User Modeling and User-Adapted Interaction (2022) 32:509–541
https://doi.org/10.1007/s11257-021-09314-7

Embarrassingly shallow auto-encoders for dynamic collaborative filtering

Olivier Jeunen[1] · Jan Van Balen[2] · Bart Goethals[1,3]

Received: 4 March 2021 / Accepted in revised form: 28 November 2021 / Published online: 13 January 2022

Abstract

Recent work has shown that despite their simplicity, item-based models optimised through ridge regression can attain highly competitive results on collaborative filtering tasks. As these models are analytically computable and thus forgo the need for often expensive iterative optimisation procedures, they have become an attractive choice for practitioners. Computing the closed-form ridge regression solution consists of inverting the Gramian item-item matrix, which is known to be a costly operation that scales poorly with the size of the item catalogue. Because of this bottleneck, the adoption of these methods is restricted to a specific set of problems where the number of items is modest. This can become especially problematic in real-world dynamical environments, where the model needs to keep up with incoming data to combat issues of cold start and concept drift. In this work, we propose Dynamic EASE$^{\mathrm{R}}$: an algorithm based on the Woodbury matrix identity that incrementally updates an existing regression model when new data arrives, either approximately or exact. By exploiting a widely accepted low-rank assumption for the user-item interaction data, this allows us to target those parts of the resulting model that need updating, and avoid a costly inversion of the entire item-item matrix with every update. We theoretically and empirically show that our newly proposed methods can entail significant efficiency gains in the right settings, broadening the scope of problems for which closed-form models are an appropriate choice.

Keywords Recommender Systems · Collaborative Filtering · Online Learning · Ridge Regression · Linear Models

Jan Van Balen: work done while the author was at Adrem Data Lab.

✉ Olivier Jeunen
olivierjeunen@gmail.com

[1] Adrem Data Lab, Department of Computer Science, University of Antwerp, Antwerp, Belgium

[2] Spotify, Brussels, Belgium

[3] Faculty of Information Technology, Monash University, Clayton, VIC, Australia

1 Introduction

Recommender systems are information retrieval applications that aim to mitigate the problem of "information overload", by matching users to certain *items* (Borchers et al. 1998). They have become ubiquitous on the world wide web and have found applications in many different areas where these *items* can represent anything from news articles and musical artists to retail products and social media accounts. Most modern approaches to recommendation are based on some form of *collaborative filtering* (Ekstrand et al. 2011), a family of methods that aim to model user preferences and learn them from a dataset of user behaviour. These methods have known widespread success over the years, and are the cornerstone of modern recommender systems research. As a consequence, the quest for more effective collaborative filtering algorithms is a very active research area, where significant strides forward are being made every year. Many novel methods are based on deep and nonlinear neural networks, and the expressiveness of this model class has made them ubiquitous in the field (Liang et al. 2018; Elahi et al. 2019; Shenbin et al. 2020). Recent work casts doubt on the reproducibility of evaluation strategies that are often adopted to empirically validate research findings (Dacrema et al. 2019; Rendle 2019; Rendle et al. 2020), making it harder to conclude whether these complex model classes are what the field needs moving forward.

In a parallel line of research, the effectiveness of simpler linear models for the collaborative filtering task has been shown time and again (Ning and Karypis 2011; Levy and Jack 2013; Sedhain et al. 2016; Steck 2019b, c; Steck et al. 2020). Most notably and recently, Embarrassingly Shallow Auto-Encoders (reversed: EASE$^{\text{R}}$) have been shown to yield highly competitive results with the state of the art, whilst often being much easier to implement, and much more efficient to compute (Steck 2019a). The closed-form solution that is available for ridge regression models is at the heart of these major advantages, as EASE$^{\text{R}}$ effectively optimises a regularised least-squares problem. Recently, EASE$^{\text{R}}$ has been extended to incorporate item metadata into two variants: CEASE$^{\text{R}}$ and ADD- EASE$^{\text{R}}$ (Jeunen et al. 2020). These extensions improve the capabilities of closed-form linear models to deal with issues such as the "long tail" (very few items account for the large majority of interactions) and "cold start" (new items do not have any interactions) (Schein et al. 2002; Park and Tuzhilin 2008; Shi et al. 2014).

The main benefit of EASE$^{\text{R}}$ and its variants over competing approaches, is their computational efficiency. As the core algorithm consists of a single inversion of the Gramian item-item matrix, it is often many times more efficient to compute than models relying on iterative optimisation techniques. As reported in the original paper, the algorithm can be implemented in just a few lines of Python and is typically computed in the order of minutes on various often used publicly available benchmark datasets (Steck 2019a). Nevertheless, matrix inversion is known to scale poorly for large matrices, and EASE$^{\text{R}}$'s reliance on it does inhibit its adoption in use-cases with large item catalogues. In such cases, methods that rely on gradient-based optimisation techniques are still preferable.

To add insult to injury, real-world systems rarely rely on a single model that is computed once and then deployed. To make this concrete: suppose we operate a hypothetical retail website, and we wish to send out an e-mail with a top-N list of personalised recommendations to our subscribed users every few days. Naturally, the model that generates these recommendation lists should evolve over time, preferably incorporating new user-item interactions that have occurred over the past days. The importance of having such a dynamic model is threefold: (1) It will generate more novel and diverse recommendations than its static counterpart (Castells et al. 2015), (2) it will be able to combat concept drift in the data (due to shifting item popularity or seasonality trends in preferences) (Gama et al. 2014), and (3) it will have the means to handle cold-start problems when either with new items or news users appear (Schein et al. 2002).

Many modern digital systems generate new data at increasingly fast rates, and this is no different for our hypothetical retail website. This is important to take into account when choosing a recommendation algorithm. Models that are already inefficient to compute initially, will only see these problems exacerbated when the predominant approach every few days is to recompute them iteratively on more and more data. This puts a theoretical limit on how often we can update the model, and incurs a computational cost that we would like to reduce. Instead, it would be preferable to have models that can be updated with new information when it arrives, but do not require a full retraining of untouched parameters for every new batch of data that comes in. This is not an easy feat, and the field of "online recommender systems" that are able to handle model updates more elegantly has seen much interest in recent years (Vinagre et al. 2020). More generally, the problem of "lifelong" or "continual" learning in the machine learning field deals with similar issues (Chen 2018).

In this work, we present a novel algorithm to incrementally update the state-of-the-art item-based linear model EASE$^{\mathrm{R}}$, which is naturally extended to include recent variants that exploit side-information: CEASE$^{\mathrm{R}}$ and ADD- EASE$^{\mathrm{R}}$. EASE$^{\mathrm{R}}$ consists of two major computation steps: (1) the generation of the Gramian item-item matrix, and (2) the inversion of this matrix that yields the solution to the regression problem.

We propose Dynamic EASE$^{\mathrm{R}}$ (DYN- EASE$^{\mathrm{R}}$), consisting of incremental update rules for these two steps that leverage the recently proposed Dynamic Index algorithm (Jeunen et al. 2019) and the well-known Woodbury matrix identity (Hager 1989), respectively. As such, DYN- EASE$^{\mathrm{R}}$ provides a way to efficiently update an existing EASE$^{\mathrm{R}}$-like model without the need of recomputing the entire regression model from scratch with every data update.

A theoretical analysis of the proposed algorithm shows that the highest efficiency gains can be expected when the rank of the update to the Gramian is low, an assumption that has been widely adopted in the recommender systems literature before (Koren et al. 2009). We show how this quantity can be bounded using simple summary statistics from the new batch of data, and support our findings with empirical results. Further experiments confirm that DYN- EASE$^{\mathrm{R}}$ is able to significantly cut down on computation time compared to iteratively retrained EASE$^{\mathrm{R}}$, in a variety of recommendation domains. Finally, we show how we can update the model with low-rank approximations when the new batch of data itself is not low-rank; providing a tuneable trade-off between the exactness of the solution and the efficiency with which it can be kept up-to-date. Empirical observations show how this approximate variant of DYN- EASE$^{\mathrm{R}}$ still yields highly competitive recommendation performance, with greatly improved update speed, and how the low-rank assumption can even improve on recommendation accuracy. As a result, our work broadens the space of recommendation problems to which the state-of-the-art linear model EASE$^{\mathrm{R}}$ can efficiently be applied. To foster the reproducibility of our work, all source code for the experiments in Sect. 4 is publicly available at github.com/olivierjeunen/dynamic-easer.

The rest of this manuscript is structured as follows: Sect. 2 introduces our use-case, with mathematical notation and relevant related work; Sect. 3 introduces DYN- EASE$^{\mathrm{R}}$ and presents a theoretical analysis of its inner workings, motivating an approximate variant; Sect. 4 presents empirical observations from a wide range of experiments and shows where DYN- EASE$^{\mathrm{R}}$ can provide meaningful improvements, findings that are in line with what the theory suggests. Section 5 concludes our work, additionally presenting a scope for future research.

2 Background and related work

We first formalise our use-case and present relevant mathematical notation used throughout the rest of this work. We are interested in the "binary, positive-only" implicit feedback setting (Verstrepen et al. 2017), where we have access to a dataset consisting of preference indications from users in $\mathcal{U}$ over items in $\mathcal{I}$ at time $t \in \mathbb{N}$, assumed from a set of interaction data $\mathcal{P} \subseteq \mathcal{U} \times \mathcal{I} \times \mathbb{N}$. Ignoring temporal information, these preferences can be represented in a binary user-item matrix $\boldsymbol{X} \in \{0, 1\}^{|\mathcal{U}| \times |\mathcal{I}|}$, where $\boldsymbol{X}_{u,i} = 1$ if we have a click, view, purchase,…for user u and item i in $\mathcal{P}$, and $\boldsymbol{X}_{u,i} = 0$ otherwise. With $\mathcal{P}_t$, we denote the set of all interactions up to time t: $\{(u, i, t') \in \mathcal{P} | t' < t\}$. Consequently, $\boldsymbol{X}_t$ is the user-item matrix constructed from the set of interactions $\mathcal{P}_t$. We will refer to the set of all items seen by user u as $\mathcal{I}_u \subseteq \mathcal{I}$, and vice versa $\mathcal{U}_i \subseteq \mathcal{U}$ for an item i. The Gramian of the user-item matrix is defined as $\boldsymbol{G} := \boldsymbol{X}^{\intercal}\boldsymbol{X}$; it is an item-item matrix that holds the co-occurrence count for items i and j at index $\boldsymbol{G}_{i,j}$. The goal at hand for a recommendation algorithm is to predict which zeroes in the user-item matrix $\boldsymbol{X}$ actually *shouldn't* be zeroes, and thus imply that the item would in some way "fit" the user's tastes and consequently make for a good item to be shown as a recommendation.

In some cases, additional information about items can be available. Such "side-information" or "metadata" often comes in the form of discrete *tags*, which can, for example, be a release year, genre or director for a movie, an artist or genre for a song, a writer for a book, or many more. Incorporating item metadata in the modelling process can help mitigate cold-start and long-tail issues, where the preference information for a given item is limited (Schein et al. 2002; Park and Tuzhilin 2008). We will refer to the set of all such tags as the *vocabulary* $\mathcal{V}$. In a similar fashion to the user-item matrix $\boldsymbol{X}$, a tag-item matrix $\boldsymbol{T} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{I}|}$ is constructed. Note that this matrix is real-valued, as it will often contain pre-computed values such as tf-idf weights instead of binary indicators.

In what follows, we present a brief introduction to item-based recommendation models, most notably ITEM- KNN (Sarwar et al. 2001), SLIM (Ning and Karypis 2011) and EASE$^{\mathrm{R}}$ (Steck 2019a). We then additionally introduce CEASE$^{\mathrm{R}}$ and ADD- EASE$^{\mathrm{R}}$ as extensions of EASE$^{\mathrm{R}}$ that incorporate item side-information whilst retaining a closed-form solution (Jeunen et al. 2020), as these are most relevant to the dynamic EASE$^{\mathrm{R}}$

algorithm we will present in Sect. 3. This section is concluded with an overview of related work in the field of incremental collaborative filtering approaches.

2.1 Item-based models, SLIM and EASE$^\text{R}$

Item-based collaborative filtering models tackle the recommendation task by defining a conceptual similarity matrix $\boldsymbol{S} \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{I}|}$. The score given to a potential recommendation is then computed as the sum of similarities between items in the user's history and the item at hand:

$$\text{score}(u, i) = \sum_{j\in\mathcal{I}_u} \boldsymbol{S}_{j,i} = (\boldsymbol{X}_{u,\cdot}\boldsymbol{S})_i \tag{1}$$

Here, $\boldsymbol{X}_{\boldsymbol{u},\cdot}$ denotes the u^{th} row of $\boldsymbol{X}$. Note that computing recommendation scores for all training users and all items simply consists of computing the matrix multiplication $\boldsymbol{X}\boldsymbol{S}$, an operation that is made more efficient when the matrix $\boldsymbol{S}$ is restricted to be sparse. Scores for items i already present in the user history $\mathcal{I}_u$ are often ignored, and the remaining items are ranked and presented in a top-N recommendation list or slate to the user. Early seminal works would define the similarity matrix $\boldsymbol{S}$ as all pairwise cosine similarities among items in the high-dimensional but sparse user-item matrix $\boldsymbol{X}$ (Sarwar et al. 2001). This has then been extended to include slightly more advanced notions of similarity such as Pearson's correlation or conditional probabilities (Deshpande and Karypis 2004). Recent work has introduced the "Dynamic Index" algorithm to incrementally compute the Gramian of $\boldsymbol{X}$, additionally showing that several conventional similarity metrics such as cosine similarity or Jaccard index can be readily computed from $\boldsymbol{G}$ when it is up-to-date (Jeunen et al. 2019).

