2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 11-14, 2020. Toronto, Canada

ADADRIFT: An Adaptive Learning Technique for Long-history Stream-based Recommender Systems

Eduardo Ferreira José
PPGIa, Pontifícia Universidade Católica do Paraná
Curitiba, Brazil
ferreira.eduardo@pucpr.edu.br

Fabricio Enembreck
PPGIa, Pontifícia Universidade Católica do Paraná
Curitiba, Brazil
fabricio@ppgia.pucpr.br

Jean Paul Barddal
PPGIa, Pontifícia Universidade Católica do Paraná
Curitiba, Brazil
jean.barddal@ppgia.pucpr.br

***Abstract*—Adaptive recommender systems are increasingly showing their importance as profiling is a dynamic problem. Their goal is to update recommendation models as new interactions take place, thus swiftly adapting to drifts in the user's behavior and desires, and item's audience. However, existing recommendation algorithms usually do not perform well during drifts, as they take long to adapt to changes, or these updates are suboptimal since they account for all profiles' preferences equally, which is often untrue as each individual and its changes are unique. In this paper, we propose the ADADRIFT algorithm to deal with user and item-based drifts in adaptive recommender systems using personalized learning rates based on profile statistics. The experiments using stream-based recommender systems (ISGD and BRISMF) across four different datasets show that ADADRIFT surpasses ADADELTA with significant improvements in recommendation rates. The best results appear when the data streams have a long history of the users' or items' interactions and drifts become noticeable. The experimentation in this work highlight the importance of handling drifts in recommender systems.**

***Index Terms*—incremental recommender systems, adaptive learning, recommender systems, data stream mining**

I. Introduction

Given the users' daily use of online platforms, whether on e-commerce, e.g., eBay, Amazon; or entertainment, e.g., Netflix, Spotify, YouTube; the user experience is becoming more relevant every day. It is vital to provide each user what he/she is looking for quickly and efficiently, thus decreasing the time wasted and providing a better experience with the platform.

Guided by advances in data science and the enormous amount of data generated by user navigation, both researchers and practitioners devote their efforts towards user's profile identification. This task is called *profiling* [1], [2], which aims at enhancing user experience by facilitating the search for the desired products through the recommendations offered by the platform. Once user profiles are created, these can be used to make recommendations.

Massive competitions in the recommendations area, driven mainly by high funding from companies such as Netflix[1] [3] and anual conference-affiliated tasks such as RecSys Challenge[2] or WSDM Cup[3] led to the development of new algorithms and significant advances in recommender systems.

Most of the efforts on recommender systems have been devoted to batch scenarios, i.e., where models are trained once and used for recommendations. Yet, batch models possess an intrinsic problem: the lack of adaptation over time [4]. Therefore, constant retraining of new models is required to maintain competitive accuracy, especially when the user behavior and preferences change over time and new items are incorporated in the system.

Thus, there is a need for incremental recommendation models that observe user trends and adapt to them without any additional effort and under-performing periods.

It is possible to find a series of adaptive learning methods in the literature, but none of them directly addresses the problem of concept drift, as they were developed to batch scenarios and do not deal with temporal relations.

Its adaptations aim to converge the model to an optimum point in a faster and more accurate way, and many techniques are not prepared for point constantly changing, which is very characteristic of streaming environments. This optimum point is often called a concept, and a concept drift is this movement occurred when, for instance, consumers change their preferences. Also, the existing techniques for adaptive learning are not optimized to recommender systems, where there is a need to deal with a huge number of users, and adaptations need to act individually. Adaptive learning with individual approaches is not present in the classic scenarios, and just a few studies in the area actually deal with incremental recommendation

At this point, it is important to mention about sequence-aware recommender systems. As brought up in [5], the problem we target in this paper is to analyze individual trends for collaborative-filtering. This is relevant as users' interests and behavior may drift over time, new items will appear, and thus, model adaptation is required.

In this paper, we claim that existing incremental techniques are not sufficient as is as they overlook the fact that each user's needs and opinions, and each item's audience change

This work is financed by the APC - Associação Paranaense de Cultura through the PIBIC Master - Combined Degree Program.

[1]https://www.netflix.com

[2]https://recsys.acm.org/

[3]http://www.wsdm-conference.org/

at different rates, performing the profile update with the same impact for every profile. Therefore, we propose a technique for identifying user profile changes individually, thus analyzing the instability of the profile upon changes in daily interaction with items.

This paper is divided as follows. Section II brings forward related work on recommender systems, as well as both batch and incremental systems, and their assessment. Section III introduces our proposal, which is later assessed in Section IV. Finally, Section V concludes this paper and states envisioned future work.

II. Related Works

This work addresses user and item-based concept drift adaptation in stream-based recommender systems. This section is divided into two fronts, as Section II-A regards Recommender Systems and Section II-B discusses stochastic gradient descent (SGD) optimization techniques.

A. Recommender Systems

Recommender systems are tasked with presenting n items that are expected to be the most relevant to a user, and these are often categorized in [6]: (i) collaborative filtering, and (ii) content-based filtering techniques. The former consists of grouping users or items based on the ratings provided to items they interacted with, thus recommending other items that are related to the group to which the user has expressed interest previously. On the other hand, the latter techniques identify similar items without the use of ratings, usually using description, category, price, year, and other characteristics related to product content.

