Modelling Concept Drift in Dynamic Data Streams for Recommender Systems

LUCIANO CAROPRESE, University Gabriele d'Annunzio of Chieti and Pescara Department of Engineering and Geosciences, Pescara, Italy and ICAR-CNR, Rende, Italy
FRANCESCO PISANI, ICAR-CNR, Rende, Italy
BRUNO MIGUEL VELOSO, Universidade Portucalense, Porto, Portugal and INESC TEC, Porto, Portugal
MATTHIAS KONIG, Leiden University, Leiden, Netherlands
GIUSEPPE MANCO, ICAR-CNR, Rende, Italy
HOLGER HOOS, Leiden University, Leiden, Netherlands and The University of British Columbia, Vancouver, Canada
JOAO GAMA, Universidade Portucalense, Porto, Portugal and INESC TEC, Porto, Portugal

Recommendation systems play a crucial role in modern e-commerce and streaming services. However, the limited availability of public datasets hampers the rapid development of more efficient and accurate recommendation algorithms within the research community. This work introduces a stream-based data generator designed to generate user preferences for a set of items while accommodating progressive changes in user preferences. The underlying principle involves using user/item embeddings to derive preferences by exploring the proximity of these embeddings. Whether randomly generated or learned from a real finite data stream, these embeddings serve as the basis for generating new preferences. We investigate how this fundamental model can adapt to shifts in user behavior over time; in our framework, changes correspond to alterations in the structure of the tripartite graph, reflecting modifications in the underlying embeddings. Through an analysis of real-life data streams, we demonstrate that the proposed model is effective in capturing actual preferences and the changes that they can exhibit over time. Thus, we characterize these changes and develop a generalized method capable of simulating realistic data, thereby generating streams with similar yet controllable drift dynamics.

Additional Key Words and Phrases: Recommender Systems, Collaborative Filtering, Concept Drift Adaptation, Data Generation

1 Introduction

Modern machine learning applications and systems need to deal with large amounts of continuous data in the form of data streams. Most real-life application scenarios require the analysis of large volumes of data whose properties or distribution can change and evolve. Notable examples are e-commerce and streaming platforms, which contain catalogs counting thousands or even millions of products or items, where user preferences and purchasing/adoption behavior can evolve over time according to changes in the catalog. Data Stream

Authors' Contact Information: Luciano Caroprese, University Gabriele d'Annunzio of Chieti and Pescara Department of Engineering and Geosciences, Pescara, Italy and ICAR-CNR, Rende, Calabria, Italy; e-mail: luciano.caroprese@unich.it; Francesco Pisani, ICAR-CNR, Rende, Calabria, Italy; e-mail: francescosergio.pisani@icar.cnr.it; Bruno Miguel Veloso, Universidade Portucalense, Porto, Porto, Portugal and INESC TEC, Porto, Porto, Portugal; e-mail: bruno.miguel.veloso@gmail.com; Matthias Konig, Leiden University, Leiden, Netherlands; e-mail: h.m.t.konig@liacs.leidenuniv.nl; Giuseppe Manco, ICAR-CNR, Rende, Calabria, Italy; e-mail: giuseppe.manco@icar.cnr.it; Holger Hoos, Leiden University, Leiden, Netherlands and The University of British Columbia, Vancouver, British Columbia, Canada; e-mail: h.h.hoos@liacs.leidenuniv.nl; Joao Gama, Universidade Portucalense, Porto, Porto, Portugal and INESC TEC, Porto, Porto, Portugal; e-mail: jgama@fep.up.pt.

ACM 2770-6699/2024/12-ART
https://doi.org/10.1145/3707693

Mining [13, 17, 26] represents the set of techniques and methods that enable data analysis fulfilling the following characteristics: (a) High amounts of data coming in an infinite stream [26]; (b) Concept Drifting [15, 42, 47], i.e., data change or evolve over time; (c) High volatility, that prevents the possibility of storing the data and calls for online methods where data is analyzed as soon as it received and it is discarded or summarised.

Besides traditional supervised [16] or unsupervised [25] learning problems on data streams, *recommendation* can be considered an extremely significant landscape for data stream mining. The massive amount of information available to users in the form of digital catalogs and online services requires the capability to appropriately select relevant content and filter out irrelevant data. Understanding and predicting users' choices and preferences is thus a challenging task and Recommender Systems (RS) play a crucial role in modeling complex situations, where prior beliefs, specific tendencies, and reciprocal influences jointly contribute to determining such preferences [1, 5, 35].

Notably, although the field has been extensively studied and several techniques and methods were proposed in the literature, there is a significant gap between research algorithms and models, and real-life industrial deployment of recommender systems. In fact, the former are often based on the unrealistic assumption that preferences are static and do not tend to change over time. As a consequence, the studies essentially focus on static datasets collected over relatively long periods, and build preference models that tend to be periodically updated. This perspective has obviously several limitations, as it ignores the streaming nature of preference data and consequently the continuous evolution of the interactions, due either to catalog updates, user churn or new acquisitions, and evolving tastes or preferences. This problem has been recently at the center of the attention of current research [2] and some initial attempts were made to devise more sophisticated models capable of capturing the dynamic nature of preferences [11, 29, 30, 43, 45].

In this paper, we address the problem of the shortage of sufficiently large public datasets with real properties, needed to train and evaluate modern recommendation systems. When evaluating recommendation systems in a streaming scenario [44], a significant challenge arises: How can we effectively assess the robustness of advanced machine learning systems and quantify their performance in a controlled experimental benchmarking environment, where all the characteristics of the data can be appropriately tuned and the algorithmic response can be evaluated accordingly? This problem is challenging for a number of relevant reasons. The most important is being that obtaining large-scale public repositories that represent interaction datasets is difficult, due to inherent privacy and technological concerns associated with their sharing. In fact, the available datasets such as Amazon [32], Netflix [7], or MovieLens [19] are orders of magnitude smaller than proprietary data and typically contain data collected in a limited period where preferences have been collected. Synthetic data generators can help us alleviate this problem by providing ad-hoc datasets where the data characteristics (number of users/items, number of interactions, data distributions) can be manipulated and the algorithmic robustness compared according to such manipulations. However, existing data generation methods do not take into account the dynamic nature of the data distribution, which is likely to occur in a streaming scenario and in the presence of evolving preferences along the time dimension. This situation is further exacerbated by the lack of general models for distribution dynamics, which should instead be accurately studied and correlated to algorithmic responses.

In practice, synthetic data generation that accounts for drifts in a continuous streaming scenario needs to address the following challenges:

(1) First, we need to hypothesize a preference model that can realistically characterize item adoption and user preferences. This model should serve as the basis for the data generation process without introducing any bias that is not actually observable in real data.
(2) Second, we need to consider all possible changes that can cause a drift within the preferences. Drifts can be abrupt or progressive, they can involve a percentage of users/items or the whole population, and they can

be caused by new emerging trends or by changes in the characteristics of users/items. All these aspects need to be modeled, and the data generation process needs to embed them.

In order to deal with the above issues, we propose a general preference model that can easily adapt to cover preference dynamics. We focus mainly on implicit preferences: Given a user/item pair, a preference is a binary response expressing the fact that the user adopted/did not adopt the item. This simple representation scheme naturally induces a per-user ordering among items, which can, hence, be ranked according to the adoption likelihood. Our basic assumption is that both users and items map to an underlying latent low-dimensional manifold, and preferences essentially occur due to closeness within this latent space. Notably, this interaction can be made explicit by uncovering the latent space and modeling the interactions as a tripartite graph connecting users, latent space dimensions, and items. As a consequence, preference dynamics can be easily made explicit by suitable modifications of the latent embeddings, which in turn represent modifications of the underlying graph, as addition/deletion of nodes or changes in the connectivity, thus producing realistic evolving preference data streams with characteristics that change over time (concept drift).

A major challenge is quantifying how realistic a preference model is when drifts are represented as changes in the underlying embeddings. We approach this by investigating the behavior of the underlying model on realistic data streams and studying how it can effectively replicate real-data distributions. In practice, in our evaluation, we adopt a scheme where *(i)* drifts can be identified in real data by observing the model's inability to capture realistic preferences, and *(ii)* the model automatically adjusts by making minimal changes to the graph structure to align actual and generated preferences. In our scenario, the optimal adjustments are those that can accurately predict actual preferences with minimal error, even in the presence of concept drift. Notably, the analysis of real data also allows us to devise and characterize the changes that typically can occur, thus devising a fully generative model for synthetic yet realistic preference data that can also be characterized by preference dynamics.

To summarize, the main contributions of the paper are:

- A preference model based on latent embeddings. According to such a model, concept drifts can be modeled as major modifications of the embeddings, that can be deemed as rewiring changes in the underlying tripartite graph. We show that this model is capable of characterizing real-data distributions and can hence be used as a basis for synthetic data generation.
- A stream-based data generator that can produce preference data also characterized by preference dynamics. We study a categorization of the possible drifts that can occur within preferences. To the best of our knowledge, this is the first attempt to systematically approach the problem and propose a solution.

The remaining sections of this paper are organized as follows. Section 2 presents a systematic literature review on synthetic data generators for recommender systems. In Section 3, we formulate the problem of generating synthetic data and propose an algorithm for detecting drifts in preference data. Section 4 provides details on experiments conducted using real data and discusses the application of learned models for emulating a data generation process. Section 5 introduces a general data generation scheme based on the insights obtained from the experiments. Finally, in Section 6, we draw general conclusions and outline future research directions.

2 Related Work

Generating synthetic data streams is not entirely novel and has been proposed in the past for general machine learning tasks, such as classification and regression [8, 38]. However, previous research on synthetic data streams for recommendation systems is limited and highly fragmented, tackling different data issues such as user privacy [37], scalability, shortage of benchmarks [6, 9, 31, 41], and, most closely related to this work, concept changes [21].

In work by Slokom et al. [37], the authors present a method to hide certain preference-related user information from a set of user-item interactions, which otherwise could be accessed. Their approach employs the CART

algorithm [10], and the output is a set of synthesized preference data, which preserves crucial statistical properties of the original data in order to be helpful for the testing of recommendation algorithms, but hides specific user information, such as their favorite actor in the case of movie recommendation. However, their method is unable to generate new ratings for unseen items, and it is unclear to what extent their approach can generate incremental data or produce data streams with dynamically changing characteristics.

The same limitation holds for the work by Belletti et al. [6]. The authors employ a data generator to produce a realistic copy of the MovieLens dataset but with a heavily extended number of users, items, and ratings. While this approach successfully scales up the MovieLens dataset by several orders of magnitude, therefore overcoming scalability, it does not address the issue of dynamic changes in user preferences. In other words, there are no parameters in the generation process that control drift in the data. In their study, Tso and Schmidt-Thieme [41] propose a synthetic data generation methodology to examine the performance of attribute-aware algorithms. They manipulate the level of informativeness of attributes in their artificial datasets to study their impact on algorithm behavior. In the domain of context-aware recommendation systems, del Carmen Rodríguez-Hernández et al. [9] developed a system specifically designed for generating preference data that incorporates contextual information. Their aim was to address the lack of suitable benchmarks in this area of research. Interestingly, their generator can mix real and synthetic data and, thus, mimic properties from existing datasets. However, a change in the data, e.g., user taste, is not considered in the generation process. The same holds for the work of Monti et al. [31], who present a cluster-based data generation method that learns the rating behavior of specific user communities and uses this information to generate new ratings.

In the work of Koren [24], the authors proposed to capture the temporal dynamics of user preferences. More specifically, they tried to model gradual and sudden drifts using linear and spline modeling of user biases. They also explored the impact of day effects on the bias. The results show that the splines are more flexible in capturing user effects on gradual drifts, and the single-day effects are useful for capturing sudden drifts. Similarly, the work of Bakir [4] describes the adoption of a matrix factorization model using SVD, and to map the temporal dynamics, the authors proposed the application of a conversion function used for the calculation of the age of the ratings, giving more weight to more recent ratings and slowly forget old preferences. The authors assumed that a gradual transition exists in the users' preferences. With the increase of interest in deep learning models (embedding), researchers developed some solutions for a sequential recommendation.

In particular, [28] used self-attention mechanisms to capture interactions between two watched items in predefined time intervals. Another approach proposed by Kang and McAuley [22] assigns weights to watched items to forget past interactions or give relevance to them. In both cases, the authors are trying to model the changes in user preferences. Our work not only models the users' preferences but also is capable of generating artificial drifts or producing synthetic data with similar data distributions.