Methods for actually *learning* an optimal item-item similarity matrix have been proposed for the task of rating prediction (Koren 2008), as well as for pairwise learning from implicit feedback (Rendle et al. 2009). Ning and Karypis were the first to propose to learn a sparse weight matrix $\boldsymbol{S}$ through a pointwise optimisation procedure, aptly dubbing their approach the Sparse LInear Method (SLIM) (Ning and Karypis 2011). SLIM optimises a least-squares regression model with elastic net regularisation, constrained to positive weights:

$$\begin{aligned} \boldsymbol{S}^* = \arg\min{}_{\boldsymbol{S}} \left\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{S}\right\|_F^2 + \lambda_1 \left\|\boldsymbol{S}\right\|_1^2 + \lambda_2 \left\|\boldsymbol{S}\right\|_F^2 , \\ \text{subject to } \operatorname{diag}(\boldsymbol{S}) = 0 \text{ and } \boldsymbol{S} \geq 0. \end{aligned} \tag{2}$$

The restriction of the diagonal to zero avoids the trivial solution where $\boldsymbol{S} = \boldsymbol{I}$. Many extensions of SLIM have been proposed in recent years, and it has become a widely used method for the collaborative filtering task (Ning and Karypis 2012; Levy and Jack 2013; Christakopoulou and Karypis 2014; Sedhain et al. 2016; Christakopoulou and Karypis 2016; Steck 2019a, c; Steck et al. 2020; Chen et al. 2020). In practice, the SLIM optimisation problem is often decomposed into $|\mathcal{I}|$ independent problems (one per target item). Although these can then be solved in an embarrassingly parallel fashion, this renders the approach intractable for very large item catalogues. Indeed, as they aim to solve $|\mathcal{I}|$ regression problems, their computational complexity is in the order of $\mathcal{O}(|\mathcal{I}|(|\mathcal{I}|-1)^{2.373})$, assuming they exploit the recent advances in efficient matrix multiplication and inversion (Le Gall 2014; Alman and Vassilevska W. 2021). The computational cost of the original SLIM approach is a known impediment for its adoption in certain use-cases; related work has reported that hyper-parameter tuning took several weeks on even medium-sized datasets (Liang et al. 2018).[1]

Steck studied whether the restrictions of SLIM to only allow *positive* item-item weights and their l_1-regularisation-induced sparsity were necessary for the resulting model to remain competitive, and concluded that this was not always the case (Steck 2019a; Steck et al. 2020). The resulting Tikhonov-regularised least-squares problem can then be formalised as:

$$\boldsymbol{S}^{*} = \arg\min_{\boldsymbol{S}} \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{S}\|_F^2 + \lambda \|\boldsymbol{S}\|_F^2 \text{, subject to } \mathrm{diag}(\boldsymbol{S}) = 0. \quad (3)$$

The main advantage of simplifying the optimisation problem at hand is that the well-known closed-form solutions for ordinary least squares (OLS) and ridge regression can now be adopted. Including the zero-diagonal constraint via Lagrange multipliers yields the Embarrassingly Shallow Auto-Encoder (EASE$^{\mathrm{R}}$) model:

$$\hat{\boldsymbol{S}} = \boldsymbol{I} - \hat{\boldsymbol{P}} \cdot \mathrm{diagMat}(\boldsymbol{1} \oslash \mathrm{diag}(\hat{\boldsymbol{P}})), \text{ where } \hat{\boldsymbol{P}} := (\boldsymbol{X}^{\intercal}\boldsymbol{X} + \lambda \boldsymbol{I})^{-1}. \quad (4)$$

As this model consists of a single regression problem to be solved and thus a single matrix inversion to be computed, its complexity is orders of magnitude smaller than that of the original SLIM variants: $\mathcal{O}(|\mathcal{I}|^{2.373})$. EASE$^{\mathrm{R}}$ no longer yields a sparse matrix, possibly making Equation 1 much less efficient to compute. Nevertheless, the author reported that there was only a marginal performance impact when simply sparsifying the learnt matrix by zeroing out weights based on their absolute values up until the desired sparsity level. As an additional advantage, EASE$^{\mathrm{R}}$ has only a single regularisation strength hyper-parameter to tune compared to the two needed for SLIM's elastic net regularisation. We refer the interested reader to Steck (2019a, b) for a full derivation of the model and additional information.

Another recent extension of the SLIM paradigm proposes to use Block-Diagonal-Regularisation (BDR) to obtain a block-aware item similarity model (Chen et al. 2020). The block-diagonal structure in the learnt matrix inherently represents clusters among items. As inter-block similarities are penalised, BDR has a sparsity-inducing effect that positively impacts the efficiency of the recommendation-generating process. Because the block-aware model presented by Chen et al. (2020) no longer has an analytically computable solution readily available, further comparison with their method is out of scope for the purposes of this work. The item-based paradigm and its closed-form instantiations have also recently been adapted for bandit-based recommendation use-cases (Jeunen and Goethals 2021).

[1] It should be noted that the authors have since released a more performant coordinate-descent-based implementation of their method (Ning et al. 2019).

2.2 Item-based models with side-information

The EASE$^{\text{R}}$ definition can be further extended to incorporate side-information in either a "collective" (CEASE$^{\text{R}}$) or "additive" (ADD- EASE$^{\text{R}}$) manner (Jeunen et al. 2020). The first method, inspired by collective SLIM (Ning and Karypis 2012), intuitively treats discrete tags equivalent to how users are treated, and re-weights their contribution to the solution of the regression problem by the diagonal weight-matrix $\boldsymbol{W} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{V}|)\times(|\mathcal{U}|+|\mathcal{V}|)}$:

$$\begin{aligned} \boldsymbol{S}^{*} &= \arg\min_{\boldsymbol{S}} \left\| \sqrt{\boldsymbol{W}}(\boldsymbol{X}' - \boldsymbol{X}'\boldsymbol{S}) \right\|_F^2 + \lambda \left\|\boldsymbol{S}\right\|_F^2, \\ &\text{subject to } \operatorname{diag}(\boldsymbol{S}) = 0, \text{ where } \boldsymbol{X}' = \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{T} \end{bmatrix}. \end{aligned} \tag{5}$$

The closed-form solution is then given by Equation 6, where $\oslash$ denotes element-wise division, diag$(\cdot)$ extracts the diagonal from a matrix, diagMat$(\cdot)$ generates a square diagonal matrix from a vector, and **1** is a vector of ones.

$$\hat{\boldsymbol{S}} = \boldsymbol{I} - \hat{\boldsymbol{P}} \cdot \operatorname{diagMat}(\mathbf{1} \oslash \operatorname{diag}(\hat{\boldsymbol{P}})), \text{ where } \hat{\boldsymbol{P}} := (\boldsymbol{X}'^{\intercal}\boldsymbol{W}\boldsymbol{X}' + \lambda\boldsymbol{I})^{-1} \tag{6}$$

The second method, ADD- EASE$^{\text{R}}$, treats the regression problem on the user-item matrix $\boldsymbol{X}$ and the one on the tag-item matrix $\boldsymbol{T}$ as two fully independent problems to solve in parallel; combining the two resulting item-item weight matrices $\boldsymbol{S}_{\boldsymbol{X}}$ and $\boldsymbol{S}_{\boldsymbol{T}}$ in an additive fashion later down the line.

$$\begin{aligned} \boldsymbol{S}^{*} &= \alpha \arg\min_{\boldsymbol{S}_{\boldsymbol{X}}} \left(\left\| \sqrt{\boldsymbol{W}_{\boldsymbol{X}}}(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{S}_{\boldsymbol{X}}) \right\|_F^2 + \lambda_{\boldsymbol{X}} \left\|\boldsymbol{S}_{\boldsymbol{X}}\right\|_F^2 \right) \\ &+ (1-\alpha) \arg\min_{\boldsymbol{S}_{\boldsymbol{T}}} \left(\left\| \sqrt{\boldsymbol{W}_{\boldsymbol{T}}}(\boldsymbol{T} - \boldsymbol{T}\boldsymbol{S}_{\boldsymbol{T}}) \right\|_F^2 + \lambda_{\boldsymbol{T}} \left\|\boldsymbol{S}_{\boldsymbol{T}}\right\|_F^2 \right), \\ &\text{subject to } \operatorname{diag}(\boldsymbol{S}_{\boldsymbol{X}}) = \operatorname{diag}(\boldsymbol{S}_{\boldsymbol{T}}) = 0. \end{aligned} \tag{7}$$

ADD- EASE$^{\text{R}}$ doubles the amount of parameters used by EASE$^{\text{R}}$ and CEASE$^{\text{R}}$, increasing its degrees of freedom at learning time at the cost of having to solve two regression problems instead of one. Note, however, that these are fully independent and can be computed in parallel. Equation 8 shows the analytical formulas to obtain the two independent models, and combine them with a blending parameter $0 \leq \alpha \leq 1$.

$$\begin{aligned} \hat{\boldsymbol{S}}_{\boldsymbol{X}} &= \boldsymbol{I} - \hat{\boldsymbol{P}}_{\boldsymbol{X}} \cdot \operatorname{diagMat}(\mathbf{1} \oslash \operatorname{diag}(\hat{\boldsymbol{P}}_{\boldsymbol{X}})), \text{ where } \hat{\boldsymbol{P}}_{\boldsymbol{X}} := (\boldsymbol{X}^{\intercal}\boldsymbol{W}_{\boldsymbol{X}}\boldsymbol{X} + \lambda_{\boldsymbol{X}}\boldsymbol{I})^{-1} \\ \hat{\boldsymbol{S}}_{\boldsymbol{T}} &= \boldsymbol{I} - \hat{\boldsymbol{P}}_{\boldsymbol{T}} \cdot \operatorname{diagMat}(\mathbf{1} \oslash \operatorname{diag}(\hat{\boldsymbol{P}}_{\boldsymbol{T}})), \text{ where } \hat{\boldsymbol{P}}_{\boldsymbol{T}} := (\boldsymbol{T}^{\intercal}\boldsymbol{W}_{\boldsymbol{T}}\boldsymbol{T} + \lambda_{\boldsymbol{T}}\boldsymbol{I})^{-1} \\ \hat{\boldsymbol{S}} &= \alpha\hat{\boldsymbol{S}}_{\boldsymbol{X}} + (1-\alpha)\hat{\boldsymbol{S}}_{\boldsymbol{T}} \end{aligned} \tag{8}$$

The computational complexity of CEASE$^{\text{R}}$ and ADD- EASE$^{\text{R}}$ remains in the order of $\mathcal{O}(|\mathcal{I}|^{2.373})$, which is equivalent to the original EASE$^{\text{R}}$ approach. As such, these methods allow item side-information to be included into the model without a significant added cost in terms of computational complexity. The main reason for this, is that we adapt the entries in the Gramian $\boldsymbol{G}$, but do not alter its dimensions.

2.3 Incremental collaborative filtering

Collaborative filtering techniques that can be incrementally updated when new data arrives are a lively research area in itself. Vinagre et al. (2014) propose incremental Stochastic Gradient Descent (SGD) as a way to dynamically update matrix factorisation models based on positive-only implicit feedback. Their methodology has first been extended to include negative feedback (Vinagre et al. 2015), and then to a co-factorisation model that is more complex than traditional matrix factorisation, but also leads to superior recommendation accuracy (Anyosa et al. 2018). He et al. (2016) propose an incremental optimisation procedure based on Alternating Least Squares (ALS), and also show how it can be applied to efficiently and effectively update matrix factorisation models. More recently, Ferreira et al. propose a method that personalises learning rates on a user-basis, reporting further improvements. In contrast, our work focuses on item-based similarity models that come with closed-form solutions, as these have been shown to be highly competitive with the state of the art in many collaborative filtering use-cases.

Instead of just incorporating new data into the model, Matuszyk et al. (2018) propose to *forget* older data that has become obsolete, reporting significantly improved performance for collaborative filtering approaches. The dynamic EASE^{R} method we propose in Sect. 3 fits perfectly into this paradigm, as it can incorporate new data just as easily as it can forget irrelevant information in a targeted manner. This type of decremental learning has the additional advantage of being able to avoid complete retraining in privacy-sensitive application areas, where specific user histories need to be removed from the model upon request.

2.4 Neural auto-encoders

The Auto-Encoder paradigm of which EASE^{R} is a specific instantiation, has gained much popularity in recent years. The Mult-VAE method proposed by Liang et al. (2018) consists of a variational auto-encoder with a multinomial likelihood, and has been a strong baseline for several years (Dacrema et al. 2019). Khawar et al. (2020) propose an architecture that first learns a grouping of items and leverages this structure when learning the auto-encoder, reporting significant gains over the original Mult-VAE method. As these methods rely on gradient-based optimisation of often highly non-convex objective functions, they rely on software packages with automatic differentiation capabilities, and typically require significant computational resources, in the form of several hours of training on machines equipped with GPUs. The methods we consider in this work are computed in the order of minutes on CPUs, and we do not include neural approaches in our comparison for this reason. Furthermore, among others, the work of Steck (2019a) and Dacrema et al. (2019) have repeatedly shown that linear item-based models can attain highly competitive recommendation accuracy compared to neural alternatives.

3 Methodology and contributions

We have given a brief history of item-based collaborative filtering models, and have discussed why EASE^R and its variants are computationally often more efficient than their counterparts based on SLIM. For very large item catalogues, however, its more than quadratic computational complexity in the number of items still becomes a very tangible issue. Because of this, the demand for an algorithm that can efficiently update EASE^R-like models when new data arrives, is still very real, and a necessity for these methods to obtain widespread adoption in practice. Recent work proposes the "Dynamic Index" algorithm as a way to incrementally update item similarities in neighbourhood-based models that adopt cosine similarity (Jeunen et al. 2019). A crucial building block of this metric and the algorithm is the efficient and incremental computation of the Gramian matrix $\boldsymbol{G} = \boldsymbol{X}^\intercal \boldsymbol{X}$. By storing $\boldsymbol{G}$ in low-overhead sparse data-structures such as inverted indices, they minimise memory overhead whilst still allowing for an amortised constant lookup time when querying $\mathcal{I}_u$, which is a requirement for incremental updates. From Eqs. 4, 6 and 8 , it is clear to see that EASE^R and its variants are dependent on this Gramian matrix as well. In fact, it is the *only* building block needed to be able to compute the resulting item-item weight matrix $\hat{\boldsymbol{S}}$. As such, we adopt parts of the Dynamic Index algorithm proposed by Jeunen et al. to first efficiently compute and then incrementally update the Gramian matrix $\boldsymbol{G}$. Once we have an up-to-date matrix $\boldsymbol{G}$, we need to compute its inverse to obtain $\hat{\boldsymbol{P}}$ and the eventual model $\hat{\boldsymbol{S}}$ from that. The matrix inversion to go from $\boldsymbol{G}$ to $\hat{\boldsymbol{P}}$ is the workhorse behind EASE^R that takes up the large majority of the computation time, as this step corresponds to solving the least-squares problem formulated in Eq. 3. Iterative re-computation of this matrix inverse every time we wish to incorporate new data into the model, is thus to be avoided if it can be.