Collaborative filtering requires less information that is usually in the $\langle U, I, R\rangle$ format, where U is the user identifier, I is the item identifier, and R is the rating given by the U-th user to the I-th item. Some datasets are also *positive-only*, where data is represented in a binary pattern. In practice, only $\langle U, I\rangle$ is given, meaning that there has been a positive interaction between U and I, whereas negative and unknown relationships are kept aside [7]. The datasets may also include timestamps if the problem is to be addressed incrementally. In this work, we focus on timestamped data as the order in which user-item interactions take place in the real world affect the training and test steps of our model.

1) Ratings: In the literature, it is possible to find several actions by the user that can be considered ratings. In addition to an explicit rating, such as a 1 to a 5-star rating on a particular item or an upvote on a social network, there are implicit ratings that are extracted without direct user interaction with the item, such as clicking on a product, making a purchase, or even viewing a large portion of a video on a platform like YouTube[4]. All of these options should be considered when tailoring a recommendation system.

Most of the datasets available follow the *explicit* pattern, where ratings vary from 1 to 5 or 1 to 10. On the other hand, implicit rating datasets follow the positive-only pattern, i.e., only positive interactions between users and items are used. The absence of an interaction or a negative interaction are treated as equivalents. To transform a dataset from an explicit to an implicit representation, it is required to define a threshold θ such that $0 < \theta \leq R_{\text{max}}$, where R_{max} is the maximum rating possible. Therefore, the dataset is binarized by transforming all instances of $\langle U, I, R\rangle$ with $R \geq \theta$ to $\langle U, I\rangle$, while the remainder of interactions is ignored.

2) Recall@N: Due to the skewed problem found in the recommendation scenario (the vast majority of items are irrelevant to the user), metrics such as RMSE and MAE end up incorporating unnecessary information when considering predictions of items with few stars, that is, irrelevant to the user. The advantage of incremental recall (shortened to Recall@N) is to use only the relevant items, thereby improving the measurement of the recommendation system in question [8].

The Recall@N [9] is computed as follows. For each positive interaction in the dataset, in the $\langle U, I\rangle$ format:

1) 1000 items not evaluated by the user U are selected at random, and it is assumed that they will be irrelevant to the user in question;
2) All 1001 items, with 1000 randomly selected items and the item I, are ranked using a regression algorithm;
3) If the item I is in the top N items, the recommendation is counted as a hit.

Equation 1 is used to extract the recall of the model, which is simply the number of hits divided by the number of recommendations obtained with the model.

$$\text{Recall@N} = \frac{\text{\# of hits}}{\text{\# of recommendations}} \tag{1}$$

3) Batch Recommender Systems: Recommender systems are often built on top of dimensionality reduction techniques via matrix factorization. Proposed in [10], matrix factorization receives as input a hyper-parameter K, which is responsible for defining the dimension of latent factors. As depicted in Figure 1, the latent factors matrices have dimensions of $A = (U \times K)$ and $B = (I \times K)$, respectively. The first matrix represents the latent factors for users, and the second stores the latent factors for items. The algorithm also uses a parameter η that is the impact of model updates, and a parameter λ, which is a regularization value responsible for penalizing high values for attributes, bringing greater generalization of the model. Theoretically, λ can be specialized for users (λ_u) and items (λ_i), but it is commonly set with the same value for both matrices.

To identify the relevance of an item to a user, the required operation is a multiplication between vectors, as stated in Equation 2. The elements of the multiplication are highlighted in Figure 1.

$$\hat{R}_{ui} = A_u \cdot B_i^T \tag{2}$$

The initial weights for matrices A and B are randomly set according to a Gaussian distribution, and then Stochastic

[4]https://youtube.com

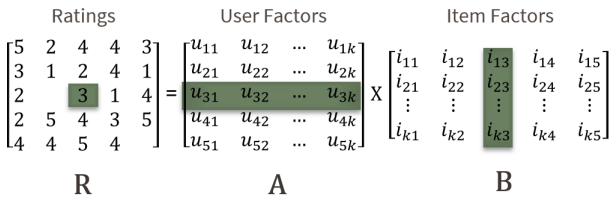

Fig. 1. Decomposition of matrix R into matrices A, referring to users; and B, referring to items. Adapted from [4].

Gradient Descent (SGD) is applied targeting minimization according to Equation 3. The first component of the equation is defined as $err_{ui} = (R_{ui} - \hat{R}_{ui})$ and returns the difference between the rating predicted by the model and the rating provided by the user (ground-truth), while the second maintains the generalization of the model and decreases the weights of the attributes. This process is applied to a set of interactions D used for batch training.

$$\min_{A,B} \sum_{(u,i)\in D} err_{ui} + \lambda(\|A_u\|^2 + \|B_i\|^2) \tag{3}$$

In each iteration, Equations 4 and 5 are used to update user and item latent vectors, respectively. At this point, it is important to emphasize the importance of η, which increases or decreases the learning pace. As described in Section III, the core of our proposal is to manipulate η effectively for drifting recommendation problems.

$$A_u \leftarrow A_u + \eta(err_{ui}B_i - \lambda A_u) \tag{4}$$
$$B_i \leftarrow B_i + \eta(err_{ui}A_u - \lambda B_i) \tag{5}$$

Additionally, it is also possible to find methods using deep learning in the literature for collaborative filtering recommendation [11]–[13]. Even though neural networks are applied to the recommendation problem, these approaches do not deal with the concept drift problem or are built on top of matrix factorization/gradient descent techniques and can still benefit from the algorithm presented in this paper.