A recent work proposed by Veloso et al. [43] models user preferences using incrementally updated embeddings. The proposed model is equipped with a drift detector to capture changes in users' preferences and initiate a real-time optimization task. The main idea is that every time during the learning process a drift is detected, the model needs to be adjusted by changing its hyperparameters.

Regarding drift detectors, there are several solutions available in the literature. Drift Detection Method (DDM) proposed by Gama et al. [14] observes the learner error rate and assumes that with the increase of observed examples, the error rate will decrease since the data distribution is stationary. If the error rate increases above a specific threshold calculated based on the error rate and standard deviation at one specific instant, it will trigger one alarm. Another variant of this method called the Early Drift Detection Method (EDDM), proposed by Baena-García et al. [3], was developed to improve detection performance under gradual concept drifts. The authors proposed adopting running average distance and running standard deviations instead of calculating the values at specific instants. The Hoeffding Drift Detection Method (HDDM) proposed by Frías-Blanco et al. [12] works with a different principle. It uses the Hoeffdings inequality and a moving average test to detect the

concept drifts. The Page-Hinkley method proposed by Sebastião and Fernandes [36] detects changes using the CUSUM control chart and computes the observed values and their mean until the current instant. If there is a change (increasing or decreasing the mean), it will signal an alarm based on a predefined threshold. Finally, the Adaptive Windowing method (ADWIN) proposed by Bifet and Gavalda [8] uses a dynamic-sized window that keeps recent examples with the same data distribution. The window can be split into two parts and compared if the data distribution is the same on both sub-windows, signaling an alarm if this condition is invalid. The focus of this work is not to develop new drift detection methods but to use them to identify the changes in the data distributions and build a portfolio of models.

Lex et al. [27] proposes the adoption of Base-Level Learning (BLL) equations from the cognitive architecture ACT-R together with recency-based modelling of music genre preferences, which indicates a tendency for individuals to listen to genres they have recently encountered. The significance of temporal drift in music genre preferences appears in all user groups. Consequently, incorporating recency information is expected to enhance prediction accuracy. Similarly, Oyewole [33] introduced a model for user temporal preferences designed to ascertain the recency weight of an item, thereby capturing shifts in user preferences over time. The core principle of this time-weighted algorithm is the determination of appropriate weights for all items, ensuring that more recently purchased items have a more significant influence on subsequent recommendations. Consequently, items purchased more recently are typically assigned higher weight values than those purchased earlier.

Han et al. [18] propose a hypergraph wavelet learning methodology to capture the subtle drifts in job preferences. To address the impact of noise in interactions, which is often a consequence of frequent preference changes (drifts), they introduce an adaptive wavelet filtering technique designed to eliminate these noisy interactions.

According to Sun et al. [39], the recommendation for each user is conceptualized as an individual learning task that aims to capture region-independent personal preferences by utilizing their check-in data without considering specific regions. Understanding users' immediate demands and modelling short-term preferences are crucial for the subsequent recommendation. To achieve this, they employ sequential models to represent users' personal preferences, thereby exploring their behaviour patterns through their complete check-in sequences using LSTM models.

Thaipisutikul [40] integrate contextual information into the subspace decomposition of short-term and long-term memory units within the Long Short-Term Memory (LSTM) framework. Specifically, they introduce a mechanism to attenuate the influence of the memory cell such that as time progresses or concept drift occurs, the impact of prior memory on the current output diminishes.

Yang et al. [48] employs a recurrent neural network (RNN)-based model to generate short-term user preferences, incorporating dual branches specifically designed for preference transfer and drift. The transfer branch and the Point of Interest (POI) embedding vectors are shared between tourists and residents within the current city. In contrast, the drift branch captures variations in preferences among different user roles, including a tourist in their home city, a tourist in the current city, and a resident in the current city.

Wangwatcharakul and Wongthanavasu [46] propose a model that learns multi transition low-dimensional factors to effectively capture variations in user interests across different periods. The model emphasizes the importance of temporal influences, noting that a well-designed algorithm should progressively diminish the impact of older data while maintaining accuracy in predicting future user preferences. Initially, the model calculates the rating shift rate using a time decay function that accounts for the forgotten information. Subsequently, it captures the latent drift in user factors between successive time points to monitor the multi transition factors.

To our knowledge, the work of Jakomin et al. [21] is the only one that addresses incremental changes in the data. In this study, the authors present a method to generate multiple inter-dependent data streams to evaluate recommendation or data fusion algorithms. The main idea behind their generator is to simulate scenarios where groups of similar users rate groups of similar items in the same manner. In other words, different probabilities connect the resulting users and item clusters. These probabilities determine the ratings that users of a given

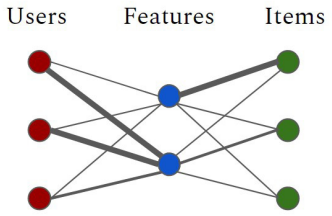


Fig. 1. User/Feature/Item graph

users cluster would give an item of a given items cluster. The output is a rating matrix where each non-zero cell represents the relation between a particular user and an item. Using arithmetic operations such as multiplication or division to the ratings at a given time allows them to simulate concept drift. However, this represents a limitation, as it does not consider changes in the latent feature space; *i.e.*, the embedding remains at a similar size, and users and items remain connected to the same features as the data evolves. Our work differs from the literature in the following aspects: *(i)* We model the user/item preferences and their evolution over time as a dynamic tripartite graph; *(ii)* We learn from a real data stream a sequence of latent matrices that models this graph, and, lastly, *(iii)* We use this sequence to produce synthetic data streams whose properties are similar to those of the original one.

3 Data stream modelling

The proposed data generator can produce synthetic streams of preference data based on a graph learned from real data. The generation process and the drift operations will be explained in the following sections.

3.1 Basic Framework

We start by introducing the notation to be used throughout the remainder of this paper. In the following, $u \in U = \{1, \ldots, N\}$ indexes a user, and $i \in I = \{1, \ldots, M\}$ indexes an item for which a user can express a preference. Let $r_{u,i} \in \{r_{\min}, \ldots, r_{\max}\}$ denote the preference (rating) of user u for item i. The range $\{r_{\min}, \ldots, r_{\max}\}$ represents a preference rank: When $r_{u,i} = r_{\min}$, user u maximally dislikes item i, while $r_{u,i} = r_{\max}$ denotes maximal preference for the item. Typical ranges are $\{0, 1\}$ (implicit preference) or $\{1, ..., 5\}$. In the following, we shall focus on implicit preferences.

The set of all preferences can be represented as a rating matrix R, or alternatively, as a set of triplets $R = \{(u_1, i_1, r_1), \ldots, (u_n, i_n, r_n)\}$, where each triplet (u_j, i_j, r_j) represents an interaction between any user u_j and any item i_j, with $r_j = r_{u_j,i_j}$ being the corresponding rating. When N and M are large, R only represents a partial and extremely small view of all possible ratings.

We adopt a matrix factorization framework for predicting the score that a user u assigns to each item i. Each user u and item i admit a representation in a K-dimensional space. We denote such representations by embedding vectors $\mathbf{p}_u, \mathbf{q}_i \in \mathbb{R}^K$, which represent the rows of the embedding matrices $\mathbf{P} \in \mathbb{R}^{N \times K}$ and $\mathbf{Q} \in \mathbb{R}^{M \times K}$.

Given user u and item i, the corresponding score $r_{u,i}$ can be modeled as a random variable with a fixed distribution whose parameters rely on the embeddings $\mathbf{p}_u$ and $\mathbf{q}_i$. This model can be represented with a weighted and undirected tripartite graph. It has three types of nodes: Nodes representing users (*user nodes*), those representing items (*item nodes*), and those representing features (*feature nodes*). Edges can only connect users to features or items to features. The weight of a user/feature edge (resp. item/feature edge) is a measure of how strongly that feature is present in the user (resp. item); see Figure 1.

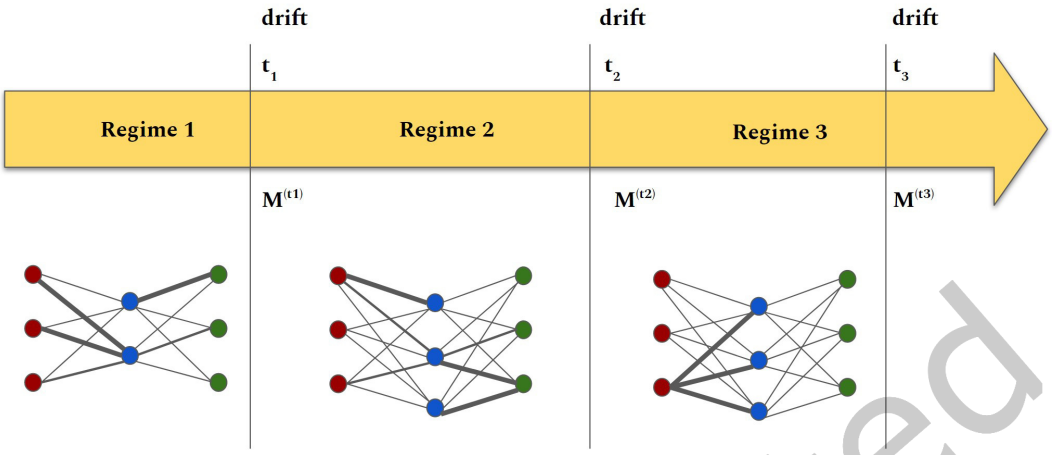


Fig. 2. Regimes in a data stream

The preference matrix R induces a natural ordering among items, where $i \succ_u j$ means that u prefers i to j, *i.e.*, $r_{u,i} > r_{u,j}$. The additional assumption that we make in this work is that the partial view R can be continuously updated. That is, either new unknown entries can be disclosed (for example, some users can express preferences for previously unseen items) or former preferences change (for example, due to change of tastes by the user, or due to a more accurate evaluation). Thus, we assume that the history of preferences produces a continuous stream of shards $R^{(1)}, R^{(2)}, \ldots, R^{(t)}$, where $R^{(t)}$ represents the view of R at time t.

As already mentioned, snapshots can overlap, *i.e.*, it can happen that both $r_{u,i}^{(n)}$ and $r_{u,i}^{(m)}$ exist for $m \neq n$, and $r_{u,i}^{(n)} \neq r_{u,i}^{(m)}$. We then define $\bigsqcup_{n \leq t} R^{(n)}$ as a merge among $R^{(1)}, R^{(2)}, \ldots, R^{(t)}$ that preserves recency. That is, by denoting $\bigsqcup_{n \leq t} R^{(n)}$ as $V^{(t)}$, we define $V_{u,i}^{(t)} = r_{u,i}^{(t^*)}$, where $t^* = \max\{t \mid (u, i, r_{u,i}) \in R^{(t)}\}$. If no such t^* exists, then $V_{u,i}^{(t)}$ is undefined.

Over time, the distribution of data can change, more or less abruptly. When this occurs, we are facing a *concept drift*. The situation is illustrated in Figure 2. Typically, we can consider a stream as the cooperation of multiple regimes that interact to produce observable data. A regime i is characterized by an underlying model, described by factorization matrices $\mathbf{P}^{(i)}$ and $\mathbf{Q}^{(i)}$. The appearance of a new regime as a result of a shift in data distribution reflects changes in user preferences or the entry of new features. As shown in the figure, the changes are essentially due to changes in the underlying graph which reflect on different samples in the data sampling process.

The challenge of managing new users or items can be addressed by adding new nodes initialized with a heuristic function that considers the characteristics of the graph in the i-th regime. However, this approach introduces an additional layer of complexity. A feasible simplification is to avoid imposing specific assumptions and instead initialize the nodes randomly, then evaluate the variation in model performance across different regimes. That is the actual implementation.

In practice, the problem we want to face is how to devise a data generation process based on factorization matrices $\mathbf{P}$ and $\mathbf{Q}$, capable of reflecting such changes. An initial attempt in this direction was made in [43]. More importantly, the questions we want to answer here are: What are the typical changes that characterize a real (or

realistic) data stream, and how can we model them by controllable (model-based) adaptations of the underlying sampling process?