3.1 Low-rank model updates with the Woodbury matrix identity

Equation 9 shows the Woodbury matrix identity, which posits that the inverse of a rank-k correction to some $n \times n$ matrix $\boldsymbol{A}$ can be computed by performing a rank-k correction on the inverse of the original matrix (Hager 1989).

$$(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{C}\boldsymbol{V})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{C}^{-1} + \boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1} \tag{9}$$

So, given $\boldsymbol{A}^{-1}$, $\boldsymbol{U}$, $\boldsymbol{C}$ and $\boldsymbol{V}$, there is no need to re-compute the inversion on the update of $\boldsymbol{A}$, but it is sufficient to multiply a few matrices and compute the inverse of $(\boldsymbol{C}^{-1} + \boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U}) \in \mathbb{R}^{k \times k}$. Naturally, for large n and $k \ll n$, the efficiency gains coming from this reformulation will be most significant. Although this is no requirement for Eq. 9 to hold, we assume $\boldsymbol{C} \in \mathbb{R}^{k \times k}$ to be a diagonal matrix. As a result, the inversion of $\boldsymbol{C}$ becomes trivial and consists of just k operations.

In our setting, suppose we have an up-to-date model at a certain time t with $\boldsymbol{X}_t$, $\boldsymbol{G}_t$, $\hat{\boldsymbol{P}}_t$ and $\hat{\boldsymbol{S}}_t$. At a given time $t+1$, suppose we have an updated user-item matrix $\boldsymbol{X}_{t+1}$, but we wish to compute $\boldsymbol{G}_{t+1}$, $\hat{\boldsymbol{P}}_{t+1}$ and the resulting $\hat{\boldsymbol{S}}_{t+1}$ as efficiently as possible. As we mentioned before, computing $\boldsymbol{G}_{t+1}$ incrementally can be achieved easily and efficiently by adopting parts of the Dynamic Index algorithm. In fact, because of the incremental nature of the algorithm, we can easily just store the *difference* in the Gramian matrix instead of its entirety: $\boldsymbol{G}_{\boldsymbol{\Delta}} = \boldsymbol{G}_{t+1} - \boldsymbol{G}_t = \boldsymbol{X}_{t+1}^{\intercal}\boldsymbol{X}_{t+1} - \boldsymbol{X}_t^{\intercal}\boldsymbol{X}_t$. Given a set of user-item interactions $\mathcal{P}_{\boldsymbol{\Delta}} \subset \mathcal{U} \times \mathcal{I}$ to include into the model and an inverted index $\mathcal{L}_t$ mapping users u to their histories $\mathcal{I}_u$, Algorithm 1 shows how to achieve this. Note that the indices holding $\mathcal{I}_u$ are just a sparse representation of the user-item matrix $\boldsymbol{X}$ and don't require any additional memory consumption. Furthermore, Algorithm 1 is easily parallellisable through the same MapReduce-like paradigm adopted by Jeunen et al. (2019). Naturally, an efficient implementation will exploit the symmetry of the Gramian $\boldsymbol{G}_{\boldsymbol{\Delta}}$ to decrease memory consumption as well as the number of increments needed at every update.

Algorithm 1 DYN- GRAM

```
Input: $\mathcal{P}_{\boldsymbol{\Delta}}, \mathcal{L}$
Output: $\boldsymbol{G}_{\boldsymbol{\Delta}}, \mathcal{L}$
$\boldsymbol{G}_{\boldsymbol{\Delta}} = 0$
for $(u, i) \in \mathcal{P}_{\boldsymbol{\Delta}}$ do
    for $j \in \mathcal{L}[u]$ do
        $\boldsymbol{G}_{\boldsymbol{\Delta},i,j} \mathrel{+}= 1$
        $\boldsymbol{G}_{\boldsymbol{\Delta},j,i} \mathrel{+}= 1$
    $\boldsymbol{G}_{\boldsymbol{\Delta},i,i} \mathrel{+}= 1$
    $\mathcal{L}[u] = \mathcal{L}[u] \cup \{i\}$
return $\boldsymbol{G}_{\boldsymbol{\Delta}}, \mathcal{L}$
```

Now, having computed $\boldsymbol{G}_{\boldsymbol{\Delta}}$, we can rewrite what we need as follows:

$$\hat{\boldsymbol{P}}_{t+1} = (\boldsymbol{G}_{t+1} + \lambda \boldsymbol{I})^{-1} = (\boldsymbol{G}_t + \lambda \boldsymbol{I} + \boldsymbol{G}_{\boldsymbol{\Delta}})^{-1}. \tag{10}$$

The form on the right-hand side already begins to resemble Woodbury's formula in Eq. 9. All that's left is to decompose $\boldsymbol{G}_{\boldsymbol{\Delta}} \in \mathbb{R}^{n \times n}$ into matrices $\boldsymbol{U} \in \mathbb{R}^{n \times k}$, $\boldsymbol{C} \in \mathbb{R}^{k \times k}$ and $\boldsymbol{V} \in \mathbb{R}^{k \times n}$. As $\boldsymbol{G}_{\boldsymbol{\Delta}}$ is the difference of two real symmetric matrices $\boldsymbol{G}_{t+1}$ and $\boldsymbol{G}_t$, it will always be a real symmetric matrix as well. This means that the eigenvectors of $\boldsymbol{G}_{\boldsymbol{\Delta}}$ can be chosen to be orthogonal to each other: $\boldsymbol{Q}^{\intercal} \equiv \boldsymbol{Q}^{-1}$. Consequently, an eigendecomposition always exists, where k is the rank of $\boldsymbol{G}_{\boldsymbol{\Delta}}$:

$$\begin{aligned} \boldsymbol{G}_{\boldsymbol{\Delta}} &= \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1} \\ &= \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\intercal} \\ &= \sum_{i=1}^{k} \boldsymbol{\Lambda}_{ii}\, \boldsymbol{Q}_{\cdot,i}\, \boldsymbol{Q}_{\cdot,i}^{\intercal}. \end{aligned} \tag{11}$$

As such, we can plug Eq. 11 containing the eigendecomposition of $\boldsymbol{G}_{\boldsymbol{\Delta}}$ into Eqs. 9 and 10 to obtain our final update rule in Eq. 12:

$$\begin{aligned} \hat{\boldsymbol{P}}_{t+1} &= (\boldsymbol{G}_t + \lambda \boldsymbol{I} + \boldsymbol{G}_{\boldsymbol{\Delta}})^{-1} \\ &= (\boldsymbol{G}_t + \lambda \boldsymbol{I} + \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\intercal})^{-1} \\ &= \hat{\boldsymbol{P}}_t - \hat{\boldsymbol{P}}_t \boldsymbol{Q}(\boldsymbol{\Lambda}^{-1} + \boldsymbol{Q}^{\intercal}\hat{\boldsymbol{P}}_t \boldsymbol{Q})^{-1}\boldsymbol{Q}^{\intercal}\hat{\boldsymbol{P}}_t. \end{aligned} \tag{12}$$

The full DYN- EASE$^{\mathrm{R}}$ procedure is presented in Algorithm 2. If the updates to the Gramian matrix are low-rank, this procedure will be much more computationally efficient than re-computing the inverse of the entire Gramian matrix from scratch, as we will show in the following subsection. The assumption that the data-generating process behind user-item interactions is generally low-rank, has been exploited far and wide in the recommender systems literature (Koren et al. 2009).

It is interesting to note that EASE$^{\mathrm{R}}$ does not follow the low-rank assumption that motivates the popular family of latent factor models for collaborative filtering. Indeed, EASE$^{\mathrm{R}}$ is a full-rank model, combatting overfitting with Gaussian priors on its parameters rather than reducing the dimensionality of the problem. The low-rank assumption we adopt here is on the update to the Gramian $\boldsymbol{G}_{\boldsymbol{\Delta}}$, instead of the full Gramian $\boldsymbol{G}$. As we will show further on, both theoretically and empirically, this assumption holds in a variety of settings.

The fact that $\boldsymbol{G}_{\boldsymbol{\Delta}}$ is symmetric and will often be very sparse in nature can be exploited when computing the eigendecomposition on line 3 of Algorithm 2, as we will show in the following section. Many modern software packages for scientific computing implement very efficient procedures specifically for such cases (e.g. SciPy (Virtanen et al. 2020)). Note that alternative algorithms to factorise $\boldsymbol{G}_{\boldsymbol{\Delta}}$ into lower-dimensional matrices exist, often relying on randomised sampling procedures (Martinsson et al. 2011; Halko et al. 2011). These algorithms are reportedly more efficient to compute than the traditional eigendecomposition, but often not geared specifically toward the high-dimensional yet sparse use-case we tackle in this work, or not equipped to exploit the symmetric structure that is typical for the Gramian. As they compute two dense matrices of $\boldsymbol{Q}$'s dimensions—their improvement in computation time comes with the cost of increased memory consumption. Furthermore, these methods are often focused on approximate matrix reconstructions whereas we are interested in an exact decomposition of the update to the Gramian. As the eigendecomposition fulfils our needs, the study of alternative factorisation methods falls out of the scope of the present work.

Throughout this section, we have focused on DYN- EASE$^{\mathrm{R}}$ as a general extension of EASE$^{\mathrm{R}}$. Naturally, our approach is trivially extended to include CEASE$^{\mathrm{R}}$, ADD- EASE$^{\mathrm{R}}$ or a weight matrix $\boldsymbol{W}$ different from the identity matrix $\boldsymbol{I}$ as well, as these variants only change the input to Algorithms 1 and 2, but bear no impact on the procedures themselves.

Algorithm 2 Exact DYN- EASE$^{\mathrm{R}}$

```
Input: $\hat{\boldsymbol{P}}_t, \mathcal{P}_{\boldsymbol{\Delta}}, \mathcal{L}_t$
Output: $\hat{\boldsymbol{P}}_{t+1}, \mathcal{L}_t + 1$.
$\boldsymbol{G}_{\boldsymbol{\Delta}}, \mathcal{L}_{t+1} =$ DYN- GRAM($\mathcal{P}_{\boldsymbol{\Delta}}, \mathcal{L}_t$) // (Algorithm 1)
$k =$ estimate-rank($\boldsymbol{G}_{\boldsymbol{\Delta}}$) // (Liberty et al. 2007; Ubaru and Saad 2016)
$\Lambda, \boldsymbol{Q} =$ eigendecomposition($\boldsymbol{G}_{\boldsymbol{\Delta}}, k$)
$\hat{\boldsymbol{P}}_{t+1} = \hat{\boldsymbol{P}}_t - \hat{\boldsymbol{P}}_t \boldsymbol{Q}(\boldsymbol{\Lambda}^{-1} + \boldsymbol{Q}^{\intercal} \hat{\boldsymbol{P}}_t \boldsymbol{Q})^{-1} \boldsymbol{Q}^{\intercal} \hat{\boldsymbol{P}}_t$
return $\hat{\boldsymbol{P}}_{t+1}$
```

3.2 Computational complexity analysis of eigendecomposition

The computational complexity of EASE$^{\mathrm{R}}$ is determined by the inversion of the Gramian, whereas the complexity of DYN- EASE$^{\mathrm{R}}$ is dictated by that of the eigendecomposition of the update to the Gramian. The computational complexity of matrix inversion, as well as that of solving the eigen-problem of a matrix, can be reduced to that of matrix multiplication (Pan and Chen 1999; Le Gall 2014). Given a square matrix of size $n \times n$, this is generally thought of as an $\mathcal{O}(n^3)$ problem. Nevertheless, specialised methods that provide improved bounds on the exponent exist, the most recent one being $\mathcal{O}(n^{2.37286})$ by Alman and Vassilevska W. (2021).

In practice, it is easily seen that more efficient algorithms can be applied to specific cases instead of the general approach. Indeed, the inversion of a diagonal matrix consists of just n operations, and algorithms to multiply sparse matrices are often much more efficient than their dense counterparts. In what follows, we provide a brief theoretical analysis of the complexity of DYN- EASE$^{\mathrm{R}}$, giving rise to an improved estimate for its computational complexity in practical settings. This bound explains the efficiency improvements of DYN- EASE$^{\mathrm{R}}$ over EASE$^{\mathrm{R}}$, and recovers the equivalence of eigendecomposition to matrix inversion in the general case.

A first important thing to note is that the Gramian is symmetric, and so is $\boldsymbol{G}_{\boldsymbol{\Delta}}$. This allows us to use the iterative method proposed by Lanczos (1950) to compute its eigen-vectors and -values.[2] The core algorithm proposed by Lanczos consists of k steps—one per nonzero eigenpair—which in turn consist of several vector and matrix manipulations. We refer the interested reader to an excellent analysis of the Lanczos algorithm provided by Paige (1980), showing *how* it works and *why* it converges. The computational complexity of every step in the method is determined by that of a matrix-vector product between the input $\boldsymbol{G}_{\boldsymbol{\Delta}}$ and an $|\mathcal{I}|$-dimensional vector. In the general case, such an operation is $\mathcal{O}(|\mathcal{I}|^2)$. In our specific case, however, $\boldsymbol{G}_{\boldsymbol{\Delta}}$ is often of an extremely sparse nature. This allows us to describe the complexity of the product as $\mathcal{O}(m \cdot |\mathcal{I}|)$, where m is the average number of nonzero values in every column of $\boldsymbol{G}_{\boldsymbol{\Delta}}$. Repeating these steps for every nonzero eigen-value-vector pair yields a final computational complexity of $\mathcal{O}(k \cdot m \cdot |\mathcal{I}|)$. When we wish to do a full-rank update on a dense matrix (i.e. $k = m = |\mathcal{I}|$), this recovers the computational complexity of general matrix inversion: $\mathcal{O}(|\mathcal{I}|^3)$. In the cases where either the rank of the update is low ($k \ll |\mathcal{I}|$) or the update to the Gramian is highly sparse ($m \ll |\mathcal{I}|$), the

[2] In our experiments, we use an efficient SciPy implementation of a variant called the Implicitly Restarted Lanczos Method (Lehoucq et al. 1998; Virtanen et al. 2020); the analysis is equivalent.

eigendecomposition will be most efficient and as a consequence, the performance benefits of DYN- EASE$^{\text{R}}$ over EASE$^{\text{R}}$ will be most apparent too. Note that although low-rankness and sparsity will often come in pairs in the practical settings we deal with, this does not have to be the case in general. As a counterexample: the identity matrix $\boldsymbol{I}$ is highly sparse yet full-rank.

