of the results is similar to that presented in Tables 3 and 4. The user-specialized variants provided the best results in most cases and outperformed the specialized-generalized versions of the AMSGrad, Nadam, and Adam optimizers. InAMSGradUser and InAdamUser proposed specialized optimizer variants provide the best results for all analyzed datasets considering RECALL@10. Regarding the NDCG@10 values, only for the SMDI-200UE dataset, InAdamUser does not appear as the best variant of the Adam Optimizer. However, considering the RMSProp optimizer, we obtained the best results in the user-specialized-generalized version (SGIRMSPropUser). We also observe different behaviors in the Amazon Books dataset results, in which the item-specialized Nadam optimizer variant (InNadamItem) presented the best performance compared to the other versions.

In contrast to the results obtained in the combined statistical analysis of all datasets (Figure 2), considering the results in every single dataset (3), the PMF model presented the best results only for the SMDI-200UE dataset. However, in the general analysis, the PMF obtained the best performance because the RECALL@10 and NDCG@10 values in the SMDI-200UE dataset were higher than those of the other datasets.

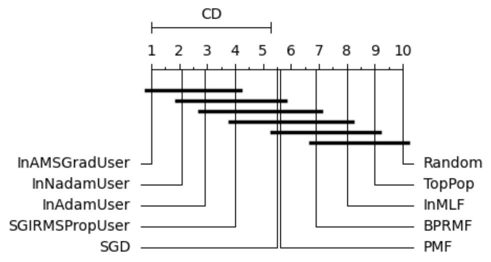
Comparing the obtained results across the analyzed datasets, we observed



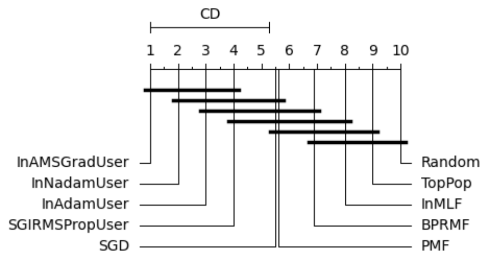
(a) Amazon Books RECALL@10 (p-value = 2.64×10^{-15}).



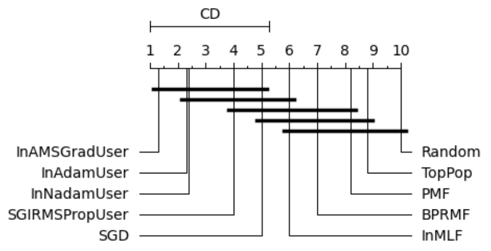
(b) Amazon Books NDCG@10 (p-value = 3.07×10^{-15}).



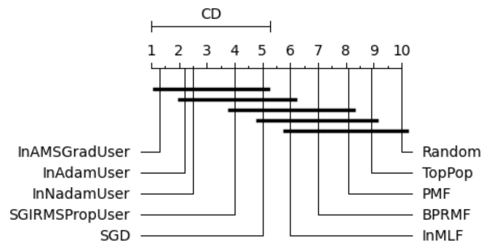
(c) Movie Lens RECALL@10 (p-value = 2.64×10^{-15}).



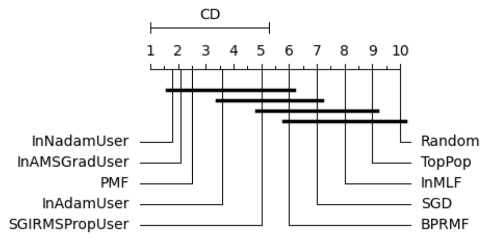
(d) Movie Lens NDCG@10 (p-value = 2.41×10^{-15}).



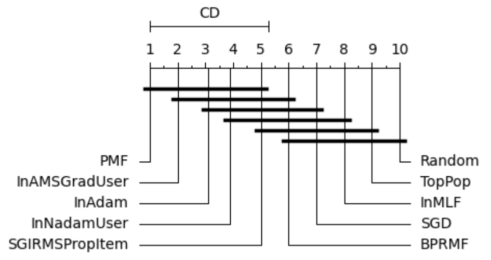
(e) Movie Tweetings RECALL@10 (p-value = 3.60×10^{-15}).



(f) Movie Tweetings NDCG@10 (p-value = 3.29×10^{-15}).



(g) SMDI-200UE RECALL@10 (p-value = 7.9×10^{-15}).



(h) SMDI-200UE NDCG@10 (p-value = 1.78×10^{-15}).

Figure 3: Critical distances of the Nemenyi test for Recall and NDCG results in different datasets. All p-values refer to the Friedman test.