4) Stream-based Recommender Systems: User profiles can change over time, so systems have been developed so that they can continuously learn user behavior over time. Analogous to the *concept drift* problem in data stream mining [14], new product creation, declining popularity, and changes in customer preferences are difficult to detect and usually jeopardize the performance of recommender systems [6].

The first adaptation of matrix factorization algorithms for online scenarios was *Biased Regularized Incremental Simultaneous Matrix Factorization* (BRISMF) [15]. BRISMF's main trait is that it is biased, i.e., it maintains a value per profile b_u and a value per item b_i. These values are independent of any multiplication, and these are added directly to $\hat{R}$ as represented in the Equation 6.

$$\hat{R}_{ui} = A_u \cdot B_i^T + b_u + b_i \tag{6}$$

This feature allows the recommender system to better customize the user, thus characterizing users who are most critical in ratings or only interact with items they like, thus allowing such systems to handle explicit ratings.

A variation to BRISMF called ISGD, specific for dealing with positive-only ratings, was proposed in [7]. The difference between BRISMF and ISGD is the lack of bias components, which are deemed unnecessary in this scenario.

Both BRISMF and ISGD were introduced by combining batch and incremental learning processes that encompass training and evaluation. Incremental training differs from batch training in that it is one-pass, i.e., the latent vectors are updated according to the arrival of ratings, and each interaction is assessed only once and in the other that they happen in the real world. On the other hand, incremental testing uses an incremental version of the Recall@N metric, which is given in Equation 1, yet, it is computed along sliding windows so that the performance of the recommender system can be tracked over time.

5) Recommender Systems Optimizations: Studies in the literature related to the optimization of hyperparameters for stream-based recommendation systems have been a hot topic recently, particularly two approaches proposed in recent years. Both solutions seek the ideal parameterization of the model in the batch period and use it in the stream environment. The first study [16] uses the Nelder-Mead optimization algorithm to find the best set of hyper-parameters, while the second performs with grid search for individually hyper-parameters [17].

The former, despite seeking optimization of the parameters is neither personalized nor adaptive. The parameters found are global, and they do not change during the streaming scenario. Although the latter uses detectors, it does not update the learning parameters continuously, doing it only when detectors fire.

B. Gradient Descent Optimization

Gradient descent optimization techniques are widely studied and commonly used for deep learning, being present in several known frameworks as Lasagne[5], Tensorflow[6] and Keras[7]. This great adhesion in several deep learning frameworks is due to the gain obtained when using them, which is significant for this segment.

Nevertheless, these techniques were idealized for the batch environment, i.e., optimization of the learning rate for training in a finite and well-defined set of data. The aforementioned recent adaptation to the streaming context of matrix factoring techniques, i.e., BRISMF and ISGD, poses a different problem, continuous learning, and new concepts adaptation.

It is already known that the functioning of ADAGRAD [18] converges to infinitesimal learning rates by constantly adding the error to the divisor. In the same way, other known techniques tend to converge to an optimal point [19]

[5]https://lasagne.readthedocs.io/en/latest/
[6]https://www.tensorflow.org/
[7]https://keras.io/

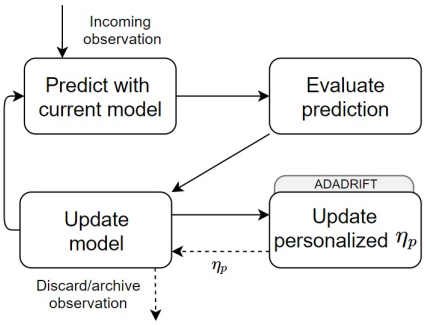

Fig. 2. Prequential Protocol when using ADADRIFT. Adapted from [4].

like RMSProp [20], Momentum [21] and Adam [22]. When converging to an optimum, these techniques will eventually take the learning rate to very low levels, failing to learn and going against the principles of data stream mining, which needs continuous training and adaptations to new situations. Due to these different issues, those techniques will not be used in this work.

ADADELTA [23] is one of the well-known optimization techniques and, although it is also developed for a batch environment, it uses a ρ decay coefficient to avoid converging to an infinitesimal point. It is possible to use it for a streaming environment since there is no convergence to a specific point, but a sliding window adaptation, that could be profitable against concept drifts. It will be used and better detailed due to this characteristic.

1) ADADELTA: ADADELTA [23] development is based on the assumption of limited learning and that the constant error sum should be windowed. The decay hyper-parameter ρ is used to perform this windowing scheme. Equation 7 performs the error accumulation, with x being the gradient error, that is err_{ui}^2.

$$\mu^{t+1} = \mu^t * \rho + (\rho - 1) * x \tag{7}$$

To make the gradient update Equations 8 and 9 are used. The learning parameter is divided by the accumulated error. ϵ has the function of avoiding an invalid value for the operation, being suggested $\epsilon = 1^{-6}$.

$$A_u \leftarrow A_u + \frac{\eta}{\sqrt{\mu^t + \epsilon}}(err_{ui}B_i - \lambda A_u) \tag{8}$$

$$B_i \leftarrow B_i + \frac{\eta}{\sqrt{\mu^t + \epsilon}}(err_{ui}A_u - \lambda B_i) \tag{9}$$

III. ADADRIFT

Recommender systems should be able to adapt to users' preferences and items' changes over time. Our major claim in this paper is that not all profiles experience change at the same pace. Some profiles are at a time of greater stabilization in a concept, while others are experiencing periods of greater change.