3.3 Efficient estimation and upper bounding of rank($\boldsymbol{G}_{\boldsymbol{\Delta}}$)

In order to compute the eigendecomposition on line 3 of Algorithm 2, the numerical rank of $\boldsymbol{G}_{\boldsymbol{\Delta}}$ would need to be known a priori. Furthermore, as we have shown, the efficiency of the update procedure is highly dependent on the assumption that this rank is much smaller than the dimensionality of the Gram-matrix itself: $k \ll |\mathcal{I}|$. It is known that matrix ranks can be estimated efficiently through the use of randomised methods (Liberty et al. 2007; Ubaru and Saad 2016); when dealing with sparse and symmetric matrices, these methods tend to attain extremely efficient performance.[3] Being able to estimate rank($\boldsymbol{G}_{\boldsymbol{\Delta}}$) of course does not guarantee that this quantity will be low. In practice, however, we notice that it is often the case. We can see that the rank of the update $\boldsymbol{G}_{\boldsymbol{\Delta}}$ depends on (1) the number of unique users in the update $\mathcal{P}_{\boldsymbol{\Delta}}$, denoted by $|\mathcal{U}_{\boldsymbol{\Delta}}|$, and (2) the average number of items in the *entire* history of these users: $\overline{|\mathcal{I}_{\mathcal{U}_{\boldsymbol{\Delta}}}|}$.

This can be intuitively seen from the fact that an index i, j in the Gramian matrix represents the number of co-occurrences between the items i and j in the dataset. As such, a new user-item interaction $(u, i) \in \mathcal{P}_{\boldsymbol{\Delta}}$ affects $\boldsymbol{G}_{i,j}, \forall j \in \mathcal{I}_u$.

Now, let $\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot]}$ be the user-item matrix containing all (including historical) user-item interactions from only the users that appear in the update. This means we can rewrite the updated Gramian matrix as follows:

$$\begin{aligned} \boldsymbol{G}_{t+1} &= \boldsymbol{G}_t - \boldsymbol{X}_t^{\intercal}\boldsymbol{X}_t + \boldsymbol{X}_{t+1}^{\intercal}\boldsymbol{X}_{t+1} \\ &= \boldsymbol{G}_t - \boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t}^{\intercal}\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t} + \boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t+1}^{\intercal}\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t+1}. \end{aligned}$$

The update then becomes: $\boldsymbol{G}_{\boldsymbol{\Delta}} = \boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t+1}^{\intercal}\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t+1} - \boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t}^{\intercal}\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t}$.

Lemma 1 *Given a $|\mathcal{U}| \times |\mathcal{I}|$ user-item matrix $\boldsymbol{X}$, its Gramian matrix $\boldsymbol{G}$, and updates to $\boldsymbol{X}$; the rank of the update of the Gramian matrix $\boldsymbol{G}_{\boldsymbol{\Delta}}$ can be upper bounded by two times the number of unique, nonzero rows in $\boldsymbol{X}_{\boldsymbol{\Delta}}$:* rank$(\boldsymbol{G}_{\boldsymbol{\Delta}}) \leq 2|\mathcal{U}_{\boldsymbol{\Delta}}|$.

Proof As the rank of a matrix is defined as its number of linearly independent row or column vectors, a (possibly loose) upper bound for rank$(\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot]})$ is given by its number of nonzero rows $|\mathcal{U}_{\boldsymbol{\Delta}}|$. Consequently, the rank of the Gramian matrix has the same bound: rank$(\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot]}^{\intercal}\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot]}) \leq |\mathcal{U}_{\boldsymbol{\Delta}}|$. It is well known that the rank of the sum of two matrices is less than or equal to the sum of the ranks of the individual matrices. Bringing those together, we have that rank$(\boldsymbol{G}_{\boldsymbol{\Delta}}) \leq 2|\mathcal{U}_{\boldsymbol{\Delta}}|$. □

[3] In the SciPy package for Python, an implementation of the randomised method presented by Liberty et al. can be found under scipy.linalg.interpolative.estimate_rank (Liberty et al. 2007; Virtanen et al. 2020).

This upper bound on $\text{rank}(\boldsymbol{G}_{\boldsymbol{\Delta}})$ holds for any update to $\boldsymbol{X}$. When users in the update are disjoint of those in $\boldsymbol{X}_t$, the bound can be tightened to $|\mathcal{U}_{\boldsymbol{\Delta}}|$. For general-purpose use cases, it is not be feasible to ensure that users in the update do not appear with partial histories in previous iterations of the model. For specific applications such as session-based recommendation, however, it is common practice to train models on the session-item matrix, which satisfies this assumption by definition (Ludewig and Jannach 2018).

Lemma 2 *Given a* $|\mathcal{U}| \times |\mathcal{I}|$ *user-item matrix* $\boldsymbol{X}$*, its Gramian matrix* $\boldsymbol{G}$*, and updates to* $\boldsymbol{X}$ *that only consist of adding new rows or altering previously zero-rows; the rank of the update of the Gramian matrix* $\boldsymbol{G}_{\boldsymbol{\Delta}}$ *can be upper bounded by the number of rows being added or altered:* $\text{rank}(\boldsymbol{G}_{\boldsymbol{\Delta}}) \leq |\mathcal{U}_{\boldsymbol{\Delta}}|$.

Proof When the update only pertains to new users, this means that $\boldsymbol{X}_{[\mathcal{U}_{\boldsymbol{\Delta}},\cdot],t} = 0$, which ensures that $\text{rank}(\boldsymbol{G}_{\boldsymbol{\Delta}}) = \text{rank}(\boldsymbol{X}_{\boldsymbol{\Delta}})$. Because $\text{rank}(\boldsymbol{X}_{\boldsymbol{\Delta}})$ is bounded by $|\mathcal{U}_{\boldsymbol{\Delta}}|$ per definition, so is $\text{rank}(\boldsymbol{G}_{\boldsymbol{\Delta}})$: $\text{rank}(\boldsymbol{G}_{\boldsymbol{\Delta}}) \leq |\mathcal{U}_{\boldsymbol{\Delta}}|$. □

We have provided bounds for $\text{rank}(\boldsymbol{G}_{\boldsymbol{\Delta}})$ by focusing on the number of users that have contributed interactions in the new batch of data that we wish to include into the model. Analogously, in some settings, it might be easier to bound the number of unique items that are being interacted with. In a news recommendation setting, for example, a new batch of data might consist of only a very limited number of items (in the order of hundreds) being read by a much higher number of users (hundreds of thousands). In this case, we can straightforwardly extend Lemmas 1 and 2 to bound the rank by the number of independent *columns* in $\boldsymbol{X}$ as opposed to its *rows*. The further reasoning and results follow trivially, bounding $\text{rank}(\boldsymbol{G}_{\boldsymbol{\Delta}})$ by $2|\mathcal{I}_{\boldsymbol{\Delta}}|$ and $|\mathcal{I}_{\boldsymbol{\Delta}}|$, respectively. Whereas the original EASE$^{\text{R}}$ approach and the need to iteratively retrain would make it a poor choice for applications with possibly vast item catalogues but smaller *active* item catalogues, such as catalogues of news articles, the presented upper bounds theoretically show why DYN- EASE$^{\text{R}}$ can provide an efficient updating mechanism.

Algorithm 3 Approximate DYN- EASE$^{\text{R}}$

```
Input: $\hat{\boldsymbol{P}}_t, \mathcal{P}_{\boldsymbol{\Delta}}, \mathcal{L}_t, k$
Output: $\hat{\boldsymbol{P}}_{t+1}, \mathcal{L}_t + 1$.
$\boldsymbol{G}_{\boldsymbol{\Delta}}, \mathcal{L}_{t+1}$ = DYN- GRAM($\mathcal{P}_{\boldsymbol{\Delta}}, \mathcal{L}_t$) // (Algorithm 1)
$\Lambda, \boldsymbol{Q}$ = truncated-eigendecomposition($\boldsymbol{G}_{\boldsymbol{\Delta}}, k$)
$\hat{\boldsymbol{P}}_{t+1} = \hat{\boldsymbol{P}}_t - \hat{\boldsymbol{P}}_t \boldsymbol{Q}(\Lambda^{-1} + \boldsymbol{Q}^\intercal \hat{\boldsymbol{P}}_t \boldsymbol{Q})^{-1} \boldsymbol{Q}^\intercal \hat{\boldsymbol{P}}_t$
return $\hat{\boldsymbol{P}}_{t+1}$
```

3.4 Approximate DYN-EASE$^{\text{R}}$ updates via truncated eigendecomposition

Naturally, the rank of the update will not always be low in general recommendation use-cases. The easiest counter-example to think of is the case where we wish to include k user-item interactions that pertain to k new and unique users as well as k unique items. This will lead to a diagonal-like structure of $\boldsymbol{X}_{\boldsymbol{\Delta}}$ and $\text{rank}(\boldsymbol{X}_{\boldsymbol{\Delta}}) = k$, which is problematic for large values of k. However, it is also not hard to see that incorporating such a batch of data into our model will not affect any of our personalisation capabilities. Indeed, as EASE$^{\mathrm{R}}$ exploits signal from item co-occurrences, data where no item co-occurrences are present is practically useless, even though it is full-rank. Although this is a contrived example, it serves to illustrate that the rank of the update is not necessarily synonymous with its informational value.

In these cases, we can still resort to updating our model $\hat{\boldsymbol{P}}$ with a low-rank approximation of $\boldsymbol{G}_{\boldsymbol{\Delta}}$ without hurting the performance of the updated model. Instead of computing the rank and a full eigendecomposition of the Gramian as shown in Algorithm 2, we can choose the rank k at which we wish to truncate, and update $\hat{\boldsymbol{P}}$ with a low-rank approximation $\tilde{\boldsymbol{G}}_{\boldsymbol{\Delta}}$ instead of the real thing. The resulting algorithm is shown in Algorithm 3, and it provides a tuneable trade-off between the exactness of the acquired solution and the efficiency of incremental updates.

Interestingly, this type of approximate update is closely related to yet another extension of the SLIM paradigm: Factored Item Similarity Models (FISM) (Kabbur et al. 2013). In FISM, the similarity matrix $\boldsymbol{S}$ is modelled as the product of two lower-dimensional latent factor matrices. The resulting low-rank model is shown to be increasingly effective as the sparsity in the user-item interactions it learns from increases, highlighting that this type of approximation does not necessarily imply a decrease in recommendation accuracy. In approximate DYN- EASE$^{\mathrm{R}}$, we do not directly model the similarity matrix $\boldsymbol{S}$ as factorised, but we update $\boldsymbol{S}$ with a factorised version of the update to the Gramian $\boldsymbol{G}_{\boldsymbol{\Delta}}$. Factorised models such as FISM or approximate DYN- EASE$^{\mathrm{R}}$ also bear resemblance to models that are often used in natural language processing applications. Indeed, the well-known WORD2VEC algorithm to learn word embeddings for natural language processing applications implicitly learns to factorise a matrix holding the (shifted positive) pointwise mutual information between word-context pairs (Mikolov et al. 2013; Levy and Goldberg 2014).

Although our motivations for approximate DYN- EASE$^{\mathrm{R}}$ are rooted in improving the computational cost of exact DYN- EASE$^{\mathrm{R}}$, the advantages of transitivity that come from adopting low-rank representations can significantly impact recommendation performance as well. Imagine items $a, b, c \in \mathcal{I}$ where (a, b) and (b, c) co-occur in the training data of user histories, but (a, c) does not. Full-rank EASE$^{\mathrm{R}}$ cannot infer a correlation between a and c in such a setting, whereas low-rank models can learn a latent factor that unifies a, b and c. This explains the advantage that low-rank models have in sparse data environments. For further insights on the advantages, differences and analogies between full-rank and low-rank models, we refer the interested reader to the work of Van Balen and Goethals (2021).

As we are factorising $\boldsymbol{G}_{\boldsymbol{\Delta}}$ by its truncated eigendecomposition, we are guaranteed to end up with the optimal rank-k approximation with respect to the mean squared error between $\tilde{\boldsymbol{G}}_{\boldsymbol{\Delta}}$ and $\boldsymbol{G}_{\boldsymbol{\Delta}}$. Naturally, with the highly sparse nature of $\boldsymbol{G}_{\boldsymbol{\Delta}}$, this optimal approximation will focus on reconstructing entries with large values, and rows or columns with many nonzero values. This corresponds to focusing on the items that occur most often in the new incoming batch of user-item interactions $\mathcal{P}_{\Delta}$. Because of this, we can expect approximate DYN- EASE$^{\mathrm{R}}$ to favour recently popular items, which can give an additional performance boost in the right application areas. Nevertheless, an in-depth discussion or validation of the efficacy of factorised EASE$^{\mathrm{R}}$-like models

Table 1 Datasets we adopt throughout the experiments presented in this work, along with their source and summary statistics that describe the user-item interactions and their sparsity

Name	nnz($\mathbf{X}$)	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{I}\rvert$	$\overline{\lvert\mathcal{U}_i\rvert}$	$\overline{\lvert\mathcal{I}_u\rvert}$	Timespan (δ)
MovieLens-25M (ML-25M)	16M	162k	30k	524	96	25 years
YooChoose	10M	1.3M	28k	359	8	6 months
RetailRocket	593k	115k	49k	12	5	4 months
Adressa	39M	1.4M	54k	725	28	3 months
Microsoft News (MIND)	16M	696k	62k	266	24	5 days
SMDI	738k	10k	7k	41	31	4 months

nnz($\mathbf{X}$) denotes the number of non-zero entries in the user-item matrix

falls outside the scope of this work, as we focus on the efficiency with which the model can be updated. If the cut-off rank k is lower than the true rank of the update, approximate DYN- EASE$^{\text{R}}$ guarantees an improvement in terms of the computational complexity of the update procedure.