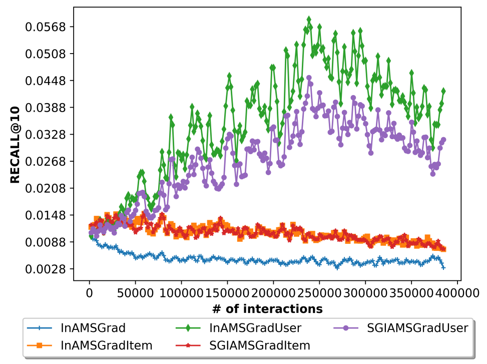
that the proposed user-specialized and user-specialized-generalized variants provided superior performance to the tested baselines in most cases. Additionally, combining the adaptive learning rate optimizers in the MF model, Adam, Nadam, AMSGrad, and RMSProp significantly increased the RECALL and NDGC rates compared with the traditional SGD baseline.

5.1. Streaming Analysis of the Results

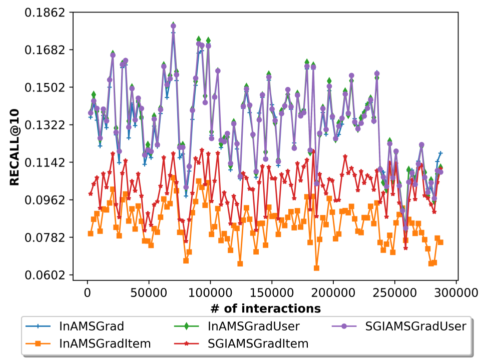
As we are working in a streaming scenario, it is essential to evaluate the tested methods by considering the evolution of the results according to the emergence of new instances ($<u, i>$) in the stream. Figure 4 presents the windowed evaluation of RECALL@10 values by analyzing the best optimizer and their variants in each dataset. We considered a window with a size 5% of the number of test interactions. Considering the Amazon Books dataset results (Figure 4a), we observe that the AMSGradUser and SGIAMSGradUser optimizers outperform other AMSGrad variants during the entire streaming test set. However, the specialized version AMSGradUser maintains the best windowed RECALL@10 values during the data stream fashion. The item-specialized (InAMSGradItem) and item-specialized-generalized (SGIAMSGradItem) variants showed similar results and presented superior results to the traditional InAMSGrad optimization strategy.

Concerning the results obtained in the Movie Lens dataset experiments (Figure 4b), we confirm the reduced differences in the RECALL@10 values compared to the windowed evaluation results with the obtained average RECALL@10 rates (Table 3). The both traditional (InAMSgrad), user-specialized (InAMSgradUser), and user-specialized-generalized (SGIAMS-gradUser) variants of the AMSGrad optimizer show similar behavior in the plotted results. Different from what we observe in the Amazon Books dataset, in the Movie Lens dataset, the item-specialized (InAMSGradItem) and item-specialized-generalized (SGIAMSGradItem) variants of AMSGrad optimizer presented inferior results than the traditional InAMSGrad version.

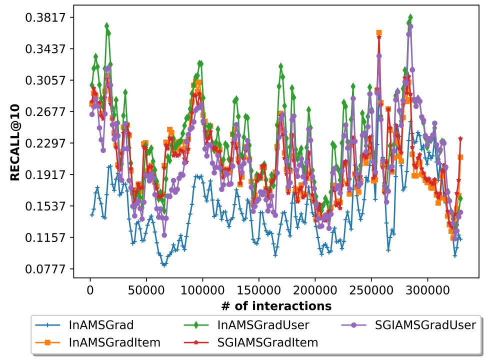
Figure 4c shows the windowed RECALL@10 values obtained using the AMSGrad optimizer variants in the movie tweeting dataset. This analysis shows that the proposed AMSGrad variants outperform the traditional optimizer. We observed that the user-specialized optimizer (InAMSGradUser) provided superior performance in some regions of the graph, as proven by the average RECALL@10 rates (Table 4), in which we obtained a RECALL@10 value of 23.20% using the InAMSGradUser optimizer. For InAMSGradItem,



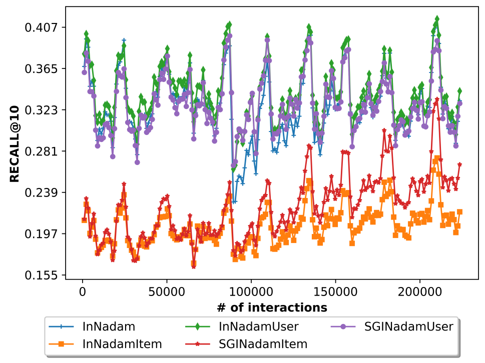
(a) AMSGrad variants in the Amazon Books dataset.