Given this rationale, this paper proposes the ADADRIFT, which consists of a dynamic learning mechanism that aims at keeping up with different learning rhythms, in a personalized way and is independent of the matrix factoring algorithm used, i.e., BRISMF and ISGD. As can be seen in the diagram shown in Figure 2, ADADRIFT adds just one more instantly step in the model update process, generating a personalized η_p for each user or item p. Overall, ADADRIFT demand an insignificant additional computational cost and lightweight memory consumption.

ADADRIFT has three input parameters: δ_L, δ_S and α. The α parameter controls the impact of learning and will be used later to obtain the dynamic learning rate η_p.

The δ_L and δ_S parameters define the size of two moving windows used to increase or decrease the learning rate of each profile. The relationship between those moving windows shows whether a concept drift is happening or not. They work in pairs with δ_L relative to the long-term average, while δ_S represents the short-term average. To δ_S capture short-term changes and δ_L long-term changes, it is necessary to keep always $\delta_S < \delta_L$.

After defining δ_L and δ_S values, ADADRIFT calculates moving averages μ_L and μ_S, respectively, according to the Equation 10. The μ^t represents the moving average μ at the beginning of the interaction, that is, at time t, while μ^{t+1} is the moving average at the end of the interaction, that is, at time $t + 1$. The same notation is also used in Equations 11 and 12, with variables σ and η.

The value of n_L represents the number of elements x already in the window and remains in the range $1 < n_L \leq \delta_L$, fixing at δ_L after the window has been completed. The n_S follows the same process, with δ_S as the maximum value possible. The variables n_S and n_L will compose the moving averages μ_S and μ_L, respectively.

The value x represents the profile instability over time and it is used for both moving averages μ_L and μ_S. Profile stability can be obtained by the standard deviation of the gradient already calculated in the Equations 4 and 5, represented respectively by $(err_{ui}B_i - \lambda A_u)$ and $(err_{ui}A_u - \lambda B_i)$, and used in user/item profile training. The standard deviation of the gradients difference shows how severe the update was, being directly linked also to the error obtained from the model. The greater the error, the higher the impact of the update on the user/item profile (Equations 4 and 5), and the greater the instability coefficient will be, representing a concept drift.

$$\mu^{t+1} = \frac{\mu^t * (n-1) + x}{n} \tag{10}$$

The moving average μ_L refers to the long-term average, i.e., it has less sensitivity and represents profile long-term stability. In conjunction with μ_L, σ_L is the incremental standard deviation, and it is used to define the intensity of the changes. Its update accounts for the moving average and can be observed in Equation 11.

Since $\delta_S < \delta_L$, the μ_S behavior is more unstable, seeking a balance between detecting a concept drift quickly (by the

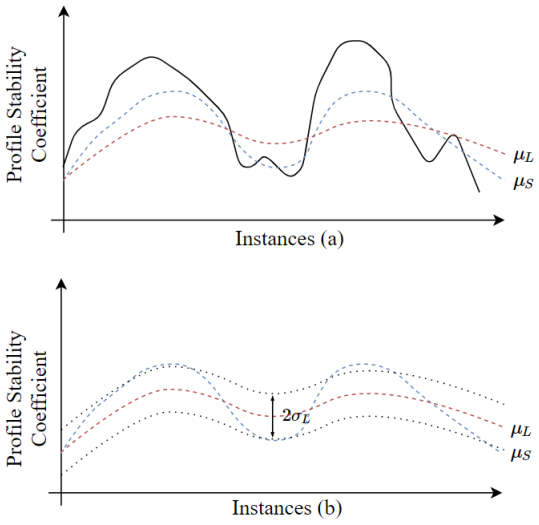

Fig. 3. Profile changes over time and the relationship with short and long term averages.

short-term moving average μ_S) and having the resilience to outliers (by the long-term moving average μ_L). This relationship between the two moving averages allows the identification of concept drifts.

$$\sigma_L^{t+1} = \frac{\sigma_L^t * (n_L - 1) + (x - \mu_L)^2}{n_L} \tag{11}$$

The behavior of this error over time can be seen in Figure 3a, where both μ_L and μ_S are represented. At times where μ_S is greater than μ_L, the dynamic learning rate will be increased proportionally with their difference, i.e., $(\mu_S - \mu_L)$, while the dynamic learning rate decreases in the opposite scenario. Equation 12 uses σ_L to observe how non-standard the recent instability coefficient pattern is, directly impacting the dynamic learning rate as the difference $(\mu_S - \mu_L)$ grows. The relationship with σ_L can be seen in Figure 3b. This change in the learning rate is calculated with Equation 12.