4 Experimental results and discussion

The goal of this section is to validate that the methods we proposed in earlier sections of this manuscript work as expected, and to investigate whether expectations grounded in theory can be substantiated with empirical observations. Concretely, the research questions we wish to answer are the following:

RQ1 Can exact DYN- EASE$^{\text{R}}$ provide more efficient model updates in comparison with iteratively retrained EASE$^{\text{R}}$?

RQ2 Can our theoretical analysis on the correlation between rank($\boldsymbol{G}_{\boldsymbol{\Delta}}$) and the runtime of DYN- EASE$^{\text{R}}$ set realistic expectations in practice?

RQ3 Do the phenomena we describe for bounding rank($\boldsymbol{G}_{\boldsymbol{\Delta}}$) occur in real-world session-based or news recommendation datasets?

RQ4 Can approximate DYN- EASE$^{\text{R}}$ provide a sensible trade-off between recommendation efficiency and effectiveness?

Table 1 shows the publicly available datasets we use throughout our experiments in an attempt to provide empirical answers to the above-mentioned research questions. The well-known MovieLens dataset (Harper and Konstan 2015) consists of explicit ratings (on a 1–5 scale) that users have given to movies, along with the time of rating. We drop ratings lower than 3.5 and treat the remainder as binary preference expressions. Additionally, we only keep users and items that appear at least 3 times throughout the dataset. This type of pre-processing is common, and ensures we are left with positive preference expressions that carry enough signal for effective personalisation (Liang et al. 2018; Beel and Brunel 2019). We take the newest and largest variant of the dataset as our starting point: MovieLens-25M. Many recommender systems applications are based on shorter browsing sessions rather than full user histories that might span years (Ludewig and Jannach 2018). As laid out in

Sect. 3.3, these set-ups can be especially amenable to our approach, as the adoption of these shorter sessions instead of longer user histories naturally decreases the rank of the update to the Gramian. We adopt two well-known datasets for session-based recommender systems: the YooChoose dataset, released in the context of the 2015 ACM RecSys Challenge (Ben-Shimon et al. 2015); and the RetailRocket dataset (Kaggle 2016). These datasets consist of implicit feedback (clicks) from users on retail products, and we compute the 3-core for users and items in the same manner we did for MovieLens-25M, after removing repeated user-item interactions. To validate our intuitions regarding DYN- EASE$^{\mathrm{R}}$ and the rank of the Gramian in news recommendation setups, we use the Adressa and Microsoft News datasets (MIND) (Gulla et al. 2017; Wu et al. 2020). These datasets contain implicit feedback inferred from browsing behaviour on news websites; we pre-process them analogously to the other datasets.

Some datasets have prohibitively large item catalogues for EASE$^{\mathrm{R}}$ to compute the inverse Gramian at once. However, the large majority of items are often at the extreme end of the so-called "long tail", only being interacted with once or twice. We prune these items to keep the EASE$^{\mathrm{R}}$ computation feasible but still highlight the advantages of DYN- EASE$^{\mathrm{R}}$.

Note that these pruning operations on rare items significantly cut down computation time for EASE$^{\mathrm{R}}$ (directly dependent on $|\mathcal{I}|$), but do not pose an unfair advantage for DYN- EASE$^{\mathrm{R}}$. Items that appear just once in the dataset blow up the size of the Gramian, but do not significantly impact the rank of the Gramian updates. Indeed, in these situations we get that $k \ll |\mathcal{I}|$, and the computational advantages of DYN- EASE$^{\mathrm{R}}$ over EASE$^{\mathrm{R}}$ become even more pronounced. We adopt such pruning as it is common practice and keeps the computational needs for reproducing our experiments reasonable. The reason we do not further explore other commonly known datasets such as the Million Song Dataset (MSD) (Bertin-Mahieux et al. 2011), is that these do not include logged timestamps that indicate *when* the user-item interactions occurred. Because of this, they are unsuited for evaluating a realistic scenario where models are incrementally retrained over time.

The final dataset we adopt is the SuperMarket Dataset with Implicit feedback (SMDI) introduced by Viniski et al. Because this dataset has a comparatively small item catalogue, the computation time for all EASE$^{\mathrm{R}}$ variants is in the order of seconds and largely dominated by variance and system overhead. We adopt the SMDI dataset to study the recommendation performance of approximate DYN- EASE$^{\mathrm{R}}$, as it exhibits a distribution shift that is largely absent in the other datasets we consider.

To foster the reproducibility of our work, all source code for the experiments we have conducted is publicly available under an open-source license at github.com/olivierjeunen/dynamic-easer/. All code is written in Python 3.7 using SciPy (Virtanen et al. 2020). Reported runtimes are wall-time as measured using an Intel Xeon processor with 14 cores. The rest of this section is structured to follow the research questions laid out above.

4.1 Efficiency of exact DYN-EASE$^{\mathrm{R}}$ (RQ1)

To verify the gains in runtime from exact DYN- EASE$^{\mathrm{R}}$ over iteratively retrained EASE$^{\mathrm{R}}$, we chronologically split the user-item interactions based on a fixed timestamp t, yielding all user-item interactions up to t, and all those after t. The Microsoft News Dataset comes with user's reading histories and clicks on shown recommendations, but the former type of interaction does not include timestamps. Because of this, we treat these historical interactions as "early data" included in the original EASE$^{\mathrm{R}}$ computation, and incorporate the timed clicks chronologically into DYN- EASE$^{\mathrm{R}}$ in the procedure described below.

We train an EASE$^{\mathrm{R}}$ model on the early batch, and log the runtime in seconds needed for this computation. This operation is repeated over 5 runs, and we report a 95% Gaussian confidence interval. As new incoming user-item interactions do not affect the dimension of the Gramian matrix that needs to be inverted, the runtime needed to compute EASE$^{\mathrm{R}}$ remains fairly constant when adding new user-interactions.

Over the newer batch of data, we employ a non-overlapping sliding window technique that chronologically generates batches of data to be included in the existing model via our proposed exact DYN- EASE$^{\mathrm{R}}$ procedure. The size of this window δ is varied to study the effects on the runtime of DYN- EASE$^{\mathrm{R}}$. Larger values of δ imply larger update batch sizes, which will often lead to an increase in rank($\boldsymbol{G}_{\boldsymbol{\Delta}}$). Naturally, when δ becomes too large, a point is reached where the overhead induced by our incremental updating method becomes prohibitively large, and it becomes favourable to fully retrain the EASE$^{\mathrm{R}}$ model. Sensible values of δ come with a restriction: when the runtime of the model update is larger than δ, this would indicate that the procedure cannot keep up with incoming data in real-time. We do not encounter this issue for any of the values of δ explored in our experiments - suggesting that DYN- EASE$^{\mathrm{R}}$ can be a good fit for various configurations.

Figure 1 visualises the resulting runtimes from the procedure laid out above, on all five considered datasets. The time for the sliding window increases over the x-axis, and runtime for varying values of δ is shown on the y-axis. The explored values of δ differ based on the dataset and use-case: for the 25-year spanning MovieLens dataset, daily updates might be sufficient; for the 3-month spanning news recommendation dataset Adressa, more frequent 5-minute updates might be more appropriate, to keep up with the highly dynamic nature of the environment.

We included values of δ that push the runtime for DYN- EASE$^{\mathrm{R}}$ up to that of EASE$^{\mathrm{R}}$ to highlight the limitations of our approach. Provided that the computing power and infrastructure is available, however, δ can be decreased to bring DYN- EASE$^{\mathrm{R}}$'s runtime into the desirable range. Note that this limitation on δ is general for online learning approaches from user-item interactions, and not specific to the methods we propose in this work.

From the runtime results, we can observe that our proposed method entails significant performance improvements compared to iterative model retraining, for a wide range of settings. Over all datasets, we observe a clear trend toward lower runtimes for shorter sliding windows and more frequent updates, as is expected from our theoretical results.

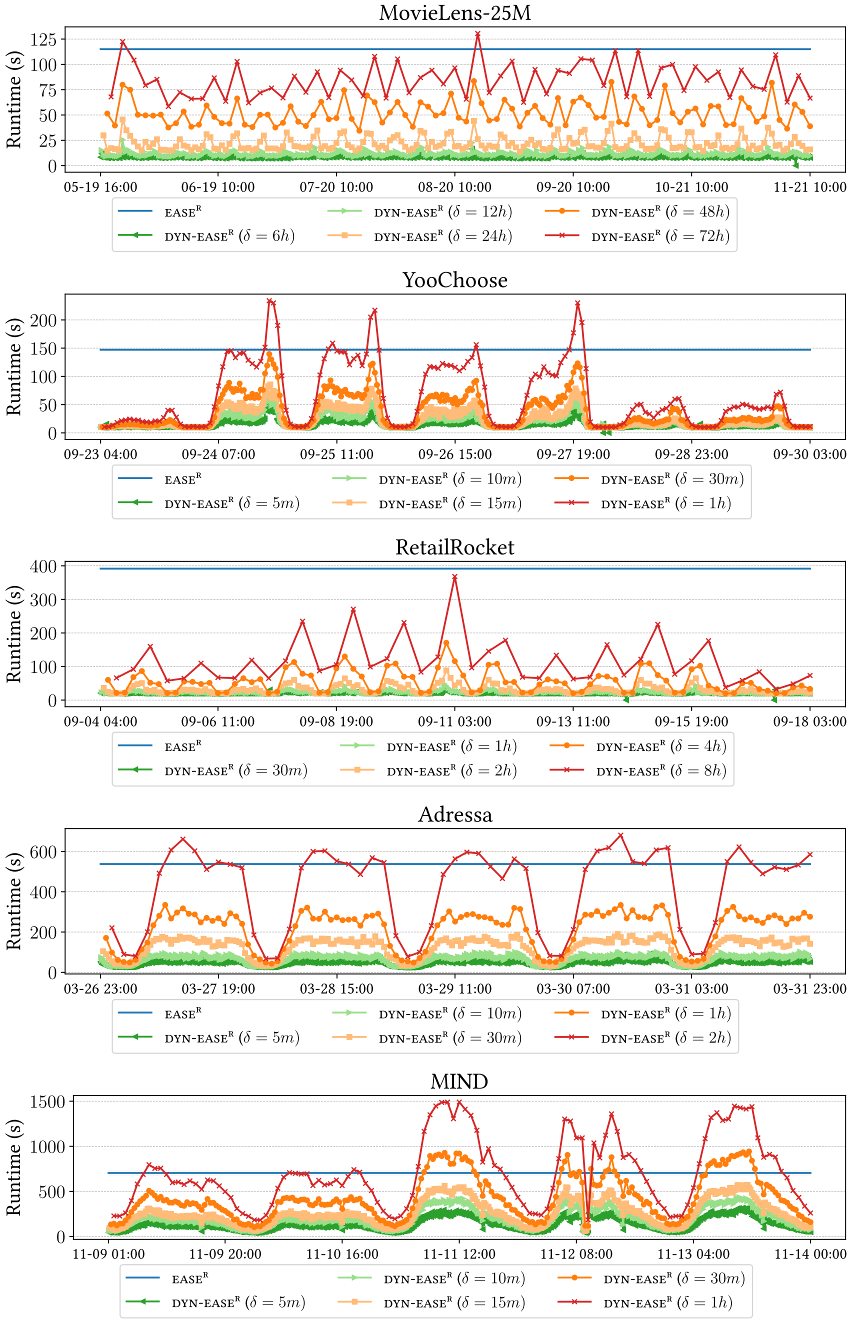

Fig. 1 Runtime results for DYN- EASE^{R} updated with different intervals (sliding window width δ), as compared to iteratively retrained EASE^{R} over the final N days of the datasets. We observe that for a wide range of interval widths δ, DYN- EASE^{R} can provide significant efficiency gains. When δ becomes too large, the overhead that comes with incremental updates becomes too high and a full retrain with vanilla EASE^{R} is favourable

As the MovieLens-25M dataset spans several decades, the amount of new user-item interactions to be incorporated on a daily basis remains modest. Exploring lower values of δ would not provide any additional insights into the performance of DYN- EASE$^{\text{R}}$ because of this. As a consequence, we obtain a clean separation between the runtime for DYN- EASE$^{\text{R}}$ on batches of different length.

The remaining four datasets represent session- and news-based recommendation environments, which are known to be much more fast-paced and dynamic. Because we focus on smaller sliding window lengths δ here, we clearly see daily seasonal patterns emerging. Indeed, DYN- EASE$^{\text{R}}$ runtime peaks coincide with peaks in website traffic. As the rank of the update is typically correlated with the number of users or items in the update, this phenomenon is to be expected. It highlights that DYN- EASE$^{\text{R}}$ is able to effectively target those model parameters that need updating, and does not spend unnecessary computing cycles on unchanged parts of the model. Note that δ does not need to be a fixed constant in real-world applications. An effective use of computing power might decrease and increase δ during traffic peaks and valleys, respectively.

4.2 Correlating rank(G_{Δ}) and runtime of exact DYN-EASE$^{\text{R}}$ (RQ2)

The runtime of the incremental updates shown in Fig. 1 is visualised against the rank of the updates in Fig. 2. We clearly observe a strong correlation between the rank of the update to the Gramian and the runtime of DYN- EASE$^{\text{R}}$, with a trend that is consistent over varying values of δ.