(b) AMSGrad variants in the Movie Lens dataset.



(c) AMSGrad variants in the Movie Tweetings dataset.



(d) Nadam variants in the SMDI-200UE dataset.

Figure 4: Windowed evaluation of RECALL@10 values obtained by the best optimizer and their variants in each dataset (We consider a window size of 5% of the number of interactions for each dataset test portion).

SGIAMSGradUser, and SGIAMSGradUser optimizers, the obtained RECALL@10 value was 20.90%.

Finally, considering the SMDI-200UE dataset results, Figure 4d shows the results for the Nadam optimizer variants. We observe a similar behavior from the obtained results in the Movie Lens dataset, in which the traditional (InNadam), user-specialized (InNadamUser), and user-specialized-generalized (SGINadamUser) variants presented similar results. Furthermore, the item specialized (InNadamItem) and item-specialized-generalized (SGINadamItem) versions also provided worse results than the traditional version. Nonetheless, we observe that although the windowed evaluation presented similar results, InNadamUser and SGINadamUser outperformed the InNadam optimizer in some regions of the test data stream.

We justify the obtained results in the analyzed datasets by comparing the size (number of instances) and number of distinct users and items. The Movie Lens and SMDI-200UE datasets had fewer users and items than the Amazon books and movie tweeting datasets. In this sense, our proposed optimizer variants provided the best performance in datasets with a higher number of users and items, which is reasonable because we trained individual optimizer parameters for each available user or item in the stream. The Amazon books and movie tweeting datasets also have more instances than the others, showing that our methods work well in large datasets with a high incidence of new users and items.

6. Conclusion

In this study, we proposed novel adaptive learning rate optimizers for incremental MF recommender systems. Our variants consider user (user-specialized and user-specialized-generalized) or item (item-specialized and item-specialized-generalized) identifiers to model and learn optimizer parameters for each user or item, respectively. We selected four adaptive optimizers, Adam, Nadam, AMSGrad, and RMSProp, and coupled them with the proposed optimizers. We tested our approaches on three real-world datasets and compared their results with those of a traditional SGD optimizer. Our proposed optimizers presented superior performance when datasets have many users and items. The larger the number of users and items, the higher the incidence of cold-start. In this sense, the specialization is more suitable when fewer interactions per user exist and more users for which we do not have information emerge from the stream. We observed that the InAMSGradUser and InNadamUser variants, which are user-specialized versions of the AMSGrad and Nadam optimizers, significantly outperform the SGD among all datasets, increasing the RECALL@10 values by up to 11.1 percentage points and NDCG@10 by up to 7.5 percentage points. Therefore, we conclude that combining adaptive learning rate optimizers in the MF model provides more accurate recommendations to users because adapting the learning rates during models' incremental updates makes the models adaptive to changes in data behavior.

Regarding disadvantages and limitations, our method increases the number of model parameters to train and update according to the number of optimizer parameters. In practice, the number of parameters is directly proportional to the number of base optimizer parameters. Suppose the base optimizer has a single parameter. In that case, our method stores the user and item latent factors as well as the user or item specialized optimizer parameters, i.e., for the RMSProp, the $\overrightarrow{v_{p(t)}}$ parameter, for Adam and Nadam, the $\overrightarrow{v_{p(t)}}$ and $\overrightarrow{m_{p(t)}}$ parameters, and for the AMSGrad, the $\overrightarrow{v_{p(t)}}$, $\overrightarrow{m_{p(t)}}$ and $\overrightarrow{max(v_{p(t)})}$ parameters.

In future work, we plan to investigate other adaptive learning rate optimizers to analyze their incremental efficiency in updating MF models. We intend to incorporate our optimizer variants into other MF recommender models, such as the PMF and BMPMF models presented in this study. In particular, we also plan on developing specialized and generalized optimization processes for high-dimensional and sparse (HiDS) collaborative filtering settings, including L_1-and-L_2 latent factor models [18], graph convolutional network latent feature analysis [19], and alternating direction methods [15], which are available for batch settings and still require adaptations for streaming settings. Finally, we plan to investigate the application of drift detectors as part of the learning process to adapt the model parameters according to changes in the data.

Acknowledgements

We would like to thank the *Conselho Nacional de Desenvolvimento Científico e Tecnológico* - CNPq (grant #142195/2019-7) for financing this research and HiMarket for their financial support. We also gratefully acknowledge the support by NVIDIA Corporation for the donation of the Titan V GPU used in this research.