It is important to note that each user/item should have private moving averages, moving standard deviation, and custom learning rate because the concept drift occurs individually, not to the entire model.

$$\eta^{t+1} = \eta^t * \alpha^{[\frac{\mu_S - \mu_L}{\sigma_L}]} \tag{12}$$

The pseudo-code can be observed in the Algorithm 1. For implementation details, the variable often mentioned p, for a user, was valued as u and the user identifier, i.e., $u23$ for the user with identifier number 23; while the notation for items is $i23$ for the item with identifier number 23. Lines 2 through 4 update the values of μ_{L_p}, μ_{S_p} and σ_{L_p}. These lines use both Equations 10 and 11. It is important to note that x is a vector of the difference between each latent factor before and after the update, so the value used is the standard deviation of that difference, represented by x_σ. With these values set for this profile, Equation 12 is applied in lines 5 and 6, resulting in η_p. Soon after, the values of n_L and n_S are updated respecting the limits δ_L and δ_S respectively, and finally, in line 11, the final value of η_p is returned.

After calculating the new η_p for profile p, p is updated according to Equations 13 and 14, replacing both Equations 4 and 5. This η replacement process can be applied to all algorithms that use such a parameter, thus showing versatility and adaptability of the proposed method.

$$A_u \leftarrow A_u + \eta_p(err_{ui}B_i - \lambda A_u) \tag{13}$$
$$B_i \leftarrow B_i + \eta_p(err_{ui}A_u - \lambda B_i) \tag{14}$$

IV. Experimentation

This section details the procedures used to perform the experimentation with the algorithm, along with the actual results and analysis.

To observe the independence of the matrix factorization algorithm in ADADRIFT, experiments are carried out using both ISGD and BRISMF. Other algorithms that can be found in the literature can also be adapted for the use of ADADRIFT, but such experiments are out of the scope of this paper.

For ISGD and BRISMF to work, one must generate an initial model using an offline approach and then deploying it to a streaming environment. This process is given in Figure 4, where parts A and B are responsible for training and testing a model using Stochastic Gradient Descent (SGD) in batch

Algorithm 1: ADADRIFT

/* **Input:** $(err_{ui}B_i - \lambda A_u)$ and p */
/* **Output:** η_p */
/* **Variables:** δ_L, δ_S, α and arrays μ_L, μ_S, η, σ_L n_L, n_S */
/* **Initialization:** Arrays with size $|A| + |B|$. η is an array and each position is initialized to the default value of the matrix factorization technique. Variables μ_L, σ and μ_S do not require initialization values, and n is initialized to 1 for each profile. */

1 $x \leftarrow (err_{ui}B_i - \lambda A_u)$
2 $\mu_{L_p} \leftarrow (\mu_{L_p} * (n_{L_p} - 1) + x_\sigma)/n_{L_p}$
3 $\mu_{S_p} \leftarrow (\mu_{S_p} * (n_{S_p} - 1) + x_\sigma)/n_{S_p}$
4 $\sigma_{L_p} \leftarrow \left(\sigma_{L_p} * (n_{L_p} - 1) + (x_\sigma - \mu_{L_p})^2\right)/n_{L_p}$
5 $s \leftarrow (\mu_{S_p} - \mu_{L_p})/\sigma_{L_p}$
6 $\eta_p \leftarrow \eta_p * \alpha^s$
7 **if** $n_{L_p} < \delta_L$ **then**
8 $\quad n_{L_p} \leftarrow n_{L_p} + 1$
9 **if** $n_{S_p} < \delta_S$ **then**
10 $\quad n_{S_p} \leftarrow n_{S_p} + 1$
11 **return** η_p

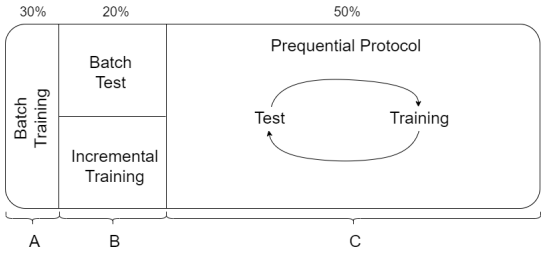

Fig. 4. The dataset was separated into three parts, the first two refer to the batch matrix generation process, while the last one is the application of the prequential protocol for validation.

mode during part A, and performing one-pass updates using the data in B.

Finally, in part C, the prequential protocol [24] is applied, which consists of predicting each datum prior to its use during a model update, always following the natural temporal order in which data is made available.

A. Datasets

Given the use of BRISMF and ISGD for this experiment, both numerical and positive-only datasets were selected, totaling two datasets per category. Considering that our objective was to observe a concept drift, it was necessary to carry out some treatments on the selected datasets to observe this problem. The concept drift, as already explained, affects both users and items. Because of this, the selection of datasets must also represent this problem. Thus, two datasets with a long history of interactions per user and two with a long history of interactions per item have been selected. It is important to keep in mind that, when referring to a profile, the rationale provided can be applied to both user and item profiles.

The datasets were purposely selected with a considerable number of instances to be pruned, whilst maintaining a significant number of ratings per profile and still a reasonable size, with around 1 million ratings. We first present the original datasets followed by their transformed variants.

For numeric ratings, the both datasets Netflix[8] [3] (NF), and Personality made available by GroupLens[9] [25] (PL) were selected. All of them are related to movie consumption, the former being from a streaming platform, and the latter with user interviewing for scientific experimentation. When it comes to positive-only datasets, we selected LastFM[10] - 1K users [26] (LFM) and converted version of the Netflix (NF5), selecting positive interactions according to the threshold $\theta = 5$. This transformation was previously detailed in Section II. More information about the original datasets is given in Table I.