We fit a polynomial of the form $f(x) = a \cdot x^b + c$ on a randomly sampled subset of 90% of measurements, and assess its performance in predicting the runtime for DYN- EASE$^{\text{R}}$ based on rank($\boldsymbol{G_{\Delta}}$) on the remaining 10% of the measurements. Table 2 shows the optimal parameters, the number of samples (runtime measurements) and the root mean squared error (RMSE) on the test sample for every dataset. Figure 2 qualitatively shows that we are able to predict the runtime for DYN- EASE$^{\text{R}}$ updates with reasonable accuracy when we know the rank of the update. Combined with the bounds on this quantity laid out in Sect. 3.3, we can use this to set an expected upper bound for the computation time of our incremental updates through DYN- EASE$^{\text{R}}$. Table 2 quantitatively shows the magnitude of the errors, reassuring our qualitatively obtained insights. Note that whereas the *absolute* RMSE increases with the datasets with larger item catalogues, the *relative* error of the model remains fairly constant. Indeed, a mean error of 5 seconds on a prediction of 10 seconds is not equivalent to being 5 seconds off when the order of magnitude is 1000 seconds. These empirical observations together with the theoretical analysis presented in Sect. 3 highlight the efficiency and favourable scalability of the proposed DYN- EASE$^{\text{R}}$ procedure.

4.3 Analysing bounds for rank(G_{Δ}) (RQ3)

Figure 3 shows the rank of the incremental updates from Fig. 1 compared to summary statistics for the batches of user-item interactions. This visualisation shows the effectiveness of the upper bounds laid out in Sect. 3.3 in order to assess their utility and provide a better understanding of the underlying dynamics for every dataset.

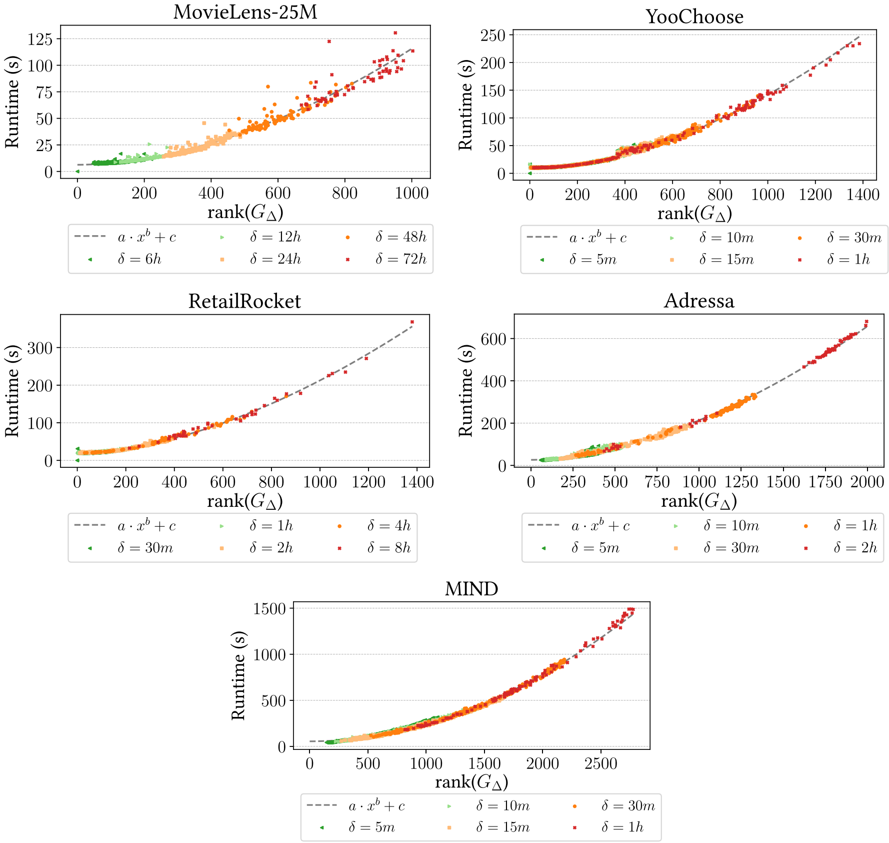

Fig. 2 Runtime for incremental updates from Fig. 1 plotted against the rank of the update to the Gramian matrix $\boldsymbol{G_\Delta}$. We observe a strong correlation between higher values of rank($\boldsymbol{G_\Delta}$) and runtime, as well as a correlation between δ and higher rank($\boldsymbol{G_\Delta}$). This result highlights that rank($\boldsymbol{G_\Delta}$) is the main driving factor between DYN- EASE$^{\mathrm{R}}$'s computational complexity, and that bounding it can give realistic expectations for DYN- EASE$^{\mathrm{R}}$ efficiency in practice

Table 2 Resulting polynomial model to predict runtime from rank($\boldsymbol{G_\Delta}$), along with the root mean squared error (RMSE) it attains and the number of observations N it was fitted on

Dataset	RMSE	N	a	b	c
MovieLens-25M (ML-25M)	2.34	1 457	3.59e−4	1.83	6.28
YooChoose	2.01	4 200	5.78e−4	1.79	9.01
RetailRocket	2.11	1 302	1.43e−3	1.72	18.81
Adressa	5.88	2 580	1.03e−3	1.75	26.35
Microsoft News (MIND)	8.77	3 000	5.98e−3	1.52	28.32

We observe that the models attain good performance in terms of RMSE, indicating that they can set realistic expectations for DYN- EASE$^{\mathrm{R}}$ runtime. Furthermore, the exponent b in the model is lower than quadratic, indicating good scaling properties for DYN- EASE$^{\mathrm{R}}$ with respect to rank($\boldsymbol{G_\Delta}$)

We observe that both for general purpose MovieLens-25M and the session-based datasets, the user-focused bound performs reasonably well in approximating the rank of the update to the Gramian. This is in line with our theoretical expectations, and confirms that the number of unique users in any given batch of user-item interactions are the main driving factor for rank($\boldsymbol{G_\Delta}$). We further see that the upper bound becomes looser as the number of unique users grows. This as well is expected behaviour, as it becomes less likely for new users' behaviour to be linearly independent of other users in the batch as the batch size grows. As mentioned in 3.3, the upper bound of $2|\mathcal{U}|$ could be tightened to $|\mathcal{U}|$ if we did not perform a hard split on time but rather divided user sessions into a "finished" and "ongoing" set. This phenomenon occurs naturally for the YooChoose dataset, where we clearly see that the $2|\mathcal{U}|$ bound is much looser. Note that the tight bounds for MovieLens might change if this dataset would include timestamps for item consumption rather than rating, as the majority of users might watch a smaller set of current series or movies. Such a recency bias would decrease the *active* item catalogue, favouring the item-based bounds.

In contrast with the user-focused datasets, the bound on the number of unique items is much tighter for the news datasets, providing an almost perfect approximation in many cases. This confirms our intuition that the rank of the update in these settings is fully determined by the number active items in the catalogue and virtually independent of the number of users or interactions in a given batch. This in turn makes these environments especially amenable to our DYN- EASE$^{\mathrm{R}}$ approach.

The number of unique users or items in a batch can give rise to reasonably tight upper bounds on the rank of the update in realistic scenarios, using real-world datasets. The absolute number of user-item interactions $|\mathcal{P}|$ provides another (impractical) bound on the rank of the update; indeed, in a worst-case scenario, every user-item interaction would pertain to a unique user and a unique item. We include the visualisation of the relation between rank($\boldsymbol{G_\Delta}$) and $|\mathcal{P}|$ to intuitively show that our proposed approach scales favourably with respect to the size of the data, a property that is most appreciated in highly dynamic environments with ever-growing dataset sizes.

4.4 Efficiency and effectiveness of approximate DYN-EASE$^{\mathrm{R}}$ (RQ4)

Finally, we wish to validate the efficiency and efficacy of approximate updates to EASE$^{\mathrm{R}}$-like models. Specifically, we wish to understand the trade-off between runtime and recall for models that are iteratively updated as new data comes in. We report experimental results for runtime and recommendation accuracy for both the Adressa and SMDI datasets. This experiment is not repeated on the other datasets, as they do not favour this type of experimental evaluation procedure. MovieLens-25M spans a too long time period, and we observe insignificant effects of model retraining on recommendation accuracy. YooChoose and RetailRocket focus on shorter user sessions, which are also unfavourable for SW-EVAL to reach statistically significant conclusions. Lastly, the Microsoft News Dataset contains bandit feedback, which is different to the organic user-item interactions we tackle in this work. This was not an issue when evaluating models' computational cost, but

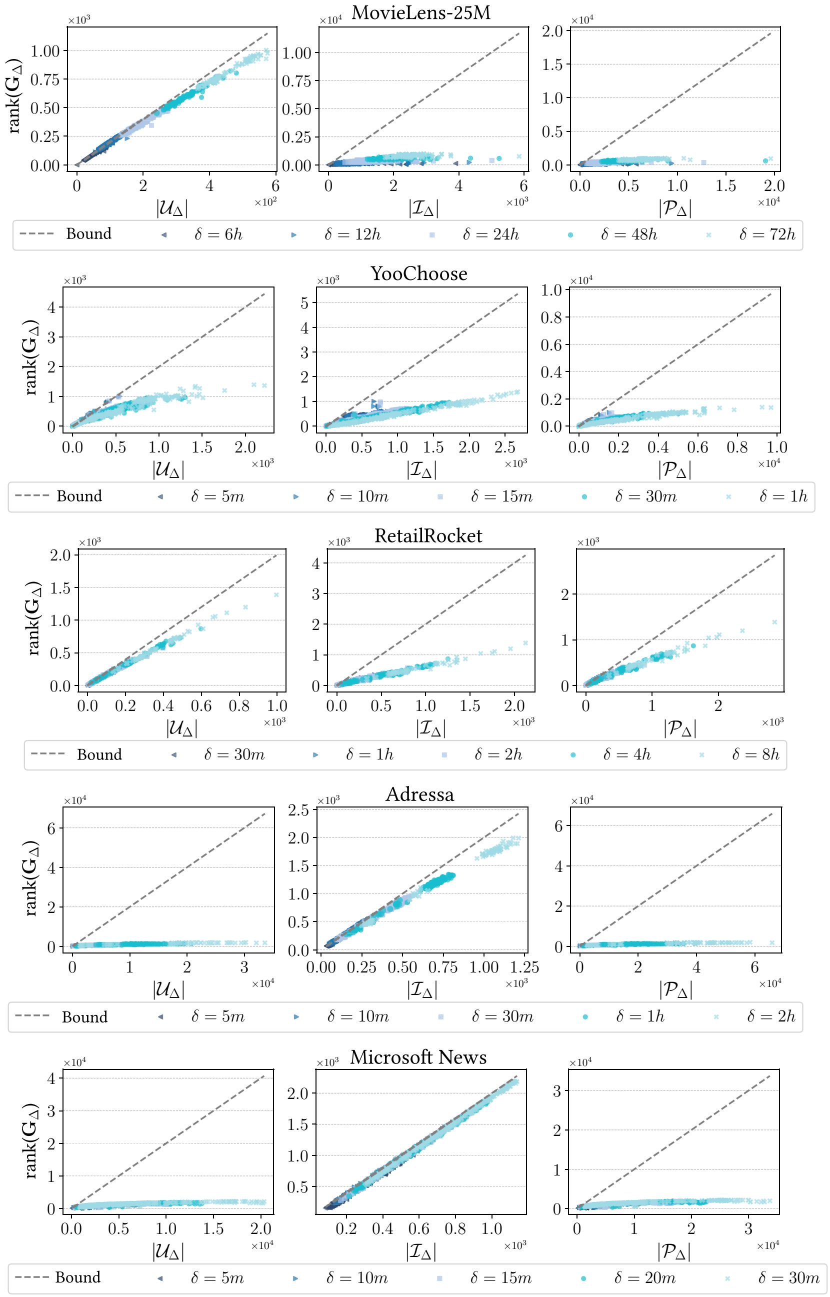

Fig. 3 Upper bounds for the batches of incremental updates from Fig. 1 plotted against the rank of the update to the Gramian matrix $\boldsymbol{G_\Delta}$. We observe that different applications that imply different data characteristics bound the rank of the update in different ways. These results confirm our expectations that where the user-focused upper bound is a good approximator for the MovieLens and RetailRocket datasets, the item-focused bound is tighter in news recommendation settings. Moreover, we observe favourable behaviour for the rank of the update in function of the number of interactions, further highlighting DYN- EASE$^{\mathrm{R}}$'s scalability

is prohibitive to properly evaluate recommendation accuracy in a common manner.

To illustrate the advantages of approximate DYN- EASE^{R}, we make use of the Sliding Window Evaluation (SW-EVAL) technique (Jeunen et al. 2018; Jeunen 2019). We train a model on all user-item interactions that have occurred up to time t. For a fixed sliding window width δ_{update}, we periodically update the model with new incoming data, both for the exact and approximate DYN- EASE^{R} variants. A concurrent sliding window with width δ_{eval} dictates the evaluation period where every competing model is evaluated on its ability to predict with which items users interacted with next. This experimental procedure is formalised in Algorithm 4. We set $\delta_{\text{update}} = 60\text{min}$ and $\delta_{\text{eval}} = 120\text{min}$ for the Adressa dataset and evaluate over the final 24 hours, and $\delta_{\text{update}} = 6\text{h}$ and $\delta_{\text{eval}} = 3\text{d}$ for the final 120 days of the SMDI dataset. This difference in order of magnitude is to keep the overall runtime of the experiments reasonable, and to ensure statistically significant results from sufficiently large evaluation sample sizes.

4.4.1 Computation time for approximate DYN-EASE^{R}

Figure 4 shows computation time for exact DYN- EASE^{R}, as well as several approximate model variants with varying cut-off ranks k. In terms of runtime improvements, we observe very favourable results for approximate updates. As is expected, the computational cost of DYN- EASE^{R}'s updates can largely be attributed to the computation of all eigen-pairs, and limiting the rank has a significant impact on the efficiency of said updates. At cut-off rank $k = 250$, the computational cost for the updates is decreased by a factor 3 or 65%.

As we have mentioned above, the computation time for all EASE^{R} variants on the SMDI dataset is in the order of seconds and largely dominated by variance and system overhead. As a result, runtime results on this dataset do not provide significant insights, and we do not report them.