We approach this problem in two different steps. In the first step, we study how a popular factorization model [34] can be used to track and describe changes in a real-life data stream. For this, we propose an inference algorithm that, given a data stream, is capable of tracking viable changes and adaptations of the underlying model to reflect the actual stream. The study and characterization of such changes will then allow us to devise a fully controllable procedure for generating data streams exhibiting concept drifts.

3.2 Learning Concept Drifts

The formulation we consider is inspired by the Bayesian Personalized Ranking (BPR) model introduced in [34]. The BPR model is a popular implementation for MF-based model, ensuring scalability, trainable on implicit feedback (no ratings are required). The training phase is quite effective, as it is performed by comparing pairs of items that a user has interacted with and optimizing the ranking of these items. This model allows us to process datasets with thousands of items and users. Being highly popular in literature, it represents the primary choice made by numerous authors in the field of Recommender Systems. Moreover, the BPR model is chosen because of its robustness and suitability for handling implicit preferences. However, the key aspect of the approach is not just the model itself but its ability to track changes over time through adjustments in the latent factor matrices. In principle, any MF model that supports a probabilistic interpretation of user/item adoption in terms of latent factors could be used.

The idea underlying this model is that a preference $i \succ_u j$ can be directly explained as closeness in a latent space where both items and users can be mapped. Mathematically, this can be devised by computing a factorization rank $\mathbf{p}_u^T \mathbf{q}_i$ for each pair (u, i), and modelling precedences by means of a Bernoulli process:

$$i \succ_u j \sim \text{Bernoulli}(p), \quad p = \sigma\left(\mathbf{p}_u^T(\mathbf{q}_i - \mathbf{q}_j)\right)$$

where $\sigma(a) = (1 + e^{-a})^{-1}$ is the logistic function. The optimal embeddings $\mathbf{M} = (\mathbf{P}, \mathbf{Q})$ can hence be obtained by optimizing the loss:

$$\ell^{bpr}(\mathbf{M}|V) = \sum_u \sum_{\substack{i,j \\ i \succ_u j}} \log \sigma\left(\mathbf{p}_u^T(\mathbf{q}_i - \mathbf{q}_j)\right) \tag{1}$$

where $i \succ_u j$ means $V_{u,i} > V_{u,j}$.

We can exploit the above learning scheme to devise a sequence of embeddings $\mathbf{M}^{(t_1)}, \ldots, \mathbf{M}^{(t_h)}, \ldots$ where each $\mathbf{M}^{(t)} = \left(\mathbf{P}^{(t)}, \mathbf{Q}^{(t)}\right)$ is trained on $V^{(t)}$ (i.e. $\bigsqcup_{n \le t} R^{(n)}$) as explained below. In particular, each model of this sequence represents a different *regime* and characterizes its data distribution.

Typically, recommender systems are optimized by exploiting historical interactions as training data, to produce a model that can predict users' future interactions in the testing data. This scheme inherently assumes that the training and testing data are drawn from the same distribution. However, distribution shifts can occur when changes in the preferences (expressed as modifications in the underlying graph discussed in the previous section) occur. In practice, a model trained on current data is likely to deteriorate its performance on future data when a substantial drift in the preferences occurs.

These observations inspire a specific protocol for training the embeddings $\mathbf{M}^{(t_1)}, \ldots, \mathbf{M}^{(t_h)}, \ldots$ characterizing the drift. The timestamp t_i here represents the drift: model $\mathbf{M}^{(t_{i-1})}$ underperforms on $R^{(t_i)}$ and, as a consequence, we need to retrain the model. The idea is to devise a continuous learning framework through the shards: at each step t the model is optimized on $V^{(t)}$. If the model substantially underperforms on $R^{(t)}$, then an adaptation is

required. Formally, we define the loss

$$\mathcal{L}^c(\mathbf{M}; R) = \sum_{u \in S} \ell^{bpr}(\mathbf{M}|R) + \lambda(||\mathbf{P}||_\infty + ||\mathbf{Q}||_\infty)$$

that includes the regularization term.

By minimizing the infinity norm of $\mathbf{P}$ and $\mathbf{Q}$, we enforce an upper bound on the matrix weights during the training process. This ensures that the weights belong to a limited range.

Then, given a current model $\mathbf{M}$, we check whether a drift has occurred by checking the loss $\mathcal{L}^c(\mathbf{M}; R^{(t)})$.

If no drift is detected, the process proceeds in two steps:

- First, a synthetic interaction dataset S is generated from $\mathbf{M}$
- Next, $\mathbf{M}$ is updated by optimizing the loss $\mathcal{L}^c(\mathbf{M}; S \sqcup R^{(t)})$.

The second step resembles a continual learning strategy [13]. In principle, the model should be updated from the entire $V^{(t)}$. However, in a streaming scenario, it is impractical to maintain the whole set of past interactions. Exploiting the current model to generate a sample S, with the same size of $R^{(t)}$, has the advantage of compacting the representation of past interactions and additionally, it provides better control on past preference shifts since the model will disregard obsolete interactions.

If a drift is detected, $\mathbf{M}$ is obsolete and requires a substantial update. In our framework, this means that the graph topology can change, with new users/items and/or features. We adopt a conservative approach to tackling this problem. For given user/feature probability vector $\mathbf{p}_u \in \mathbb{R}^K$, it is possible to devise an infinite-dimensional extension $\mathbf{p}'_u \in \mathbb{R}^\infty$ initialized such that $p'_{u,k} = p_{u,k}$ if $1 \leq k \leq K$, and $p'_{u,k} = 0$ for $k > K$, and the same property holds for item/feature vector $\mathbf{q}_i$. Thus, given $\mathbf{M}$, we denote by $\mathbf{M}'$ its infinite-dimensional counterpart. The learning objective on the new regime can hence be stated as computing a new version $\mathbf{M}'$ with minimal differences w.r.t. $\mathbf{M}$. Formally, we devise the loss

$$\mathcal{L}^d(\mathbf{M}'; \mathbf{M}, R^{(t)}) = \mathcal{L}^c(\mathbf{M}'; R^{(t)}) + \delta\Bigg(\sum_u \sum_{k=1}^{K} |p'_{u,k} - p_{u,k}| + \sum_i \sum_{k=1}^{K} |q'_{i,k} - q_{i,k}| + \sum_u \sum_{k>K} |p'_{u,k}| + \sum_i \sum_{k>K} |q'_{i,k}| \Bigg)$$

the second term in the loss represents the divergence between underlying feature vectors. The rationale here is that the updated model should penalize unnecessary discrepancies with the previous model on the current features, while at the same time letting the model introduce new features in order to adapt to the current changes. In practice, the training starts by adding Δ_K new columns (initially set to 0) to both $\mathbf{P}$ and $\mathbf{Q}$ and then optimizing the extended model through $\mathcal{L}^d$ on the new shard $R^{(t)}$. Algorithm 1 provides an overview of the learning procedure. In this procedure, we utilize the driftDetection function to detect any drifts in the losses. The effectiveness of drift detection heavily relies on the chosen signal, which in our case is primarily the loss values. By monitoring the loss function's value, we can identify significant deviations in the model's performance, indicating the presence of drifts. However, it is worth noting that the choice of the signal for drift detection can vary depending on the specific application and problem domain, which can lead to different performance for detection algorithms. We employ the $HDDM_W$ algorithm [12], a well-known drift detection method based on Hoeffding's bounds with a moving weighted average test. $HDDM_W$ utilizes the Exponentially Weighted Moving Average (EWMA) statistic as an estimator. Given a stream of values (in our case, loss values) as input, it determines the estimated status of the stream: *Stable*, *Warning*, or *Drift*. [1]

Therefore, the model is updated using two training modalities:

[1]Notice that the adoption of $HDDM_W$ requires bounded signals, which can be guaranteed by bounding the difference in Eq. 1.

- when a drift occurs, we train the model $\mathbf{M}'$ with the data of the current shard and penalizing the distance between $\mathbf{M}'$ and the model learned in the previous regime $\mathbf{M}$;
- otherwise, the current model $\mathbf{M}$ is refined on data of the current regime and a sample of historical data generated from the model $\mathbf{M}$ of the current regime. This guarantees that the model is aware of both a summary of the previous data and the new data available.

Finally, the generateSamples function leverages the matrix $\mathbf{M}$ to generate n_S new samples. This procedure involves sampling items for each user based on their score, with the probability of selection proportional to the score. The item frequency for each user (or user frequency for each item) can be estimated using a Zipf distribution or provided as input to preserve the original properties. These frequencies help determine the number of items to sample and bias the probability of selecting low-popularity items.

The sample is combined with the shard's data to train the current model. This approach aims to retain historical information without requiring the storage of all data in the current regime, in this way we balance the weight of new preferences and the past ones. However, when a drift occurs, we switch to a different strategy aimed at minimizing the deviation from the previous model; thereby reducing the impact of the limited current data, i.e. the preferences of the current shard, on updating the new model and preserving a soft transition between regimes. The value of parameter n_S has been set equal to the size of the B^{train} set of the current shard, weighting equal to the number of preferences observed in the current shard and in the current regime.

4 Experiments

We conducted an empirical evaluation of Algorithm 1 with two main objectives. Firstly, we aimed to monitor actual drift in realistic scenarios. Secondly, we aimed to devise a data generator that captures the properties of realistic data streams and effectively models concept drift similar to what is observed in real data.

We processed three real data streams using our proposed approach. During the training phase, we identified a sequence of regimes for each dataset, each characterized by its data distribution. We associated a matrix factorization (MF) model with each regime. Using this sequence of models, we generated a synthetic data stream that closely resembled the original stream. This was achieved by imposing the same regimes in the same order, with an equal number of samples for each regime. By comparing the original and synthetic streams, we assessed the fidelity of the generator in sampling new data.

Our approach yielded satisfactory results when the number of item preferences was sufficiently high to train a model capable of replicating a real stream. This was especially evident in datasets where there were numerous items with few preferences and a core set of highly popular items. The behavior can be explained by the model's inability to accurately estimate the distribution of underrepresented items. To achieve good generation performance, it was crucial to have a sufficient number of preferences for each item. This condition was naturally met when the number of users significantly exceeded the number of items. In fact, the first and third datasets satisfied this condition, as the distribution of the synthetic data closely resembled that of the original data.

We also chose a second dataset that did not exhibit this behavioral pattern, where the number of users was much smaller than the number of items. In this case, the distribution of the synthetic data generated by the generator deviated significantly from the original data. To further support our hypothesis, we reversed the roles of users and items and repeated the experiment, demonstrating the importance of having an adequate number of preferences to derive an effective model. The results were satisfactory in this case.

Additionally, we selected a third dataset to analyze changes in distributions during a drift. We drew insights from a study by Yang et al. [49], where they tackled the challenge of developing a recommender system that can adapt to the dynamic nature of user preferences and item popularity.
To foster reproducibility, we have publicly released all the data and code required to reproduce our experiments.[2]

[2]https://github.com/fsp22/mcd_dds4rs

Algorithm 1: Learning Latent Matrices

Input: Stream $\mathbb{R} = R^{(1)}, \ldots, R^{(n)}, \ldots$ of preferences ordered in shards;
number of samples n_S to generate for training the model within the i-th shard, in conjunction with the shard's data.
Output: A list $\mathbf{M}^{(0)}, \ldots, \mathbf{M}^{(h)}, \ldots$ of preference models