Both Netflix datasets (CD-NF and CD-NF5) own a long history for some films. Therefore, there are a large number of interactions for a few items, allowing item-based concept drift to arise during the rating stream. When considering the other two datasets (CD-PL and CD-LFM), it is possible to observe a long history of interactions per user. Similarly to how item-based concept drift was treated, these datasets are used to highlight the user-based concept drift.

For the three most massive datasets (LastFM, Netflix, and Netflix GTE 5), a cut was made of the last 2 million instances. From then on, pruning was performed according to the values presented in Table II, dropping users in CD-LFM and CD-PL datasets, and items for CD-NF and CD-NF5 where the number of ratings was smaller than the pre-defined threshold. The dataset with these procedures applied, we add the CD prefix, i.e., CD-PL, CD-NF, CD-NF5, and CD-LFM. The result of this procedure can be seen in Table III. As a consequence, it was possible to obtain datasets with a large average number of ratings per user/item compared to other datasets in the literature, with the possibility of producing a concept drift due to the users and items' long history of iterations.

B. Results and Discussion

ADADRIFT has 3 parameters that require tuning: α, δ_L, δ_S. Therefore, a grid search has been conducted with the goal of determining the most suited configuration. The grid search for this search was $\alpha \in \{1.001, 1.1\}$; $(\delta_L, \delta_S) \in \{(20, 10), (100, 50), (200, 100)\}$. The number of latent factors k was set to 60, and the regularization factor $\lambda = 0.001$

[8]https://www.kaggle.com/netflix-inc/netflix-prize-data

[9]https://grouplens.org/datasets/personality-2018/

[10]http://ocelma.net/MusicRecommendationDataset/

TABLE I
ORIGINAL DATASETS DETAILS USED DURING EXPERIMENTATION.

Dataset	Size	Users	Items	Mean Iterations[11]	Sparsity
Netflix (NF)	100,480,507	480,189	17,770	5,654	98.82%
Personality (PL)	1,028,751	1,820	35,196	565	98.41%
LastFM 1K (LFM)	19,150,868	992	1,500,661	19,305	99.69%
Netflix GTE 5 (NF5)	23,168,232	463,616	17,775	1304	99,72%

TABLE II
PROCEDURE APPLIED TO EMPHASIZE CONCEPT DRIFT.

Dataset	Last 2M Slice	Prune Threshold	Prune Type	Sign
Netflix	X	1K	Item	CD-NF
Personality		150	User	CD-PL
LastFM 1K	X	5K	User	CD-LFM
Netflix GTE 5	X	1K	Item	CD-NF5

[11]Since we are looking for different types of concept drift, the average considered is users' iterations for CD-PL and CD-LFM, while average items' iterations for CD-NF and CD-NF5.

TABLE III
DATASETS WITH THE CONCEPT DRIFT TREATMENT.

Dataset	Size	Users	Items	Mean Iterations[11]	Sparsity
CD-PL	991,456	1,289	35,144	769	97,83%
CD-NF	810,046	108,269	452	1,792	98,34%
CD-NF5	1,012,145	141,132	468	2,162	98,47%
CD-LFM	1,028,379	119	260,098	8,641	98,72%

TABLE IV
RESULTS IN PERCENT GAIN AGAINST THE BASELINE MODEL IN RECALL@N WITH $N \in \{1, 5, 10, 20\}$.

Algorithm	BRISMF		ISGD		Average
	CD-PL	CD-NF	CD-NF5	CD-LFM	
Recall@1					
Base Model	$1.31 * 10^{-1}$	$5.56 * 10^{-3}$	$1.20 * 10^{-3}$	$1.07 * 10^{-2}$	-
ADADELTA	−92.68%	+105.14%	+514.15%	+27.80%	+138.60%
ADADRIFT	−4.28%	+156.91%	+401.65%	+908.22%	+365.63%
Recall@5					
Base Model	$2.57 * 10^{-1}$	$1.69 * 10^{-2}$	$3.88 * 10^{-3}$	$2.49 * 10^{-2}$	-
ADADELTA	−87.90%	+75.46%	+419.52%	+36.81%	+110.97%
ADADRIFT	−4.19%	+99.68%	+295.51%	+622.41%	+253.35%
Recall@10					
Base Model	$3.55 * 10^{-1}$	$2.96 * 10^{-2}$	$8.05 * 10^{-3}$	$4.85 * 10^{-2}$	-
ADADELTA	−83.18%	+67.58%	+331.82%	+10.25%	+81.62%
ADADRIFT	−4.30%	+71.02%	+211.25%	+346.06%	+156.01%
Recall@20					
Base Model	$4.82 * 10^{-1}$	$5.25 * 10^{-2}$	$1.77 * 10^{-2}$	$1.20 * 10^{-1}$	-
ADADELTA	−75.65%	+63.66%	+255.42%	−34.01%	+52.36%
ADADRIFT	−4.99%	+47.85%	+143.51%	+113.75%	+75.03%