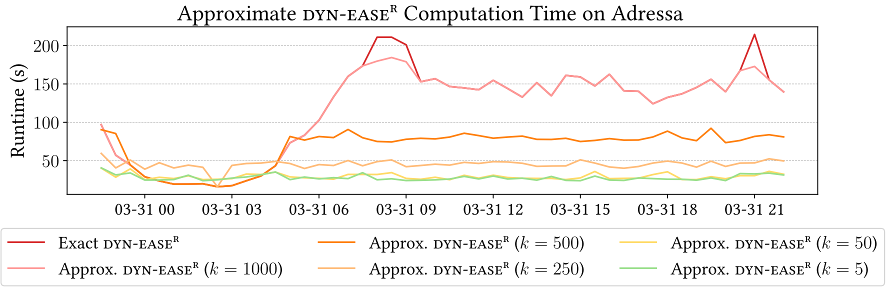

Fig. 4 Runtime results for exact and approximate DYN- EASE$^{\mathrm{R}}$ variants, with varying cut-off ranks k. We observe a quick and steady decline in computation time needed for lower values of k, which can be attributed to less computation time spent finding eigen-pairs

Algorithm 4 Sliding Window Evaluation Procedure

```
Input: Pageviews $\mathcal{P}$, evaluation period timestamps $(t, t_{\max})$, update intervals $\delta_{\text{update}}$, evaluation sliding
    window width $\delta_{\text{eval}}$, list $K$ of cut-off ranks $k$ to consider.
Output: Recommendation accuracy measurements $\mathcal{R}$.
$\boldsymbol{P}_t := \text{EASE}^{\text{R}}(\mathcal{P}_t)$
for $k \in K$ do
    $\boldsymbol{P}_{k,t} := \boldsymbol{P}_t$
$t' := t + \delta_{\text{update}}$
while $t' < t_{\max}$ do
    $\boldsymbol{P}_{t'} := \text{EXACT DYN- EASE}^{\text{R}}(\boldsymbol{P}_{t'-\delta_{\text{update}}}, \mathcal{P}_{(t'-\delta_{\text{update}}, t')})$
    for $k \in K$ do
        $\boldsymbol{P}_{k,t'} := \text{APPROXIMATE DYN- EASE}^{\text{R}}(\boldsymbol{P}_{k,t'-\delta_{\text{update}}}, \mathcal{P}_{(t'-\delta_{\text{update}}, t')}, k)$
    if $(t' - t) \mod \delta_{\text{eval}} = 0$ then
        $\mathcal{R} \leftarrow \text{SW- EVAL}(\boldsymbol{P}_t, \boldsymbol{P}_{t'}, \boldsymbol{P}_{k,t'}, \mathcal{P}_{(t', t'+\delta_{\text{eval}})})$
    $t' := t' + \delta_{\text{update}}$
return $\mathcal{R}$
```

4.4.2 Recommendation accuracy for approximate DYN-EASE$^{\mathrm{R}}$

Figure 5 visualises Recall@K for $K \in \{1, 5, 10, 20\}$ over time on the Adressa and SMDI datasets. The SMDI dataset has large variance on the number of active users over time, which heavily influences the statistical significance of some of the evaluation results, as they are based on insufficient samples. We do not include evaluation results where the evaluation set consisted of less than 100 users. The denominator for our Recall measure is $\min(K, |\mathcal{I}_u^{\text{test}}|)$ instead of the number of held-out items $|\mathcal{I}_u^{\text{test}}|$, to ensure that a perfect value of 1 can be attained at all cut-offs k.

Additional to several approximate model variants, we include recommendation accuracy results for a model that is not updated over time, yielding a lower bound on the accuracy we can expect from approximate updates. On the Adressa dataset, we observe that such a model quickly deteriorates over time. This is to be expected, as the Adressa dataset and news recommendation application are heavily biased toward recent items and interactions. This bias is less clear on the SMDI dataset at lower cut-

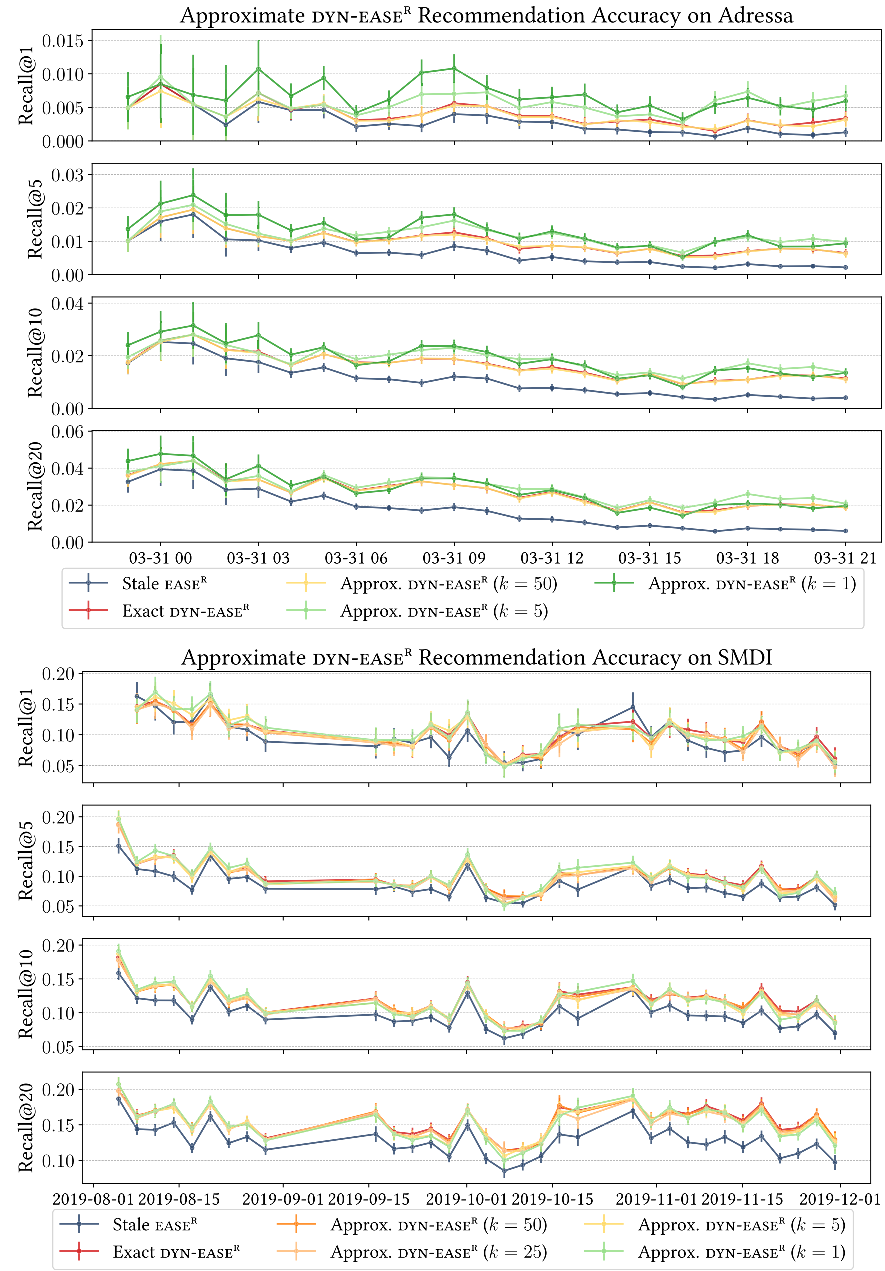

Fig. 5 Recommendation accuracy results for exact and approximate DYN- EASE^{R} variants, with varying cut-off ranks k. We additionally report results for a stale EASE^{R} model that does not incorporate new user-item data over time. We observe that exact (DYN-)EASE^{R}'s recommendation accuracy can largely be retained by approximate variants , as the recommendation accuracy measurements remain within statistical noise of one another for a large range of cut-off ranks k. For $k = 5$, we observe a statistically significant improvement as time goes on. For $k = 1$, we observe a decrease in recommendation accuracy with respect to the exact model

Table 3 Runtime and recommendation accuracy measurements on the Adressa and SMDI datasets, for the experimental procedure laid out in Algorithm 4

Dataset	Algorithm	k	Runtime (s)	Δ %	R@1 (%)	Δ %	R@5 (%)	Δ %	R@10 (%)	Δ %	R@20 (%)	Δ %
Adressa	EASE^R (*stale*)		–	–	0.30	−25.6%	0.67	−33.0%	1.07	−34.0%	1.73	−36.0%
	DYN- EASE^R (*exact*)		118s	0%	0.40	0%	1.00	0%	1.64	0%	2.72	0%
	DYN- EASE^R (*approx.*)	1000	116s	−2%	0.4	0%	1.00	0%	1.64	0%	2.72	0%
		500	68s	−43%	0.40	0%	1.00	0%	1.64	0%	2.73	0%
		250	45s	−62%	0.40	0%	1.00	0%	1.64	0%	2.73	0%
		50	29s	−75%	0.39	−3.9%	0.99	−0.8%	1.62	−0.9%	2.71	−0.7%
		5	28s	−76%	0.56	+38.9%	1.21	+21.0%	1.87	+14.0%	2.96	+7.7%
		1	27s	−77%	0.67	+67.0%	1.30	+30.0%	1.90	+16.0%	2.87	+5.3%
SMDI	EASE^R (*stale*)		–	–	9.57	-6.8%	8.52	−16.0%	9.87	-16.0%	12.73	−18.0%
	DYN- EASE^R (*exact*)		6s	–	10.27	0%	10.13	0%	11.68	0%	15.53	0%
	DYN- EASE^R (*approx.*)	100	–	–	10.20	−0.7%	10.14	+0.1%	11.64	−0.4%	15.51	−0.1%
		50	–	–	9.96	−3.0%	10.01	−1.0%	11.56	−1.0%	15.42	−0.7%
		25	–	–	9.79	−4.6%	9.90	−2.3%	11.46	−1.9%	15.25	−1.8%
		5	–	–	10.28	+0.1%	10.01	−1.2%	11.52	−1.4%	15.28	−1.4%
		1	–	–	10.39	+1.2%	10.21	+0.8%	11.60	−0.7%	15.19	−2.2%

The R@K column shows measurements for Recall@K as described in the text. We compare exact DYN- EASE^R with approximate variants at varying cut-off ranks k, and observe favourable trade-offs off ranks K, but clearly manifests itself for Recall@20 near the end of the evaluation period.

For both datasets, we observe that the accuracy of approximate DYN- EASE^{R} variants for high values of k, is statistically insignificantly different from exact DYN- EASE^{R}. This highlights that the N-fold improvement in terms of computational cost can come with a negligible impact on recommendation accuracy, showing the advantages of approximate computations. For Adressa, we observe a statistically significant improvement over exact DYN- EASE^{R} for low values of k. This can be attributed to the reasons laid out in Sect. 3, as the low-rank approximation handles sparsity, transitivity, and favours recently popular items. These model characteristics are highly favourable in news recommendation settings—but might have smaller influence on supermarket data. Nevertheless, the results highlight that many efficient low-rank updates can yield highly competitive models compared to more costly full-rank updates.

All runtime and recommendation accuracy measurements are aggregated in Table 3, providing further insights on the trade-off between runtime and recommendation accuracy for approximate DYN- EASE^{R}. We denote the Recall@K measure as R@K for improved spacing. On the SMDI dataset, the differences in recommendation accuracy among exactly or approximately update model variants were not found to be statistically significant at the $p = 0.05$ level. The differences between the stale EASE^{R} and DYN- EASE^{R} models are significant.

The size of the Adressa dataset yields more statistical power, and both the differences between stale EASE^{R} and DYN- EASE^{R} and those between exact DYN- EASE^{R} and approximate DYN- EASE^{R} with $k \in \{1, 5\}$ were found to be statistically significant.

5 Conclusion

Linear item-based models are an attractive choice for many collaborative filtering tasks due to their conceptual simplicity, interpretability, and recommendation accuracy. Recent work has shown that the analytical solution that is available for ridge regression can significantly improve the scalability of such methods, with a state-of-the-art algorithm called EASE^{R} (Steck 2019a). EASE^{R} consists of a single matrix inversion of the Gramian. As its computational complexity does not rely on the number of users or even the number of user-item interactions in the training set, it is particularly well suited to use-cases with many users or interactions, with the sole constraint that the size of the item catalogue is limited.

When deployed in real-world applications, models often need to be periodically recomputed to incorporate new data and account for newly available items and shifting user preferences, as well as general concept drift. Iteratively retraining an EASE^{R}-like model from scratch puts additional strain on such real-world applications, putting a hard upper limit on the frequency of model updates that can be attained, and possibly driving up computational costs. This especially limits the application of EASE^{R} in domains where item recency is an important factor deciding on item relevance—such as in retail or news recommendation.

In this work, we propose a novel and exact updating algorithm for embarrassingly shallow auto-encoders that combines parts of the Dynamic Index algorithm (Jeunen et al. 2019) and the Woodbury matrix identity (Hager 1989): Dynamic EASE^{R} (DYN- EASE^{R}). We have provided a thorough theoretical analysis of our proposed approach, highlighting in which cases it can provide a considerable advantage over iteratively retrained EASE^{R}, and in which cases it does not. These theoretical insights are corroborated by empirical insights from extensive experiments, showing that DYN- EASE^{R} is well suited for efficient and effective online collaborative filtering in various real-world applications.

DYN- EASE^{R} exploits the sparse and symmetric structure of the Gramian to efficiently compute the eigendecomposition of the Gramian update. When the rank of the update is large, however, this operation can still become prohibitively expensive. To mitigate this problem, we have additionally proposed an approximate DYN- EASE^{R} variant that uses a low-rank approximation of the Gramian update as opposed to its exact decomposition. Empirical results highlight further efficiency improvements at a small cost for recommendation accuracy. Our work broadens the scope of problems for which item-based models based on ridge regression are an appropriate choice in practice. To foster the reproducibility of our work, the source code for all our experiments is publicly available under an open-source license at github.com/olivierjeunen/dynamic-easer.