$t = 0$; $\ell = \emptyset$
$q = \emptyset$; $r = \emptyset$
initialize $\mathbf{M}^{(t)}$
for *each* $R^{(i)}$ *in* $\mathbb{R}$ **do**
  $B^{train}, B^{test} = \texttt{splitData}\left(R^{(i)}\right)$
  $\ell^{drift} = \mathcal{L}^c\left(\mathbf{M}^{(t)}; B^{test}\right)$
  $drift = \texttt{driftDetection}(\ell^{drift}, \ell)$
  if *drift* **then**
    $\mathbf{M}^{(t+1)} := \texttt{extendModel}(\mathbf{M}^{(t)}, \Delta_K)$
    optimize $\mathbf{M}^{(t+1)}$ on $\mathcal{L}^d(\mathbf{M}^{(t+1)}; \mathbf{M}^{(t)}, B^{train})$
    t+=1
  end
  else
    $S, q, r = \texttt{generateSamples}(\mathbf{M}^{(t)}, n_S, q, r)$
    optimize $\mathbf{M}^{(t)}$ on $\mathcal{L}^c\left(\mathbf{M}^{(t)}; S \sqcup B^{train}\right)$
  end
  $\ell^{eval} = \mathcal{L}^c(\mathbf{M}^{(t)}; B^{test})$
  $\ell = \ell \cup \{\ell^{eval}\}$
  Update user and item popularities q, r based on $R^{(i)}$
end
return $\bigcup_t \{\mathbf{M}^{(t)}\}$
Function `generateSamples`($\mathbf{M}$, n_S, $q = \emptyset$, $r = \emptyset$, $\alpha = 1.8$, $\beta = 1.9$, $c = 1$, $d = 5$):
  Function `normalize`(μ):
    $\tilde{p} = \sigma(c \cdot \mu - d)$
    return $\tilde{p}/(\sum_i \tilde{p}_i)$
  if $q = \emptyset$ **then**
    for *each* $u \in U$ **do** $q_u = Zipf(\alpha, 1, M/c)$
  end
  if $r = \emptyset$ **then**
    for *each* $i \in I$ **do** $r_i = Zipf(\beta, 1, N/d)$
  end
  for *each* $u \in U$ **do**
    $q_u = \lceil n_S q_u / (\sum_u q_u) \rceil$
    $\tilde{p}_u = \texttt{normalize}(\mathbf{p}_u \cdot \mathbf{Q})$
    $p_u = (\tilde{p}_u \odot r)/(\sum_i \tilde{p}_{u,i} r_i)$
    $n_u = \min(q_u, |\{i | p_{u,i} > 0\}|)$
    Sample with replacement n_u items I_u from I with probability proportional to p_u
    $S = \texttt{generateTriplets}(u, I_u)$
  end
  return S, q, r

4.1 Datasets

Our analysis is conducted on the following popular benchmark datasets, each from a different domain and having specific characteristics:

- **Microsoft News (MIND)**: This dataset consists of news articles published on the Microsoft News website and the feedback logs generated by 1 million users over a span of 6 days. We utilize the click information of each news article by users to construct a dataset. The read news articles serve as user preferences, and and we consider only users who have at least one preference (news read). The final dataset comprises 19,206 items and 3,958,500 preferences from 750,434 users. The dataset can be accessed at https://msnews.github.io/.

- **Amazon Video Games**: This dataset contains ratings provided by users for video games. The ratings range from 1 to 5. As our approach focuses on implicit feedback, we binarize the data by assigning a value of 1 to user-item pairs where the rating is strictly greater than 3, and 0 otherwise. To address GPU memory limitations, we process a subset of the dataset, considering the last 824,752 preferences from 26,038 users registered over a span of 10 years, with respect to 567,865 items. The dataset can be accessed at http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/ratings_Video_Games.csv.
- **Yoochoose**: This dataset was published for the RecSys Challenge 2015. It comprises a collection of sessions from a retailer, specifically focusing on the purchase events of users. To streamline the dataset, we selected items that were interacted with more than 4 times and removed users with fewer than 3 purchased items. After the cleaning process, the dataset consists of 143,481 users, 9,975 items, and 642,237 interactions. The dataset can be accessed at https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015.

These datasets were chosen due to their popularity and diverse characteristics, enabling us to conduct comprehensive analyses and draw meaningful conclusions.

4.2 Modelling Data Streams

Following the protocol presented in Section 3, we test our model on the datasets reported in Section 4.1. For each user u and each item i which is positive for u (i.e. $(u, i, 1) \in R^{(t)}$), four negative items j are sampled (i.e. $(u, j, 1) \notin R^{(t)}$). In our framework, we consider as *negative* any item that is not explicitly *positive* for a user. This includes both items that the user has never interacted with and items that the user explicitly dislikes. Thus, in this case, $(u, j, 1)$ must not belong to $R^{(t)}$ which is different from saying that u dislikes j, i.e., $(u, j, 0)$ must belong to $R^{(t)}$.

This approach is common in recommender systems, particularly when dealing with large item catalogs. The rationale behind this method lies in the typical characteristics of recommendation scenarios. In most cases, the number of items a user likes is significantly smaller than the total number of available items. For instance, in a catalog of millions of products, a user might only be interested in a few hundred. This creates a very sparse preference matrix. Given this sparsity, randomly sampling an item has a high probability of selecting one that the user would not like or has not interacted with, making it a reasonable proxy for negative feedback. This assumption simplifies the modeling process and is especially useful when detailed interaction data (like viewed but not purchased items) is not available.

The triplets (u, i, j) are processed in chronological order grouping them in batches of 2 000 tuples. We split each batch B in two sets: i) a training set B^{train} (70%), and ii) a test set B^{test} (30%).

To train the model, we use 20 epochs; the optimizer is the ADAM algorithm with a learning rate of 0.001, and the loss functions presented in Section 3.2. After each batch, we evaluate the model on the test set, and we report some performance metrics. We consider $\ell^{bpr}(\mathbf{M}|B^{test})$ and the Hit-Rate score. Let us recall the definition of this last metric: Given a user u occurring in B^{test}, we consider the set P_u of its positive items (*i.e.*, $P_u = \{i \mid (u, i, j) \in B^{test}\}$) and sample a set N_u of negative items ($|N_u| = 100$, in our case). Then for a given cutoff k (we use $k = 10$ and $k = 5$) we consider the $k - 1$ negative items in N_u for which $\mathbf{M}$ returns the highest score. Among them, we consider the item j with the lowest score. There is a hit with cutoff k, for the user u and the item $i \in P_u$, if $\mathbf{M}(u, i) \geq \mathbf{M}(u, j)$. In the following, we report both HR@5 and HR@10.

4.3 Experiment on MIND

Figure 3 shows the loss computed on the test part of the current batch (*i.e.*, $\mathcal{L}^{bpr}(\mathbf{M}, B^{test})$) averaged on a moving window of 20 batches. The moving average highlights the increasing/decreasing trend, disregarding a local minimum/maximum. The orange vertical lines indicate the batches where the drift algorithm has detected a warning before a potential drift; meanwhile, the red lines indicate a drift.

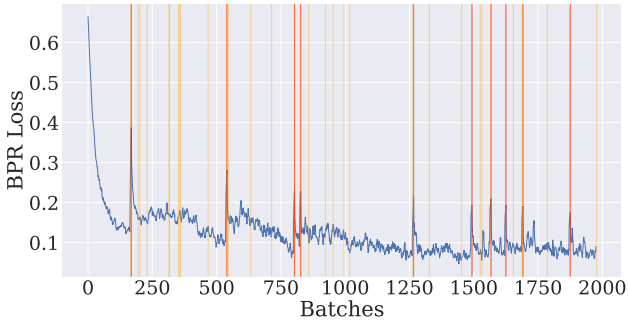
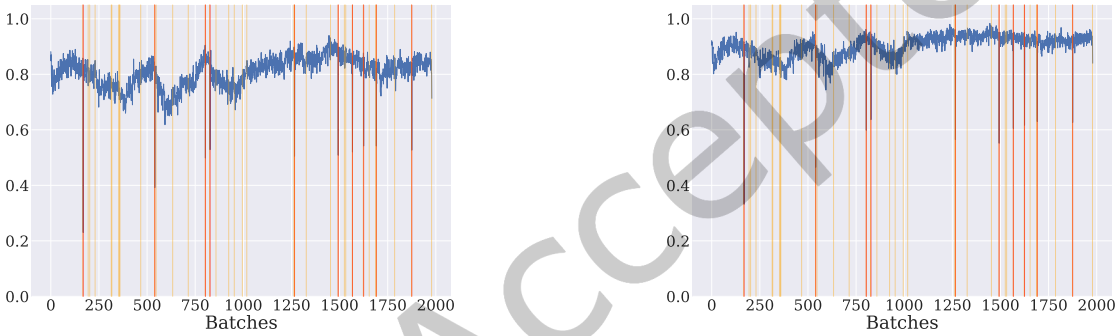


Fig. 3. Loss on test set part of each batch of the Microsoft News dataset.



(a) HitRate@5

(b) HitRate@10

Fig. 4. Results for the Microsoft News dataset.

The loss is relatively stable with minimal increase/decrease due to the nature of the dataset. The number of items is less than the number of users, while the news are equally scattered across the dataset that covers only six days. For this reason, each batch contains a few *new* items, and the preferences on hot news registered in a few hours should motivate the high number of warning and drift alerts and the value differences between consecutive batches. We highlight that each batch contains the news read in a few minutes.

The good performance in the loss chart can be also seen in the HitRate charts. Just after Batch 500, in the Figure 4a, we can see a low HR value (HR@5 $\approx$ 0.6). This behaviour could be explained by a drift following multiple warnings in the region close to the minimum. Probably, this indicates a significant change. Comparing the HR@5 values with the HR@10 values, the second ones are better for the same batches, indicating that the model worsens the predictions only on some items while maintaining good overall performance. The number of warnings is due to the fluctuations recorded in the loss.

The visual representation in Figure 5 provides insights into the distinctions between segments of the real data corresponding to different regimes and their synthetic counterparts. The synthetic data in each regime is generated by sampling from the model trained on the corresponding real data. To ensure a fair and meaningful comparison between the synthetic and original data streams, we meticulously structured the synthetic data to

mirror the original data. This involved maintaining the same regimes in the same order and ensuring an equal number of samples for each regime.

In the charts, the horizontal axis represents the i-th frequency denoted as k, while the vertical axis indicates the cumulative count of items (i.e., $P(f > k)$), normalized by the sum of values. Upon analyzing the distributions, we observe that the frequencies exhibit comparable patterns, thereby confirming the effectiveness of training the different models. The similarity is also confirmed by a positive outcome of the Kolmogorov-Smirnov test [20] on each regime.

The observed similarity between the distributions in the charts serves as strong validation for the effectiveness and fidelity of our approach in generating synthetic data that accurately reproduces the characteristics of the original stream.

It's crucial to note that our comparison focuses on frequency distributions rather than preference distributions in the charts. This deliberate choice is motivated by the recognition that comparing preference distributions can be problematic due to the influence of popularity bias on predictive models for preference data [23]. Popularity bias tends to overlook contributions from cold start users and low-popularity items, especially in preference data with vast item catalogs. Comparing distributions based on preferences would consequently fail, due to inaccurate associations involving cold-start users and/or low-popularity items. By contrast, it is important to highlight that such inaccuracies do not significantly impact the global modeling capabilities of the learned embedding matrices, which is the main focus here.

4.4 Experiment on Amazon Video Games

We conducted similar experiments for the other two datasets. Firstly, for the Amazon Video Games dataset, we analyzed the loss values (Figure 6) and HitRate (Figure 7) on the test set. In these charts, we observe the presence of 7 regimes, indicating 6 detected drifts by the algorithm. We observe similar patterns as seen in the MIND dataset, where the performance deteriorates when multiple drifts occur in close proximity.

To compare the item distribution of the synthetic stream with the real stream, we examined the distributions shown in Figure 8. In contrast to the MIND dataset, the distributions in this case are significantly worse. This indicates that the number of preferences for each item is insufficient to acquire the necessary knowledge required to accurately replicate the stream.

To further support this observation, we inverted the roles of users and items in the dataset. By doing so, the number of items became smaller than the number of users, resembling the MIND dataset. Analyzing the loss performance (Figure 9) computed on each batch (i.e., $\mathbf{M}, B^{test}$) after the training phase, we notice a rapid increase in loss value when a drift appears in the stream. Subsequently, the loss value decreases as the model adjusts to the new data. Although the drift detection algorithm triggers a false warning, it has minimal impact on the loss value. We do not observe any significant changes in the HitRate values.

We computed HR@5 and HR@10 (Figure 10) and observed similar behavior in both settings. The model performs poorly in the vicinity of a drift due to significant changes in the data distribution, even with new user/item preferences or previously unseen items. However, after the drift, the new model is trained on the updated data, and within a few batches, it regains optimal performance.

In contrast to the previous case, the distributions of the generated data are noticeably improved in this scenario. As depicted in Figure 8, the curves representing the distributions are very close to each other. This observation aligns with our intuition regarding the significance of having an adequate number of item preferences to train a generative model effectively. The close similarity between the curves confirms the importance of having a sufficient number of preferences for each item in order to accurately replicate the data.