References

[1] F. Ricci, L. Rokach, B. Shapira, Introduction to recommender systems handbook, in: Recommender Systems Handbook, Springer, 2011, pp. 1–35.

[2] S. Zhang, L. Yao, A. Sun, Y. Tay, Deep learning based recommender system: A survey and new perspectives, ACM Compututing Surveys 52 (2019) 5:1–5:38.

[3] G. Li, Z. Zhang, L. Wang, Q. Chen, J. Pan, One-class collaborative filtering based on rating prediction and ranking prediction, Knowledge Based Systems 124 (2017) 46–54.

[4] J. Bobadilla, F. Ortega, A. Hernando, A. Gutiérrez, Recommender systems survey, Knowledge-based systems 46 (2013) 109–132.

[5] T. Yu, O. J. Mengshoel, A. Jude, E. Feller, J. Forgeat, N. Radia, Incremental learning for matrix factorization in recommender systems, in: 2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016, IEEE Computer Society, 2016, pp. 1056–1063.

[6] A. D. Viniski, J. P. Barddal, A. de Souza Britto Jr., F. Enembreck, H. V. A. de Campos, A case study of batch and incremental recommender systems in supermarket data under concept drifts and cold start, Expert Systems with Applications 176 (2021) 114890.

[7] E. S. Babüroglu, A. Durmusoglu, T. Dereli, Novel hybrid pair recommendations based on a large-scale comparative study of concept drift detection, Expert Systems with Applications 163 (2021) 113786.

[8] I. Rabiu, N. Salim, A. Da'u, A. Osman, M. Nasser, Exploiting dynamic changes from latent features to improve recommendation using temporal matrix factorization, Egyptian Informatics Journal (2020).

[9] K. Laghmari, C. Marsala, M. Ramdani, An adapted incremental graded multi-label classification model for recommendation systems, Progress in Artificial Intelligence 7 (2018) 15–29.

[10] T. Tieleman, G. Hinton, Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning, Technical Report, Technical report, 2012.

[11] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[12] S. J. Reddi, S. Kale, S. Kumar, On the convergence of adam and beyond, in: 6th International Conference on Learning Representations, ICLR

2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018.

[13] T. Dozat, Incorporating nesterov momentum into adam (2016) 2013–2016.

[14] X. Luo, W. Qin, A. Dong, K. Sedraoui, M. Zhou, Efficient and high-quality recommendations via momentum-incorporated parallel stochastic gradient descent-based learning, IEEE/CAA Journal of Automatica Sinica 8 (2021) 402–411.

[15] X. Luo, M. Zhou, Z. Wang, Y. Xia, Q. Zhu, An effective scheme for qos estimation via alternating direction method-based matrix factorization, IEEE Transactions on Services Computing 12 (2016) 503–518.

[16] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, BPR: bayesian personalized ranking from implicit feedback, CoRR abs/1205.2618 (2012). `arXiv:1205.2618`.

[17] X. Luo, Y. Yuan, S. Chen, N. Zeng, Z. Wang, Position-transitional particle swarm optimization-incorporated latent factor analysis, IEEE Transactions on Knowledge and Data Engineering 34 (2022) 3958–3970.

[18] D. Wu, M. Shang, X. Luo, Z. Wang, An l1-and-l2-norm-oriented latent factor model for recommender systems, IEEE Transactions on Neural Networks and Learning Systems 33 (2022) 5775–5788.

[19] F. Bi, T. He, Y. Xie, X. Luo, Two-stream graph convolutional network-incorporated latent feature analysis, IEEE Transactions on Services Computing (2023).

[20] G. Takács, I. Pilászy, B. Németh, D. Tikk, Scalable collaborative filtering approaches for large recommender systems, Journal of Machine Learning Research 10 (2009) 623–656.

[21] Y. Koren, R. M. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, IEEE Computer 42 (2009) 30–37.

[22] J. Vinagre, A. M. Jorge, J. Gama, Fast incremental matrix factorization for recommendation with positive-only feedback, in: User Modeling, Adaptation, and Personalization - 22nd International Conference, UMAP

2014, Aalborg, Denmark, July 7-11, 2014. Proceedings, volume 8538 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 459–470.

[23] J. Chen, J. Fang, W. Liu, T. Tang, C. Yang, clmf: A fine-grained and portable alternating least squares algorithm for parallel matrix factorization, Future Gener. Comput. Syst. 108 (2020) 1192–1205.

[24] S. K. Raghuwanshi, R. K. Pateriya, Accelerated singular value decomposition (asvd) using momentum based gradient descent optimization, Journal of King Saud University-Computer and Information Sciences (2018).