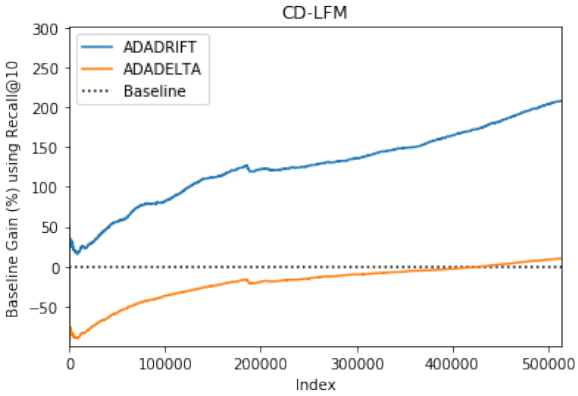

Fig. 5. The earnings of the models compared to the baseline (ISGD), measured in percentage gain from Recall@10. The dataset used was CD-LFM, which consists of the treatment for concept drift carried out in LastFM 1K users.

was set for both BRISMF and ISGD. The best globally[12] performing ADADRIFT configuration was $\alpha = 1.1$ and $(\delta_L, \delta_S) = (200, 100)$. For ADADELTA, the main hyper-parameter is the decay rate, which has been explored in the set $\rho \in \{0.99, 0.95\}$. The best result shown was $\rho = 0.99$. We used $\epsilon = 1^{-6}$, which has been recommended in the article where ADADELTA was proposed [23].

The results of the experiments were presented in Table IV, considering the percent gain against the baseline model, i.e., BRISMF and ISGD, without any other technique. This approach was used due to Recall's scale, which generally remains small and makes it difficult to measure/compare gains.

To observe the gain over time, the results obtained for Recall@10 for the CD-LFM and CD-NF5 datasets can be seen in the Figures 5 and 6 respectively. The CD-LFM dataset is the one with the highest number of average ratings per profile, having about 4 times more than the second-largest,

[12]It is important to highlight that the tuning process was carried out globally; that is, the best configuration was obtained considering all datasets. This procedure was chosen to avoid an optimism of the results, but it also implies that the result has room for improvement if the search for hyper-parameters is carried out considering the nuances of each dataset.

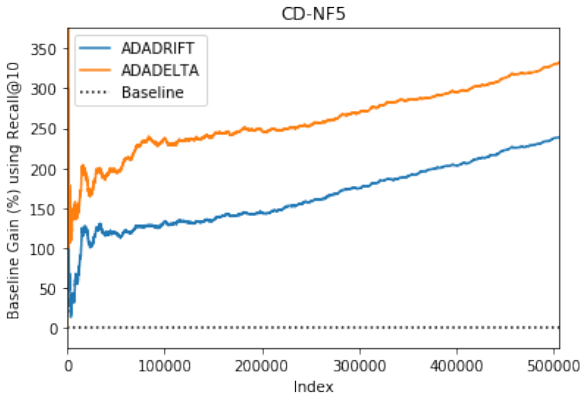

Fig. 6. The earnings of the models compared to the baseline (ISGD), measured in percentage gain from Recall@10. The dataset used was CD-NF5, which consists of the treatment for concept drift carried out in positive-only Netflix dataset.

CD-NF5. Because of this, the incidence of changes from concept tends to be higher. This characteristic showed a great impact on the results, as can be seen in Figure 5. ADADRIFT has a significant and constant gain over time compared to baseline and ADADELTA. This improvement skyrockets in the last 125,000 instances, reaching the mark of 346.06% in the last instance[13]. This growth is even greater, resulting in a performance of 908.22% for Recall@1 compared to the gain of only 27.80% when using ADADELTA, representing 32.67 times more effective. This high performance is due to the users' preferences changing and ADADRIFT adaptations, which are stacking in the final results.

As the average number of ratings per profile decreases for the datasets, there is a degradation in the gain of the algorithm. Concerning the CD-NF5 dataset, it is possible to observe an advantage of ADADELTA, although ADADRIFT still presents gains compared to the baseline and being a competitive algorithm. The numerical version of the Netflix dataset, CD-NF, has a very similar result between both algorithms, with slight advantages alternating between the metrics used. Because ADADRIFT remains competitive, it is possible to see a better result in the average gain for the Recall@N$|N \in \{1, 5, 10, 20\}$. This degradation is expected due to the decrease in the incidence of concept drifts, which can generate adaptation where it is not necessary and bring errors. The graph in Figure 6 shows similar growth for both algorithms, with ADADELTA having a constant advantage of around 100% of the ISGD performance. This difference remains throughout the course, with ADADELTA being 1.39 times more effective than ADADRIFT in the last measurement.

ADADELTA ends up degrading even more by decreasing the average number of ratings per profile, with ADADRIFT maintaining a similar result to BRISMF. The importance of

[13]Due to the functioning of the Recall previously detailed, the last point represents all hits divided by the number of elements tested. The last point should be used for measurement.

the average number of ratings is clear when observing that the ordering by average ratings is the same as the ADADRIFT's gain compared to the baseline for Recall@N$|N \in \{1, 5, 10\}$ in CD-PL, CD-NF, CD-NF5, and CD-LFM. This effect is expected due to the problem that ADADRIFT was thought to, which will appear exclusively when disposable of a long interaction history on the platform.

The performance highly correlated with the iteration history size is desired for easy decision making in which scenarios his use is profitable or not. Considering that the production scenario has a larger number of ratings per profile than the datasets present in the literature, ADADRIFT will thrive with the adaptation to concept drifts.