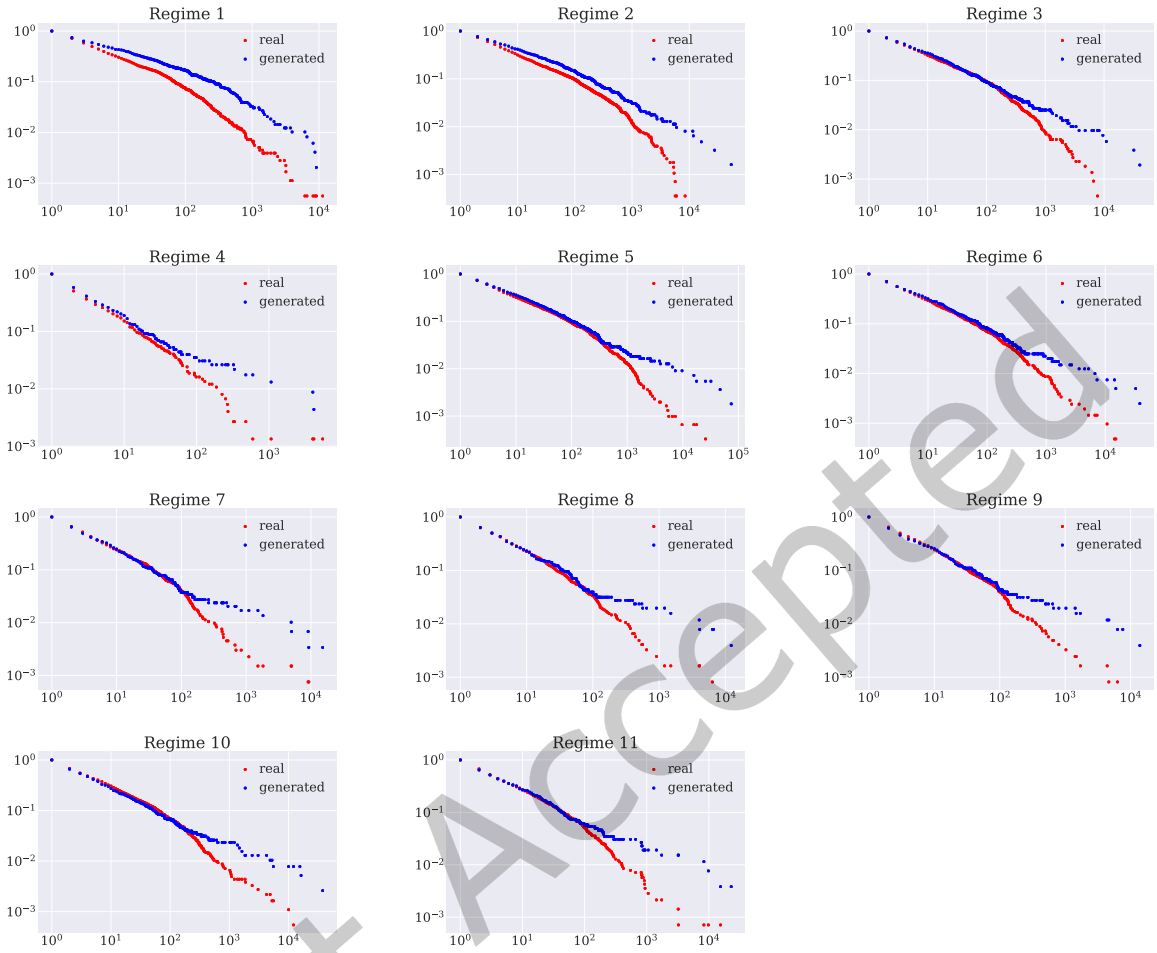


Fig. 5. Comparison of real and synthetic stream for the Microsoft News dataset: The chart shows the $P(f > k)$ where the i-th frequency is denoted as k (log-scale on both axes).

4.5 Experiment on Yoochoose

The results from the Yoochoose dataset exhibit a behavior similar to the previous dataset. Figure 11 illustrates the loss on the test set, highlighting a single drift where there is a noticeable increase in the loss value. However, the changes in the data distribution have only partially affected the quality measures, as depicted in the HitRate charts in Figure 12.

This observation suggests that while the model's performance is impacted by the drift, it is able to adapt and adjust to the new data to some extent. The changes in the data distribution have resulted in a decrease in the HitRate, indicating that the model's recommendations are less accurate during the drift period. However, it is worth noting that the model gradually recovers and improves its performance after the drift.

Overall, the results from the Yoochoose dataset confirm the presence of drift and the model's ability to adapt, albeit with some impact on the quality of recommendations during the transition period.

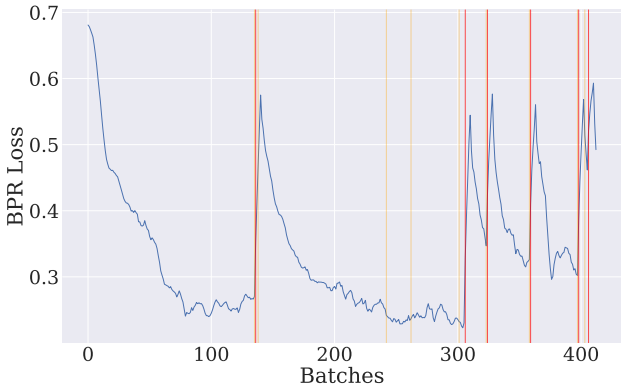
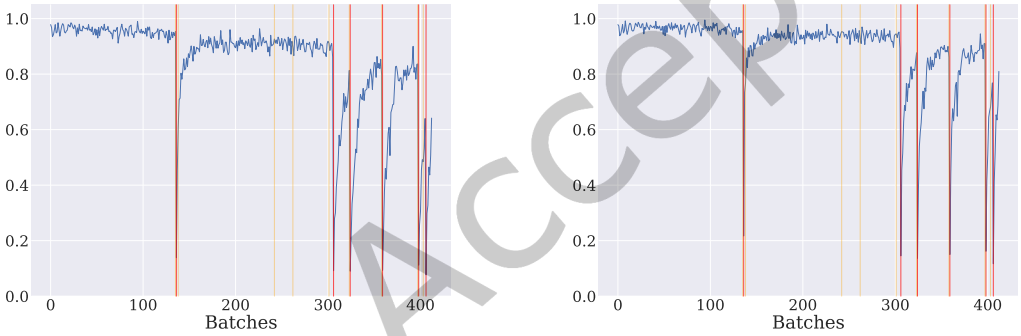


Fig. 6. Loss on test set part of each batch of the Amazon Video Games dataset.



(a) HitRate@5 (b) HitRate@10

Fig. 7. Results for the Amazon Video Games dataset.

4.6 Comparison with a Baseline Algorithm

In this section, a comparison was made between the BPR algorithm and a model that could be more easily adaptable to managing new users/items. We have chosen a variational autoencoder, in which the number of items the system can handle is fixed a priori. The model receives as input the items that a user has already seen and, based on this information, assigns a score to the entire catalog to generate a ranking and thus build a recommendation list. Having the whole catalog available from the outset gives the model a competitive advantage over the BPR model, which instead adapts by updating the latent space when significant changes are detected in the system, both in terms of new users/items and/or new preference patterns for existing users/items. Naturally, the impact on performance will be more noticeable with many items distributed over a broad temporal range. Conversely, in the presence of lower variability, i.e., when the model can learn patterns in the initial moments that are valid for the entire observation period, a model like the VAE will have a competitive advantage over an adaptive model.

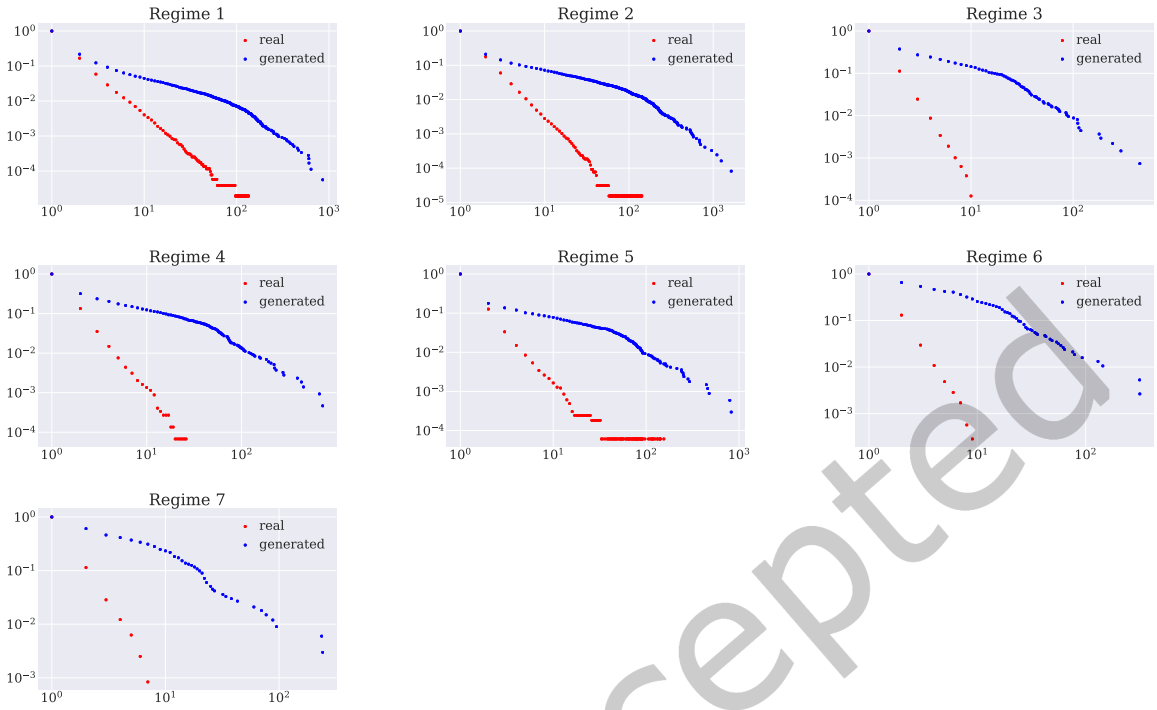


Fig. 8. Comparison of real and synthetic stream for the Amazon Video Games dataset: The chart shows the $P(f > k)$ where the i-th frequency is denoted as k (log-scale on both axes).

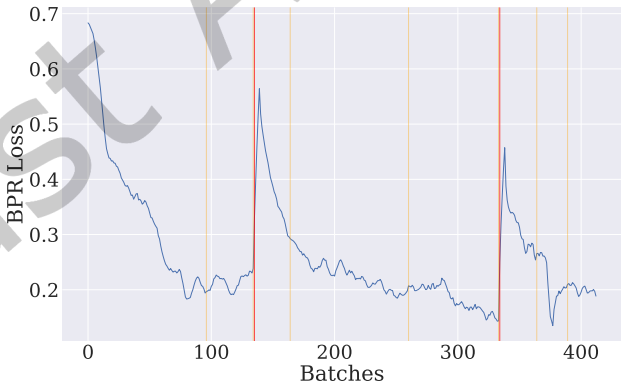
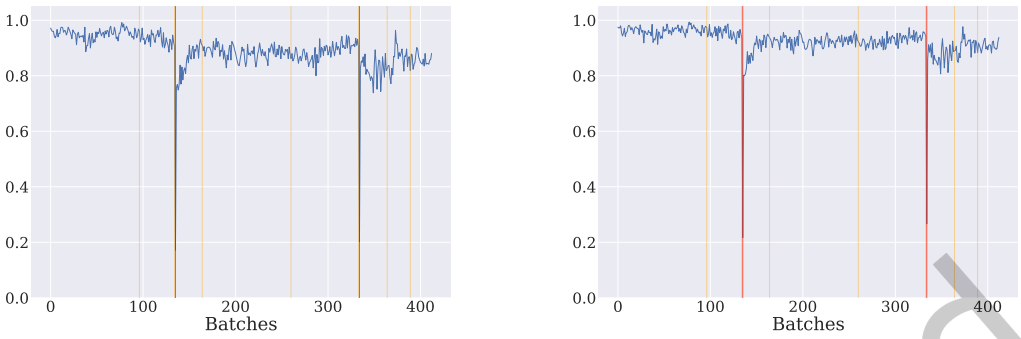


Fig. 9. Loss on test set part of each batch of the Inverted Amazon Video Games dataset.