A promising area for future work is to further improve DYN- EASE^{R}'s computational efficiency by looking at alternative (approximate) matrix decompositions that exploit efficient random sampling (Halko et al. 2011; Martinsson et al. 2011), as the bottleneck of our current approach lies in the computation of the exact eigendecomposition of the update to the Gramian. Furthermore, we would like to explore applications of our efficient incremental updating scheme to more general multi-label regression tasks beyond the collaborative filtering use-case we tackle in this work.

Acknowledgements This work received funding from the Flemish Government (AI Research Programme).

References

Alman, J., Vassilevska, V. W.: A refined laser method and faster matrix multiplication. In Proc. of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '21. Society for Industrial and Applied Mathematics, (2021)

Anyosa, S.C., Vinagre, J., Jorge, A.M.: Incremental matrix co-factorization for recommender systems with implicit feedback. In Companion Proceedings of the The Web Conference 2018, WWW '18, page 1413–1418. International World Wide Web Conferences Steering Committee, (2018). ISBN 9781450356404

Beel, J., Brunel, V.: Data pruning in recommender systems research: Best practice or malpractice? In Proceedings of the 13th ACM Conference on Recommender Systems, RecSys '19 (2019)

Ben-Shimon, D., Tsikinovsky, A., Friedmann, M., Shapira, B., Rokach, L., Hoerle, J.: Recsys challenge 2015 and the yoochoose dataset. In: Proceedings of the 9th ACM Conference on Recommender Systems, RecSys '15, pp. 357–358. ACM (2015)

Bertin-Mahieux, T., Ellis, D.P.W., Whitman, B., Lamere, P.: The million song dataset. In: Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR '11 (2011)

Borchers, A., Herlocker, J., Konstan, J., Riedl, J.: Ganging up on information overload. Computer **31**(4), 106–108 (1998). ISSN 0018-9162

Castells, P., Hurley, N.J., Vargas, S.: Novelty and Diversity in Recommender Systems, pp. 881–918. Springer, US (2015)

Chen, B., Liu, Z.: Lifelong machine learning. Synth. Lect. Artif. Intel. Mach. Learn. **12**(3), 1–207 (2018)

Chen, Y., Wang, Y., Zhao, X., Zou, J., de Rijke, M.: Block-aware item similarity models for top-n recommendation. ACM Trans. Inf. Syst. **38**(4), 1–26 (2020)

Christakopoulou, E., Karypis, G.: Hoslim: higher-order sparse linear method for top-n recommender systems. In: Advances in Knowledge Discovery and Data Mining, pp. 38–49. Springer, New York (2014)

Christakopoulou, E., Karypis, G.: Local item-item models for top-n recommendation. In: Proceedings of the 10th ACM Conference on Recommender Systems, RecSys ’16, pp. 67–74. ACM (2016). ISBN 978-1-4503-4035-9

Dacrema, M.F., Cremonesi, P., Jannach, D.: Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In: Proceedings of the 13th ACM Conference on Recommender Systems, RecSys ’19, pp. 101–109. ACM, (2019). ISBN 978-1-4503-6243-6

Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. ACM Trans. Inf. Syst. **22**(1), 143–177 (2004)

Ekstrand, M.D., Riedl, J.T., Konstan, J.A.: Collaborative filtering recommender systems. Found. Trends Hum. Comput. Interact. **4**(2), 81–173 (2011). ISSN 1551-3955

Elahi, E., Wang, W., Ray, D., Fenton, A., Jebara, T.: Variational low rank multinomials for collaborative filtering with side-information. In: Proceedings of the 13th ACM Conference on Recommender Systems, RecSys ’19, pp. 340–347. ACM (2019). ISBN 978-1-4503-6243-6

Ferreira, E.J., Enembreck, F., Barddal, J.P.: Adadrift: An adaptive learning technique for long-history stream-based recommender systems. In: Proceedings of the 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 2593–2600 (2020)

Gama, I., Žliobaitė, J., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46**(4), (2014). ISSN 0360-0300

Gulla, J.A., Zhang, L., Liu, P., Özgöbek, Ö., Su, X.: The adressa dataset for news recommendation. In Proceedings of the International Conference on Web Intelligence, WI ’17, pp. 1042–1048. Association for Computing Machinery, (2017). ISBN 9781450349512

Hager, W.W.: Updating the inverse of a matrix. SIAM Rev. **31**(2), 221–239 (1989)

Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. **53**(2), 217–288 (2011). https://doi.org/10.1137/090771806

Harper, F.M., Konstan, J.A.: The movielens datasets: history and context. ACM Trans. Interact. Intel. Syst. **5(4):19:1–19:19**, 19:1-19:19 (2015)

He, X., Zhang, H., Kan, M., Chua, T.: Fast matrix factorization for online recommendation with implicit feedback. In: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’16, pp. 549–558. ACM (2016)

Jeunen, O.: Revisiting offline evaluation for implicit-feedback recommender systems. In: Proceedings of the 13th ACM Conference on Recommender Systems, RecSys ’19, pp. 596–600. ACM, (2019). ISBN 978-1-4503-6243-6

Jeunen, O., Goethals, B.: Pessimistic reward models for off-policy learning in recommendation. In: Proceedings of the 15th ACM Conference on Recommender Systems, RecSys ’21 (2021)

Jeunen, O., Verstrepen, K., Goethals, B.: Fair offline evaluation methodologies for implicit-feedback recommender systems with mnar data. In: Proceedings of the REVEAL 18 Workshop on Offline Evaluation for Recommender Systems (RecSys ’18), October (2018)

Jeunen, O., Verstrepen, K., Goethals, B.: Efficient similarity computation for collaborative filtering in dynamic environments. In: Proceedings of the 13th ACM Conference on Recommender Systems, RecSys ’19, pp. D251–259. ACM, (2019). ISBN 978-1-4503-6243-6

Jeunen, O., Van Balen, J., Goethals, B.: Closed-form models for collaborative filtering with side-information. In: Fourteenth ACM Conference on Recommender Systems, RecSys ’20, pp. 651–656, (2020). ISBN 9781450375832

Kabbur, S., Ning, X., Karypis, G.: Fism: Factored item similarity models for top-n recommender systems. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’13, pp. 659–667, (2013). ISBN 9781450321747

Kaggle. RetailRocket Recommender System Dataset, 2016. URL https://www.kaggle.com/retailrocket/ecommerce-dataset

Khawar, F., Poon, L., Zhang, N.L.: Learning the structure of auto-encoding recommenders. In: Proceedigns of The Web Conference 2020, WWW ’20, pp. 519–529. ACM (2020)

Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’08, pp. 426–434. Association for Computing Machinery, (2008). ISBN 9781605581934

Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, (2009). ISSN 0018-9162

Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. Natl. Bur. Stand. **45**, 255–282 (1950)

Le Gall, F.: Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ISSAC '14, pp. 296–303. Association for Computing Machinery (2014)

Lehoucq, R.B., Sorensen, D. C., Yang, C.: ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods. SIAM (1998)

Levy, M., Jack, K.: Efficient top-n recommendation by linear regression. In: RecSys Large Scale Recommender Systems Workshop (2013)

Levy, O., Goldberg, Y.: Neural word embedding as implicit matrix factorization. Adv. Neural Inform. Process. Syst. **27**, 2177–2185 (2014)

Liang, D., Krishnan, R.G., Hoffman, M.D., Jebara, T.: Variational autoencoders for collaborative filtering. In: Proceedings of the 2018 World Wide Web Conference, WWW '18, pp. 689–698. International World Wide Web Conferences Steering Committee, ACM (2018)

Liberty, E., Woolfe, F., Martinsson, P., Rokhlin, V., Tygert, M.: Randomized algorithms for the low-rank approximation of matrices. Proc. Natl. Acad. Sci. **104**(51), 20167–20172 (2007)

Ludewig, M., Jannach, D.: Evaluation of session-based recommendation algorithms. User Model. User-Adap. Inter. **28**(4), 331–390 (2018)

Martinsson, P.G., Rokhlin, V., Tygert, M.: A randomized algorithm for the decomposition of matrices. Appl. Comput. Harmon. Anal. **30**(1), 47–68 (2011). ISSN 1063-5203

Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A.M., Gama, J.: Forgetting techniques for stream-based matrix factorization in recommender systems. Knowl. Inf. Syst. **55**(2), 275–304 (2018). https://doi.org/10.1007/s10115-017-1091-8

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. Adv. Neural Inform. Process. Syst. **26**, 3111–3119 (2013)

Ning, X., Karypis, G.: Slim: Sparse linear methods for top-n recommender systems. In: Proceedings of the 2011 IEEE 11th International Conference on Data Mining, ICDM '11, pp. 497–506. IEEE Computer Society (2011). ISBN 978-0-7695-4408-3

Ning, X., Karypis, G.: Sparse linear methods with side information for top-n recommendations. In: Proceedings of the 6th ACM Conference on Recommender Systems, RecSys '12, pp. 155–162, (2012). ISBN 9781450312707

Ning, X., Nikolakopoulos, A.N., Shui, Z., Sharma,M., Karypis, G.: SLIM Library for Recommender Systems, (2019). URL https://github.com/KarypisLab/SLIM

Paige, C.C.: Accuracy and effectiveness of the lanczos algorithm for the symmetric eigenproblem. Linear Algebra Appl. **34**, 235–258 (1980)

Pan, V.Y., Chen, Z.Q.: The complexity of the matrix eigenproblem. In: Proceedings of the 31st Annual ACM Symposium on Theory of Computing, STOC '99, pp. 507–516. Association for Computing Machinery, (1999)

Park, Y., Tuzhilin, A.: The long tail of recommender systems and how to leverage it. In: Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys '08, pp. 11–18, (2008). URL https://doi.org/10.1145/1454008.1454012

Rendle, S.: Evaluation metrics for item recommendation under sampling (2019)

Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI '09, pp. 452–461. AUAI Press (2009)

Rendle, S., Krichene, W., Zhang, L., Anderson, J.: Neural collaborative filtering vs. matrix factorization revisited. In: Fourteenth ACM Conference on Recommender Systems, RecSys '20, pp. 240–248, (2020). ISBN 9781450375832

Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, WWW '01, pp. 285–295. ACM (2001)

Schein, A.I., Popescul, A., Ungar, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '02, pp. 253–260. ACM (2002)

Sedhain, S., Menon, A.K., Sanner, S., Braziunas, D.: On the effectiveness of linear models for one-class collaborative filtering. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI'16, pp. 229–235 (2016)

Shenbin, I., Alekseev, A., Tutubalina, E., Malykh, V., Nikolenko, S.I.: Recvae: A new variational autoencoder for top-n recommendations with implicit feedback. In: Proceedings of the 13th International Conference on Web Search and Data Mining, WSDM '20, pp. 528–536, (2020)

Shi, Y., Larson, M., Hanjalic, A.: Collaborative filtering beyond the user-item matrix: a survey of the state of the art and future challenges. ACM Comput. Surv. **47**(1) (2014)

Steck, H.: Embarrassingly shallow autoencoders for sparse data. In: The World Wide Web Conference, WWW '19, pp. 3251–3257 (2019)

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

Steck, H.: Markov random fields for collaborative filtering. Adv. Neural Inform. Process. Syst. **32**, 5473–5484 (2019)

Steck, H., Dimakopoulou, M., Riabov, N., Jebara, T.: Admm slim: Sparse recommendations for many users. In: Proceedings of the 13th International Conference on Web Search and Data Mining, WSDM '20, pp. 555–563 (2020)

Ubaru, S., Saad, Y.: Fast methods for estimating the numerical rank of large matrices. In: Proceedings of the 33rd International Conference on Machine Learning, Vol. 48, ICML'16, pp. 468–477. JMLR.org (2016)

Van Balen, J., Goethals, B.: High-dimensional sparse embeddings for collaborative filtering. In: Proceedings of the Web Conference 2021, WWW '21, pp. 575–581. ACM (2021)

Verstrepen, K., Bhaduriy, K., Cule, B., Goethals, B.: Collaborative filtering for binary, positiveonly data. SIGKDD Explor. Newsl., 19(1):1–21, (2017). ISSN 1931-0145

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

Vinagre, J., Jorge, A.M., Gama, J.: Collaborative filtering with recency-based negative feedback. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC '15, pp. 963–965, (2015). ISBN 9781450331968

Vinagre, J., Jorge, A.M., Al-Ghossein, M., Bifet, A.: ORSUM - Workshop on Online Recommender Systems and User Modeling, pp. 619–620. ACM, (2020). ISBN 9781450375832

Viniski, A.D., Barddal, J.P., de Souza Britto Jr., A., Enembreck,F., de Campos, H.V.A.: A case study of batch and incremental recommender systems in supermarket data under concept drifts and cold start. Expert Systems with Applications, 176:114890, 2021. ISSN 0957-4174

Virtanen, P., Gommers, R., Oliphant, T.E., et al.: Scipy 1.0: fundamental algorithms for scientific computing in python. Nat. Methods **17**(3), 261–272 (2020)

Wu, F., Qiao, Y., Chen, J., Wu, C., Qi, T., Lian, J., Liu, D., Xie, X., Gao, J., Wu, W., Zhou, M.: MIND: A large-scale dataset for news recommendation. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL'20, pp. 3597–3606. Association for Computational Linguistics, (2020)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Olivier Jeunen is a postdoctoral scientist at the University of Antwerp, where he received a PhD for his thesis "*Offline Approaches to Recommendation with Online Success*" in 2021. He has a track record of collaborating with prominent industrial research laboratories, and his recent work has been recognised with the ACM RecSys '21 Best Student Paper Award.

Jan Van Balen is a senior research scientist at Spotify, interested in recommender systems and music information retrieval. He has previously worked at University of Antwerp (Belgium), Apple Music, and Utrecht University (The Netherlands).

Bart Goethals is a full professor and chair of the department of computer science at the University of Antwerp in Belgium, as well as an adjunct professor at Monash University in Melbourne, Australia. His research focuses on pattern mining and recommender systems. Additionally, he co-founded and is now the CTO of university spin-off company Froomle.