V. Conclusion and Future Work

This paper proposed the ADADRIFT adaptive learning technique. ADADRIFT increases the learning rate to more effectively adapt to concept drift, and decreases it during moments of concept stability, thus avoiding unrequired changes in the learning process and increased error rates. ADADRIFT was applied to different learning schemes across different datasets.

Experiments were carried out involving 4 datasets, 2 different base algorithms (ISGD and BRISMF), 2 different scenarios (user-based and item-based concept drift), and a different adaptive learning technique. ADADRIFT yielded, on average, better results due to the stability of the algorithm given by the dualistic relationship between moving averages.

ADADRIFT's performance has a high correlation with the average size of iteration history, an expected behavior since concept drifts need time to appear. ADADRIFT had the best performance in the modified dataset of LastFM 1K users (CD-LFM), which possibly had a higher incidence of concept drifts. It still performed well for the other datasets used, despite losing slightly to ADADELTA in the version with only positive interactions of Netflix dataset, CD-NF5. When using the dataset with the lowest interaction history, it was not possible to observe any gain in use, resulting in the opposite effect: a degradation when using the algorithm. Despite this, there was a slight degradation of ADADRIFT compared to a worse result for ADADELTA.

It is important to note that the problem of concept drift is not exclusively for stream-based recommender systems, but with the entire area of data stream mining. ADADRIFT can be evaluated in other types of applications that extrapolate recommender systems, i.e., whenever gradient descent techniques are applied in streaming environments such as classification and regression problems.

For future work, we expect to analyze ADADRIFT's behavior with other matrix factorization algorithms, as well as analyzing the impact of other hyper-parameters, such as the number of latent factors that could possibly impact profile change adaptation over time.

References

[1] B. Veloso, F. Leal, H. G. Vélez, B. Malheiro, and J. C. Burguillo, "Highlights 1 . Parallel data stream algorithm 2 . Scalable Recommendation Engine 3 . Profiling and Recommendation algorithms using scalable data analytics recommendations," *J. Parallel Distrib. Comput.*, 2018.

[2] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-Based Systems*, vol. 46, pp. 109–132, jul 2013.

[3] J. Bennett and S. Lanning, "The Netflix Prize," 2007.

[4] A. Jorge, V. Vinagre, João, D. Marcos, G. João, S. Carlos, P. Matuszyk, and M. Spiliopoulou, "Scalable Online Top-N Recommender Systems," vol. 278, pp. 3–20, 2017.

[5] M. Quadrana, P. Cremonesi, and D. Jannach, "Sequence-aware recommender systems," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–36, sep 2018.

[6] S. Chang, Y. Zhang, J. Tang, D. Yin, Y. Chang, M. A. Hasegawa-Johnson, and T. S. Huang, "Streaming Recommender Systems," *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, pp. 381–389, 2017.

[7] J. Vinagre, A. M. Jorge, and J. Gama, "Fast Incremental Matrix Factorization for Recommendation with Positive-Only Feedback," pp. 459–470, 2014.

[8] P. Matuszyk and M. Spiliopoulou, "Stream-based semi-supervised learning for recommender systems," *Machine Learning*, vol. 106, no. 6, pp. 771–798, 2017.

[9] P. Cremonesi, P. Milano, Y. Koren, and R. Turrin, "Performance of Recommender Algorithms on Top-N Recommendation Tasks Categories and Subject Descriptors," no. September, 2010.

[10] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl, "Application of Dimensionality Reduction in Recommender System – A Case Study," *KDD*, vol. 35, no. 4, pp. 429–443, 2001.

[11] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," 2015.

[12] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," 2018.

[13] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[14] J. Gama, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "1 A Survey on Concept Drift Adaptation ˜," vol. 1, no. 1, 2013.

[15] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Scalable Collaborative Filtering Approaches for Large Recommender Systems," *Journal of Machine Learning Research*, vol. 10, pp. 623–656, 2009.

[16] B. Veloso, J. Gama, B. Malheiro, and J. Vinagre, "Self hyper-parameter tuning for stream recommendation algorithms," in *ECML PKDD 2018 Workshops*, A. Monreale, C. Alzate, M. Kamp, Y. Krishnamurthy, D. Paurat, M. Sayed-Mouchaweh, A. Bifet, J. Gama, and R. P. Ribeiro, Eds. Cham: Springer International Publishing, 2019, pp. 91–102.

[17] B. Veloso, B. Malheiro, and J. Foss, "Stream recommendation using individual hyper-parameters," 08 2019.

[18] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[19] S. Ruder, "An overview of gradient descent optimization algorithms." 2016.

[20] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio, "Rmsprop and equilibrated adaptive learning rates for non-convex optimization." *CoRR*, vol. abs/1502.04390, 2015.

[21] N. Qian, "On the momentum term in gradient descent learning algorithms." *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[23] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1212.html#abs-1212-5701

[24] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," pp. 317–346, 2013.

[25] T. T. Nguyen, F. M. Harper, L. Terveen, and J. A. Konstan, "User personality and user satisfaction with recommender systems," *Information Systems Frontiers*, vol. 20, no. 6, pp. 1173–1189, Sep. 2017.

[26] O. Celma, *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.