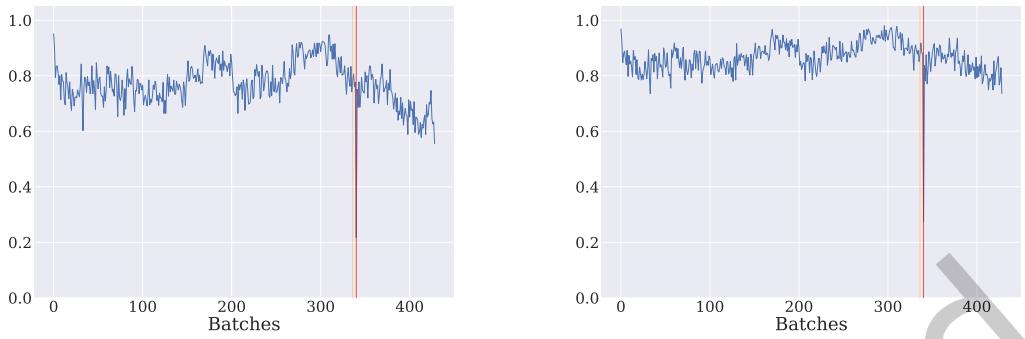
(a) HitRate@5 (b) HitRate@10

Fig. 10. Results for the Inverted Amazon Video Games dataset.

Fig. 11. Loss on test set part of each batch of the Yoochoose dataset.

Performance was measured for three datasets: MIND, Amazon Video Games with reversed roles, and Yoochoose. The experiment could not be executed for the original Amazon Video Games dataset because the model required excessive memory due to the high number of items (567,865). Considering the Hitrate@5 values (the Hitrate@10 metric shows similar trends and is not reported for brevity), for the MIND dataset (Figure 15), we can observe that the VAE model reaches a peak value in the middle of the stream and maintains constant performance for the rest of the observation period. Conversely, the BPR model exhibits more fluctuating performance following the detection of drifts. The nature of the data can explain this: the dataset consists of news produced over 6 days. In many cases, these are news items covering the same event, resulting in reduced variability in identified user preference patterns. The system can learn highly generalizable recommendation models. For instance, it is common to observe many users sharing similar patterns because they read multiple news items on the same topic. In this case, the VAE is more effective in detecting latent space similarities than the BPR.

For the Amazon Video Games dataset with reversed roles (Figure 16), i.e., where users and items were swapped to have a catalog of 25k items, we observed worse performance for the VAE. As the number of new items grows,



(a) HitRate@5 (b) HitRate@10

Fig. 12. Results for the Yoochoose dataset.

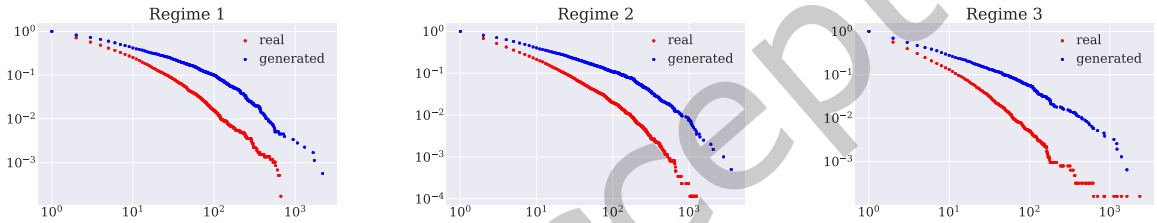


Fig. 13. Comparison of real and synthetic stream for the Inverted Amazon Video Games dataset: The chart shows the $P(f > k)$ where the i-th frequency is denoted as k (log-scale on both axes).

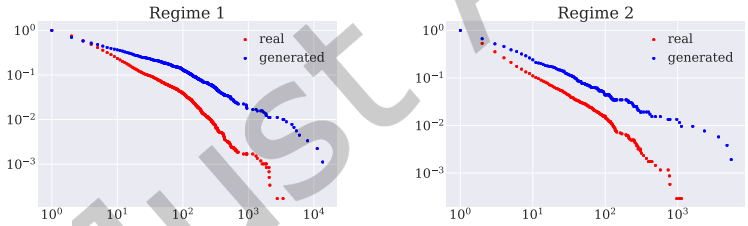


Fig. 14. Comparison of real and synthetic stream for the Yoochoose dataset: The chart shows the $P(f > k)$ where the i-th frequency is denoted as k (log-scale on both axes).

the model's performance degrades. Additionally, the recommendation list contains items for which no preferences are observed. An adaptive system like BPR manages these changes better, ensuring superior performance.

We can see a similar trend for the Yoochoose dataset in Figure 17. The main difference lies in the oscillating pattern seen in the charts. The system adapts more quickly to changes in preferences, but the limited temporal range prevents the model from converging to mid-high levels. In this context, BPR adapts faster to changes, while the VAE system tends to be more stable around a lower median value than BPR. Moreover, a descending trend is

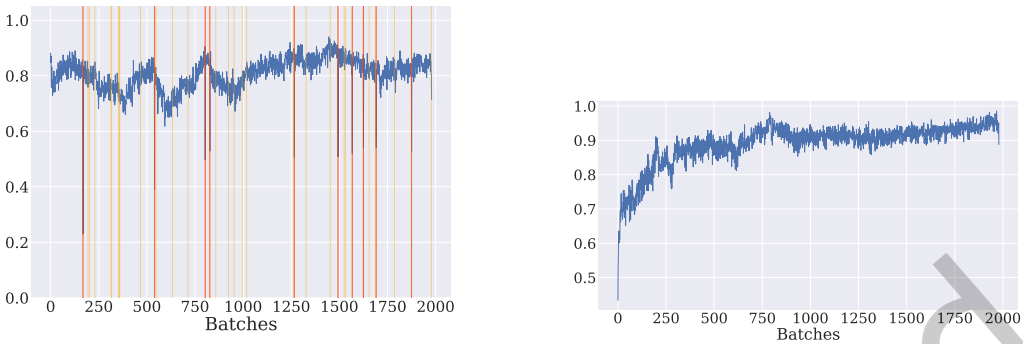


Fig. 15. Comparing BPR and VAE on the MIND dataset (HitRate@5).

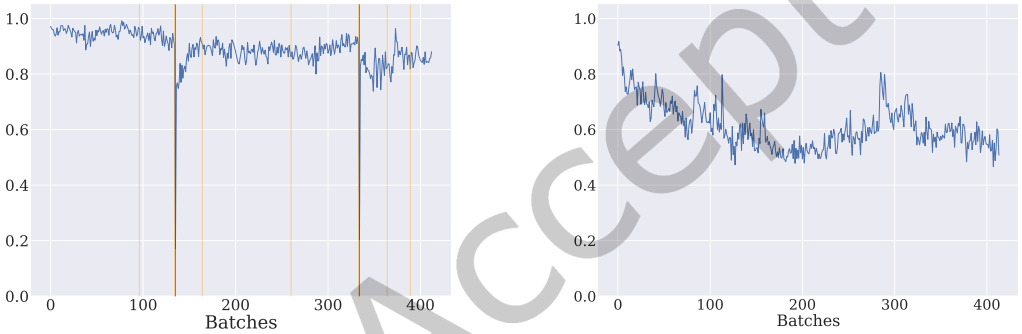


Fig. 16. Comparing BPR and VAE on the Inverted Amazon Video Games dataset (HitRate@5).

evident in the last shards of the stream: a relatively high number of new items appears with a limited number of preferences, and both systems struggle to identify new patterns effectively.

4.7 Ablation study

Our approach aims to identify a method that allows generating realistic data, replicating the trends present in a real stream. To capture possible changes in preferences, regimes homogeneous in expressed preferences are identified, and their corresponding models are stored to generate data derived from those behavioral models. Two analyses were conducted to evaluate this protocol:

- try ignoring the regime identification, so we train the model only on the first regime and then test the performance on data of the next regimes;
- compare the performance ignoring the stream nature of the data. We sample a 'small' training set with data of all regimes, then test on the full dataset w.r.t regimes.

In the first experiment, we trained a single model only on data belonging to the first regime. As soon as a drift is detected, the model is frozen and tested on the rest of the stream. It is evident that the model achieves good performance only on the data belonging to the regime it was trained on and performs poorly on the remaining

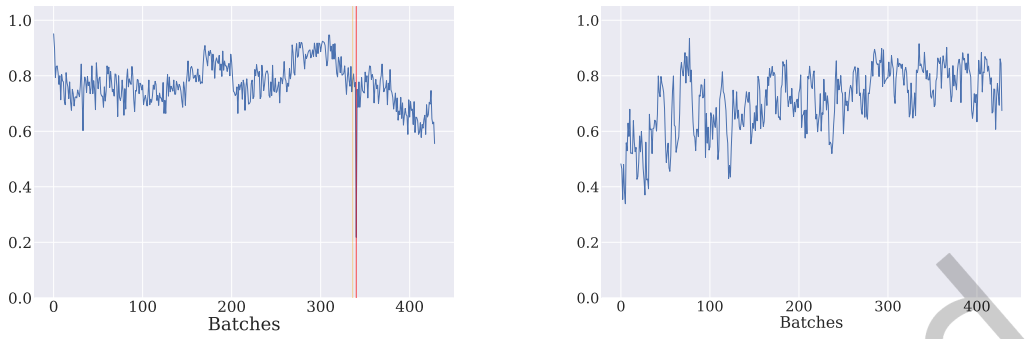
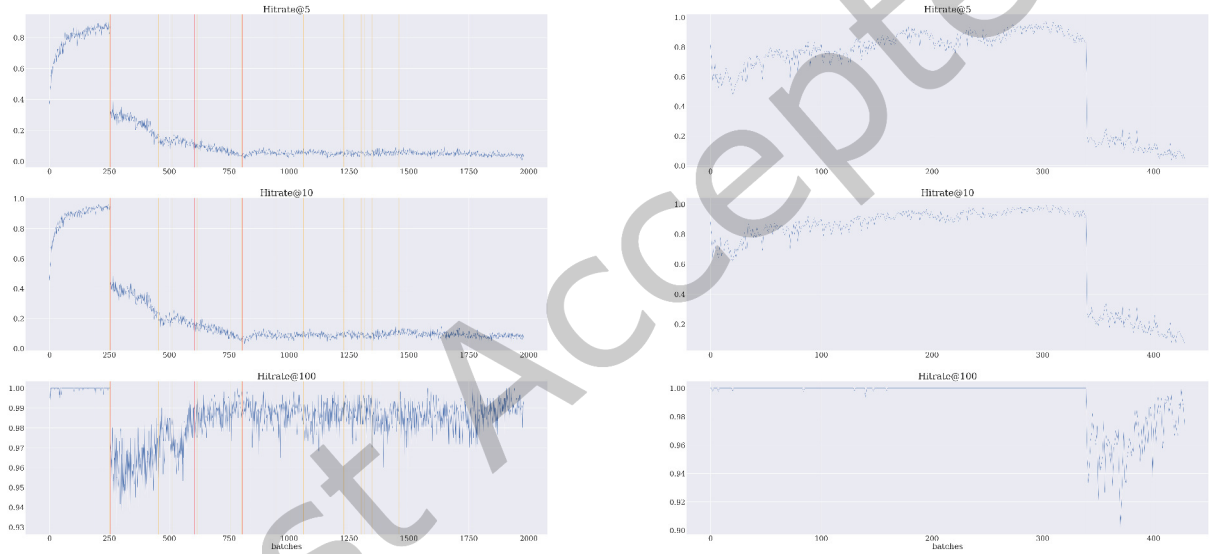


Fig. 17. Comparing BPR and VAE on the Yoochoose dataset (HitRate@5).