[25] F. Rezaeimehr, P. Moradi, S. Ahmadian, N. N. Qader, M. Jalili, TCARS: time- and community-aware recommendation system, Future Gener. Comput. Syst. 78 (2018) 419–429. URL: `https://doi.org/10.1016/j.future.2017.04.003`. doi:10.1016/j.future.2017.04.003.

[26] X. Yuan, L. Han, S. Qian, G. Xu, H. Yan, Singular value decomposition based recommendation using imputed data, Knowl. Based Syst. 163 (2019) 485–494. URL: `https://doi.org/10.1016/j.knosys.2018.09.011`. doi:10.1016/j.knosys.2018.09.011.

[27] R. Salakhutdinov, A. Mnih, Probabilistic matrix factorization, in: J. C. Platt, D. Koller, Y. Singer, S. T. Roweis (Eds.), Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007, Curran Associates, Inc., 2007, pp. 1257–1264.

[28] L. Breiman, Bagging predictors, Machine Learning 24 (1996) 123–140.

[29] J. Ding, F. Feng, X. He, G. Yu, Y. Li, D. Jin, An improved sampler for bayesian personalized ranking by leveraging view data, in: Companion Proceedings of the The Web Conference 2018, 2018, pp. 13–14.

[30] T. Greenberg-Toledo, R. Mazor, A. Haj-Ali, S. Kvatinsky, Supporting the momentum training algorithm using a memristor-based synapse, IEEE Transactions on Circuits and Systems I: Regular Papers 66 (2019) 1571–1583.

[31] S. Sun, Z. Cao, H. Zhu, J. Zhao, A survey of optimization methods from a machine learning perspective, IEEE Transasctions on Cybernetics 50 (2020) 3668–3681.

[32] M. Kastrati, M. Biba, A state-of-the-art survey of advanced optimization methods in machine learning, in: E. Xhina, K. Hoxha (Eds.), Proceedings of the 4th International Conference on Recent Trends and Applications in Computer Science and Information Technology, Tirana, Albania, May 21st - to - 22nd, 2021, volume 2872 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 1–10.

[33] E. Dogo, O. Afolabi, N. Nwulu, B. Twala, C. Aigbavboa, A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks, in: 2018 international conference on computational techniques, electronics and mechanical systems (CTEMS), IEEE, 2018, pp. 92–99.

[34] Y. Yu, F. Liu, Effective neural network training with a new weighting mechanism-based optimization algorithm, IEEE Access 7 (2019) 72403–72410.

[35] S. Chaudhury, T. Yamasaki, Robustness of adaptive neural network optimization under training noise, IEEE Access 9 (2021) 37039–37053.

[36] D. M. Nguyen, E. Tsiligianni, N. Deligiannis, Learning discrete matrix factorization models, IEEE Signal Processing Letters 25 (2018) 720–724.

[37] J. McAuley, Amazon product data, 2014.

[38] F. M. Harper, J. A. Konstan, The movielens datasets: History and context, TiiS 5 (2016) 19:1–19:19.

[39] S. Dooms, T. D. Pessemier, L. Martens, Mining cross-domain rating datasets from structured data on twitter, in: 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume, ACM, 2014, pp. 621–624.

[40] J. Gama, R. Sebastião, P. P. Rodrigues, On evaluating stream learning algorithms, Machine Learning 90 (2013) 317–346.

[41] P. Cremonesi, Y. Koren, R. Turrin, Performance of recommender algorithms on top-n recommendation tasks, in: X. Amatriain, M. Torrens, P. Resnick, M. Zanker (Eds.), Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010, ACM, 2010, pp. 39–46.

[42] M. S. Kristoffersen, S. E. Shepstone, Z. Tan, A dataset for inferring contextual preferences of users watching TV, in: T. Mitrovic, J. Zhang, L. Chen, D. Chin (Eds.), Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization, UMAP 2018, Singapore, July 08-11, 2018, ACM, 2018, pp. 367–368.

[43] J. Demsar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.

[44] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of the american statistical association 32 (1937) 675–701.

[45] P. B. Nemenyi, Distribution-free multiple comparisons, PhD thesis, Princeton University, 1963.

[46] Q. Yuan, L. Chen, S. Zhao, Factorization vs. regularization: fusing heterogeneous social relationships in top-n recommendation, in: Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011, ACM, 2011, pp. 245–252.

[47] X. He, T. Chen, M. Kan, X. Chen, Trirank: Review-aware explainable recommendation by modeling aspects, in: Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015, ACM, 2015, pp. 1661–1670.