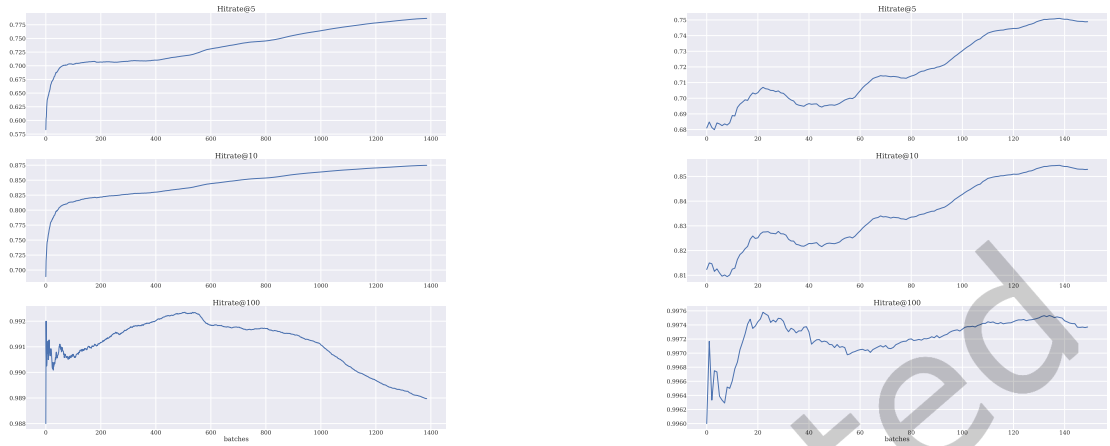
(a) MIND dataset

(b) Yoochoose dataset

Fig. 18. Test performance of the first model, i.e. trained only on first regime, on the entire stream.

regimes (Fig. 18). This result is not surprising but confirms the need to implement a mechanism of continuous training that creates and trains a new model when the characteristics of the stream data change, and their properties can no longer be captured by the previous one.

In the second experiment, we created a training set by randomly extracting 30% of the samples from the original dataset. We trained a single model using this training set and verified that its performance is similar to that achieved by the system we are proposing (Fig. 19), which creates and trains a new model when a new drift occurs. However, this simple model is not realistic and cannot be applied in real-world contexts because it has the advantage of immediately analyzing data samples from the entire history of the stream. It is not realistic because no model can '*look*' into the future. The objective is only to test whether the detected changes are present in the



(a) Mind dataset (b) Yoochoose dataset

Fig. 19. Test performance on the dataset with a model trained on a sample of data.

original dataset or are only a side effect of the model training. Breaking the chronological order of preferences could heavily impact dataset evaluation: For Yoochoose and Amazon Video Games, it means changing the order when items '*appear*' to the system. For the MIND dataset, the side effect is breaking the correlation between news: a sequence of news on the same event loses meaning, influencing the performance evaluation. However, this behavior can be accepted considering the scope of this experiment. The architecture we propose, however, remains competitive with this '*ideal*' model and matches its performance.

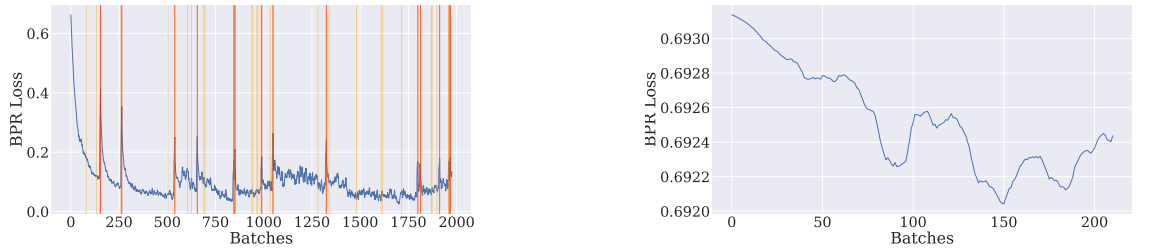
In a further set of experiments, we evaluate whether the approach effectively supports actual shifts in data distribution. In our approach, we only increase dimensionality when a drift is detected. This means that if no increase in loss is detected, we do not modify the structure of the current model. Conversely, any increase in the loss is identified as a structural inability of the model to properly capture user preferences, prompting us to train a new model with updated dimensionality.

Figure 20b illustrates the test loss trend for the Yoochoose dataset after shuffling the data. In this case, the dataset no longer demonstrates features that degrade model performance. Essentially, the model fails to adapt to new items or preferences. A similar trend is observed with the Amazon dataset.

Figure 20a, where the focus is on the MIND dataset, presents a contrasting behavior: the model's performance worsens upon encountering new items. In this dataset, items typically follow an inherent periodicity due to evolving topics over time. Users tend to read news on the same topic within a short timeframe before moving on to the next one. Shuffling exacerbates this scenario, leading the framework to adjust by identifying a greater number of regimes.

5 Enabling full generation

The rationale of the above analysis is that, by tracking a realistic data stream, it is possible to devise a sequence of models that can be used to generate a synthetic data stream similar to an original one. The comparison between the original stream and the synthetic stream in the previous section allowed us to evaluate the fidelity level of the



(a) Mind dataset (b) Yoochoose dataset

Fig. 20. Test performance on the dataset with a model trained on a sample of data.

Fig. 21. Summary statistics on changes in the regimes detected. The first column refers to the Microsoft News dataset. The other two columns refer to (original and inverted) Amazon Video Games. The last column refers to the Yoochoose dataset.

generator in sampling new data. Clearly, this framework allows to generate streams with an arbitrary number of regimes, in an arbitrary order, and with an arbitrary number of samples. However, this generation can only rely on the models extracted from the original data stream. This is clearly a limitation since we would like to generate new regimes that can also deviate from the extracted models. Better, we would like to devise a general scheme where we can synthesize the matrices associated with each regime, and even enable changes of these matrices along the shards.

In Figure 21, we present statistical insights into the changes observed in the data from the previous experiments. The first row provides information about the appearance/disappearance of users and items across different regimes. It illustrates the dynamic nature of the dataset, where users and items can appear or vanish over time.

The second row focuses on the feature vectors associated with users and items and how they evolve across the regimes. While a significant number of components remain stable, indicating consistent user and item characteristics, certain users undergo substantial changes in their feature vectors. This suggests shifts in their preferences or characteristics over time.

Additionally, Figures 5, 8, 13, and 14 display other noteworthy changes in the distributions. These figures provide visual representations of the variations observed in the generated data, reflecting shifts in user preferences,

trends, or other significant modifications. The number of users is reported in log-scale on the X-axis, while the number of items is reported in log-scale on the Y-axis.

In general, the following drift directions can be identified:

- Users can undergo changes in their tastes, leading to shifts in their preferences. This can be visualized as a rewiring of certain user nodes in the feature graph discussed in Section 3.1.
- New trends may emerge, influencing user preferences. This is directly linked to the structure of the feature graph and how nodes (representing users and items) are connected within it.
- New users can join the system, expressing their preferences and impacting the underlying preference distributions. Similarly, new items can be introduced, further altering the distribution of available items.
- Conversely, some users may leave the system or become inactive, and certain items may become obsolete. These changes result in modifications to the underlying item distribution.

By understanding and addressing these drift directions, we can effectively capture the dynamics of user preferences, trends, and the evolution of the item space.

In Algorithm 1, we observed that the generation of new samples (using the generateSamples procedure) depends on the $\mathbf{P}$ and $\mathbf{Q}$ matrices, as well as the user/item frequency distributions q and r. Consequently, if we want to capture the described drifts in the generated preferences, we need to modify these parameters accordingly. We can develop a versatile stream generator inspired by the concept of geometric closeness in the BPR model, which can effortlessly integrate the aforementioned drift models.

To begin, we define a set of nearly orthogonal generating vectors in K-dimensional space, denoted as $\mathbf{g}_1, \ldots, \mathbf{g}_h$ (i.e., designed such that $\mathbf{g}_i \cdot \mathbf{g}_j \approx 0$, for $i \neq j$). Conceptually, these vectors represent clusters of users and items. In practice, we can generate feature vectors $\mathbf{p}_u$ for users u (and $\mathbf{q}_i$ for items i) aligned with a specific generating vector $\mathbf{g}_s$. These user and item feature vectors are created by introducing random variations along the K dimensions based on $\mathbf{g}_s$. To generate preferences, we utilize the generateSamples function, which incorporates random user/item frequencies q and r accordingly.

Consequently, the aforementioned drifts can be effectively modeled by making appropriate adjustments to these feature vectors, as described below.

Drifting users: To capture changes in user preferences, we can model the shift in tastes for a user u with a feature vector $\mathbf{p}_u$ associated with the generating vector $\mathbf{g}_s$. This shift can be represented as a movement towards a different generating vector $\mathbf{g}_t$. Consequently, a new version of the user's feature vector $\mathbf{p}_u$ can be obtained by combining $\mathbf{g}_s$ and $\mathbf{g}_t$ through a linear combination.

New trends: To incorporate new trends, we introduce changes in the latent dimension K. Each generating vector $\mathbf{g}_s$ is extended with additional dimensions, denoted as k'. User and item feature vectors are extended accordingly, and this extension affects their geometric proximity.

Distribution drifts: To account for changes in user and item distributions, we can introduce new users and items by associating them with a generator vector and generating their respective feature vectors. Theoretical frequencies in q can be assigned to these new users/items. Alternatively, new generating vectors $\mathbf{g}_s$ can be created, allowing for the association of new items/users and the drift of existing items. User churns and item obsolescence can be modeled by annihilating their respective frequencies in q and r.

In Algorithm 2, we present a sample generation scheme that summarizes the aforementioned concepts using a set of policies for preference generation. We identify six main policies, and provide example implementations for some of these policies. They essentially introduce adjustments to generating vectors, feature matrices, or frequency distributions. By utilizing the modified feature matrices and/or frequency distributions, we can generate preferences for the shard associated with each policy.

Algorithm 2: Stream Generator

Input: Number h of cluster generating vectors; Cluster user size N_s and item size M_s, for $1 \leq s \leq h$; Number n of initial preferences to generate; A list of $T-1$ policies specific to drifting regimes and preferences to genarate n_t

Output: Preference models $\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(T)}$ and associated preferences $R^{(1)}, \ldots, R^{(T)}$.

$\mathbf{M}^{(1)}, G$ = `generateFeatureMatrices`$(\{N_1, \ldots, N_h\}, \{M_1, \ldots, M_h\})$
$R^{(1)}, q, r$ = `generateSamples`$(\mathbf{M}, n)$
for *each* $t \in \{2, \ldots, T\}$ **do**
  $\mathbf{M}^{(t)} = \mathbf{M}^{(t-1)}$
  if *the* t*-th regime contains 'user drift' with percentage* α **then**
    $\mathbf{M}^{(t)}, G$ = `coalesceDrift`$(\alpha, G, \mathbf{M}^{(t)})$
  end
  if *the* t*-th regime contains 'new trends' with dimension* k **then**
    $\mathbf{M}^{(t)}, G$ = `directionDrift`$(\alpha, G, \mathbf{M}^{(t)})$
  end
  if *the* t*-th regime contains 'new users' with dimension* k **then**
    $\mathbf{M}^{(t)}, q$ = `updateP`$(a, G, \mathbf{M}^{(t)})$
  end
  if *the* t*-th regime contains 'new items' with dimension* k **then**
    $\mathbf{M}^{(t)}, r$ = `updateQ`$(a, G, \mathbf{M}^{(t)})$
  end
  if *the* t*-th regime contains 'users churn' with percentage* α **then**
    q = `churnDrift`(α, q)
  end
  if *the* t*-th regime contains 'items expire' with percentage* α **then**
    r = `obsolescenceDrift`(α, r)
  end
  $R^{(t)}, q, r$ = `generateSamples`$(\mathbf{M}, n_t, q, r)$
end
Function `generateFeatureMatrices`$(\{N_1, \ldots, N_h\}, \{M_1, \ldots, M_h\})$:
  Sample $K > h$
  Generate h orthogonal vectors $\mathbf{g}_1, \ldots, \mathbf{g}_h \in \mathbb{R}^K$
  for *each* $s \in \{1, \ldots, h\}$ **do**
    for *each* $u \in \{1, \ldots, N_s\}$ *and* $i \in \{1, \ldots, M_s\}$ **do** Generate two random perturbations of $\mathbf{g}_s$ as $\mathbf{p}_u$ and $\mathbf{q}_i$
  end
  return $\mathbf{M}, \{\mathbf{g}_1, \ldots, \mathbf{g}_h\}$
Function `coalesceDrift`$(\alpha, G, \mathbf{M})$:
  Sample α% users from U into U'
  for *each* $u \in U'$ **do**
    Let $\mathbf{g}_s$ be the generating vector associated with u.
    Randomly sample $\mathbf{g}_t$ with $t \neq s$ from G and $\beta \in [0, 1]$
    $\mathbf{g}_{h+1} = \beta \mathbf{g}_s + (1-\beta)\mathbf{g}_t$; Add $\mathbf{g}_{h+1}$ to G
    Generate a random perturbations of $\mathbf{g}_t$ as $\mathbf{p}'_u$ and update $\mathbf{p}_u$ as $\beta \mathbf{p}_s + (1-\beta)\mathbf{p}'_u$
  end
  return $\mathbf{M}, G$
Function `directionDrift`$(k, G, \mathbf{M})$:
  for *each* $s \in \{1, \ldots, h\}$ **do**
    extend $\mathbf{g}_s$ with k new random dimensions
    for *each* $u \in \{1, \ldots, N_s\}$ *and* $i \in \{1, \ldots, M_s\}$ **do** extend $\mathbf{p}_u$ and $\mathbf{q}_i$ by using the random perturbations from the generating vectors
  end
  return $\mathbf{M}, G$
Function `updateP`$(k, G, \mathbf{M})$:
  Sample k generating vectors with replacement from G into G'
  for *each* $\mathbf{g} \in G'$ **do**
    Generate a random perturbation of $\mathbf{g}$ as $\mathbf{p}_u$ where u is a new user associated with $\mathbf{g}$
    Sample a random Zipf frequency q_u
  end
  return $\mathbf{M}, q$
Function `obsolescenceDrift`(α, r):
  Sample α% users from U into U'
  for *each* $u \in U'$ **do** update r_u by multiplying it with a random decay w
  return $\mathbf{M}, q$

6 Conclusions

In this study, we introduced a technique for modeling user/item preferences and their evolution over time using a dynamic tripartite graph representation. This graph is constructed as a sequence of matrix factorization models

trained on the source data stream, with each model capturing the properties and data distribution of a specific regime. The transition from one model to another occurs when a concept drift is detected.

We explored this concept in two main directions. Firstly, we utilized the learned sequence of models to generate synthetic data streams that exhibit similar properties to the original data. This approach enables the generation of realistic synthetic data streams, facilitating the development of more efficient and accurate recommendation systems. The experimental evaluation demonstrated the effectiveness of our approach in generating data streams that accurately reflect the dynamics of the underlying data distribution.

Building upon the insights from the preliminary study, we developed a more general approach capable of mapping specific drifting operations to changes in the underlying tripartite graphs. These changes can be parameterized, allowing for controlled data generation strategies where various drifting strategies can be implemented by manipulating the tripartite graph.

It is worth noting that our proposed approach is based on a specific preference model that assumes geometric closeness of users and items in the latent embedding space as a measure of interaction. As a direction for future research, it would be interesting to adapt the proposed scheme to other preference modeling strategies that do not rely on geometric closeness. For instance, exploring how the model can be tuned to work with Markovian models where preferences are correlated, and previous interactions can influence a user's future preferences.

Another fascinating topic is the study of strategies for managing new users and/or items. For instance, a new type of node could be added to the graph (Figure 1) to account for new entries and apply targeted strategies to model the behavior of new users effectively. Similarly, for items, one could consider incorporating business logic, such as prioritizing new, more profitable items. Also, these strategies could be considered a viable solution to the cold-start problem of the recommender system.

7 Acknowledgements

This work was partially supported by Project "SERICS - SEcurity and RIghts In the CyberSpace" (PE00000014) and partially by the EU H2020 ICT48 project "HumanE-AI-Net" under contract #952026.

References

[1] Charu C. Aggarwal. 2016. *Recommender Systems.* Springer.

[2] Marie Al-Ghossein, Talel Abdessalem, and Anthony Barré. 2021. A Survey on Stream-Based Recommender Systems. *ACM Comput. Surv.* 54, 5, Article 104 (2021).

[3] Manuel Baena-García, José del Campo-Ávila, Raul Fidalgo, Albert Bifet, Ricard Gavalda, and Rafael Morales-Bueno. 2006. Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams*, Vol. 6. Citeseer, 77–86.

[4] Cigdem Bakir. 2018. Collaborative filtering with temporal dynamics with using singular value decomposition. *Tehnički Vjesnik* 25, 1 (2018), 130–135.

[5] Nicola Barbieri, Giuseppe Manco, and Ettore Ritacco. 2014. *Probabilistic Approaches to Recommendations.* Morgan & Claypool Publishers.

[6] Francois Belletti, Karthik Lakshmanan, Walid Krichene, Yi-Fan Chen, and John Anderson. 2019. Scalable realistic recommendation datasets through fractal expansions. *arXiv preprint arXiv:1901.08910* (2019).

[7] James Bennett, Charles Elkan, Bing Liu, Padhraic Smyth, and Domonkos Tikk. 2007. KDD Cup and Workshop 2007. *SIGKDD Explor. Newsl.* 9, 2 (2007), 51–52.

[8] Albert Bifet and Ricard Gavalda. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 443–448.

[9] María del Carmen Rodríguez-Hernández, Sergio Ilarri, Ramón Hermoso, and Raquel Trillo-Lado. 2017. DataGenCARS: A generator of synthetic data for the evaluation of context-aware recommendation systems. *Pervasive and Mobile Computing* 38 (2017), 516–541.

[10] Jörg Drechsler and Jerome P Reiter. 2011. An empirical evaluation of easily implemented, nonparametric methods for generating synthetic datasets. *Computational Statistics & Data Analysis* 55, 12 (2011), 3232–3243.

[11] Eduardo Ferreira José, Fabricio Enembreck, and Jean Paul Barddal. 2020. ADADRIFT: An Adaptive Learning Technique for Long-history Stream-based Recommender Systems. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2593–2600.

[12] Isvani Frías-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Rafael Morales-Bueno, Agustín Ortiz-Díaz, and Yailé Caballero-Mota. 2015. Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. *IEEE Transactions on Knowledge and Data*

Engineering 27, 3 (2015), 810–823. https://doi.org/10.1109/TKDE.2014.2345382
[13] Joao Gama. 2010. *Knowledge Discovery from Data Streams.* Chapman & Hall/CRC.
[14] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*. Springer, 286–295.
[15] João Gama, Indre Žliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *Comput. Surveys* 46, 4 (2014).
[16] Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. 2017. Adaptive random forests for evolving data stream classification. *Machine Learning* 106, 9 (2017), 1469–1495.
[17] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. 2019. Machine Learning for Streaming Data: State of the Art, Challenges, and Opportunities. *SIGKDD Explor. Newsl.* 21, 2 (2019), 6–22.
[18] Xiao Han, Chen Zhu, Xiao Hu, Chuan Qin, Xiangyu Zhao, and Hengshu Zhu. 2024. Adapting Job Recommendations to User Preference Drift with Behavioral-Semantic Fusion Learning. *arXiv preprint arXiv:2407.00082* (2024).
[19] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4 (2015).
[20] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. 2014. *Nonparametric Statistical Methods.* John Wiley & Sons.
[21] Martin Jakomin, Tomaž Curk, and Zoran Bosnić. 2018. Generating inter-dependent data streams for recommender systems. *Simulation Modelling Practice and Theory* 88 (2018), 1–16.
[22] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *IEEE International Conference on Data Mining*. IEEE, 197–206.
[23] Anastasiia Klimashevskaia, Dietmar Jannach, Mehdi Elahi, and Christoph Trattner. 2024. A Survey on Popularity Bias in Recommender Systems. *User Modeling and User-Adapted Interaction* (2024).
[24] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 447–456.
[25] Lukasz Korycki and Bartosz Krawczyk. 2019. Unsupervised Drift Detector Ensembles for Data Stream Mining. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 317–325.
[26] Jure Leskovec, Anand Rajaraman, and Jeffrey Ullman. 2020. *Mining Massive Data.* Cambridge University Press.
[27] Elisabeth Lex, Dominik Kowald, and Markus Schedl. 2020. Modeling Popularity and Temporal Drift of Music Genre Preferences. *Trans. Int. Soc. Music. Inf. Retr.* 3, 1 (2020), 17–30.
[28] Jiacheng Li, Yujie Wang, and Julian McAuley. 2020. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 322–330.
[29] Pawel Matuszyk and Myra Spiliopoulou. 2017. Stream-based semi-supervised learning for recommender systems. *Machine Learning* 106, 6 (2017), 771–798.
[30] Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. 2018. Forgetting techniques for stream-based matrix factorization in recommender systems. *Knowledge and Information Systems* 55, 2 (2018), 275–304.
[31] Diego Monti, Giuseppe Rizzo, and Maurizio Morisio. 2019. All You Need is Ratings: A Clustering Approach to Synthetic Rating Datasets Generation. *REVEAL Workshop at ACM RecSys Conference* (2019).
[32] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 188–197.
[33] Stanley Ade Oyewole. 2021. *Image content-based user preference elicitation for personalised mobile recommendation of shopping items.* Ph. D. Dissertation. Department of Information Technology, Faculty of Accounting and Informatics, Durban University of Technology Durban, South Africa.
[34] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Conference on Uncertainty in Artificial Intelligence (UAI '09)*. 452–461.
[35] Francesco Ricci, Lior Rokach, and Bracha Shapira (Eds.). 2015. *Recommender Systems Handbook* (2nd ed.). Springer Publishing Company, Incorporated.
[36] Raquel Sebastião and José Maria Fernandes. 2017. Supporting the Page-Hinkley test with empirical mode decomposition for change detection. In *Foundations of Intelligent Systems: 23rd International Symposium, ISMIS 2017, Warsaw, Poland, June 26-29, 2017, Proceedings 23*. Springer, 492–498.
[37] Manel Slokom, Martha Larson, and Alan Hanjalic. 2020. Partially Synthetic Data for Recommender Systems: Prediction Performance and Preference Hiding. *arXiv preprint arXiv:2008.03797* (2020).
[38] Ricardo Sousa and Joao Gama. 2016. First Principle Models Based Dataset Generation for Multi-Target Regression and Multi-Label Classification Evaluation.. In *STREAMEVOLV@ ECML-PKDD*.
[39] Huimin Sun, Jiajie Xu, Kai Zheng, Pengpeng Zhao, Pingfu Chao, and Xiaofang Zhou. 2021. MFNP: A Meta-optimized Model for Few-shot Next POI Recommendation.. In *IJCAI*, Vol. 2021. 3017–3023.

[40] Tipajin Thaipisutikul. 2022. An adaptive temporal-concept drift model for sequential recommendation. *ECTI Transactions on Computer and Information Technology (ECTI-CIT)* 16, 2 (2022), 222–236.
[41] Karen HL Tso and Lars Schmidt-Thieme. 2006. Empirical analysis of attribute-aware recommender system algorithms using synthetic data. *J. Comput.* 1, 4 (2006), 18–29.
[42] Alexey Tsymbal. 2004. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin* 106, 2 (2004), 58.
[43] Bruno Veloso, Luciano Caroprese, Matthias König, Sónia Teixeira, Giuseppe Manco, Holger H. Hoos, and João Gama. 2021. Hyper-parameter Optimization for Latent Spaces. In *Procs. ECML/PKDD 2021 European Conference on Machine Learning and Knowledge Discovery in Databases (Lecture Notes in Computer Science, Vol. 12977)*. 249–264.
[44] João Vinagre, Alípio Mário Jorge, Conceição Rocha, and Joao Gama. 2021. Statistically Robust Evaluation of Stream-Based Recommender Systems. *IEEE Transactions on Knowledge and Data Engineering* 33, 7 (2021), 2971–2982.
[45] João Vinagre, Alípio Mário Jorge, and João Gama. 2018. Online Gradient Boosting for Incremental Recommender Systems. In *Discovery Science*. 209–223.
[46] Charinya Wangwatcharakul and Sartra Wongthanavasu. 2021. A novel temporal recommender system based on multiple transitions in user preference drift and topic review evolution. *Expert Systems with Applications* 185 (2021), 115626.
[47] Geoffrey I. Webb, Loong Kuan Lee, François Petitjean, and Bart Goethals. 2017. Understanding Concept Drift. *CoRR* abs/1704.00362 (2017). arXiv:1704.00362
[48] Taoru Yang, Yong Gao, Zhou Huang, and Yu Liu. 2023. UPTDNet: A User Preference Transfer and Drift Network for Cross-City Next POI Recommendation. *International Journal of Intelligent Systems* 2023, 1 (2023), 9091570.
[49] Zhengyi Yang, Xiangnan He, Jizhi Zhang, Jiancan Wu, Xin Xin, Jiawei Chen, and Xiang Wang. 2023. A Generic Learning Framework for Sequential Recommendation with Distribution Shifts. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Received 15 December 2022; revised 30 July 2024; accepted 17 September 2024