A Survey on Stream-Based Recommender Systems

MARIE AL-GHOSSEIN, University of Helsinki
TALEL ABDESSALEM, LTCI, Télécom Paris, Institut Polytechnique de Paris
ANTHONY BARRÉ, AccorHotels

Recommender Systems (RS) have proven to be effective tools to help users overcome information overload, and significant advances have been made in the field over the past two decades. Although addressing the recommendation problem required first a formulation that could be easily studied and evaluated, there currently exists a gap between research contributions and industrial applications where RS are actually deployed. In particular, most RS are meant to function in batch: they rely on a large static dataset and build a recommendation model that is only periodically updated. This functioning introduces several limitations in various settings, leading to considering more realistic settings where RS learn from continuous streams of interactions. Such RS are framed as Stream-Based Recommender Systems (SBRS).

In this article, we review SBRS, underline their relation with time-aware RS and online adaptive learning, and present and categorize existing work that tackle the corresponding problem and its multiple facets. We discuss the methodologies used to evaluate SBRS and the adapted datasets that can be used, and finally we outline open challenges in the area.

CCS Concepts: • **Information systems → Recommender systems**;

Additional Key Words and Phrases: Personalization, recommendation, data streams, concept drifts

ACM Reference format:
Marie Al-Ghossein, Talel Abdessalem, and Anthony Barré. 2021. A Survey on Stream-Based Recommender Systems. *ACM Comput. Surv.* 54, 5, Article 104 (May 2021), 36 pages.
https://doi.org/10.1145/3453443

1 INTRODUCTION

Recommender Systems (RS) [118] are tools and techniques providing personalized suggestions of items for users, the items being drawn from a large catalog. There has been extensive research trying to tackle the different aspects and challenges of RS especially due to their usefulness in scenarios where users are subject to information overload. They have wide applicability since they can increase core business metrics [149] and have been used to recommend movies, videos, music, and friends, among others.

This work was done while M. Al-Ghossein was at Télécom Paris, France.
Authors' addresses: M. Al-Ghossein (corresponding author), University of Helsinki, Faculty of Science, Kumpula Campus, PL 68 (Pietari Kalmin katu 5), 00014 Helsingin Yliopisto, Finland; email: marie.al-ghossein@helsinki.fi; T. Abdessalem, Télécom Paris, 19 place Marguerite Perey, 91120 Palaiseau, France; email: talel.abdessalem@telecom-paristech.com; A. Barré, AccorHotels, 82 rue Henri Farman, 92130 Issy-les-Moulineaux, France; email: anthony.barre@live.fr.

0360-0300/2021/05-ART104 $15.00
https://doi.org/10.1145/3453443

When relying on the widely used **Collaborative Filtering (CF)** techniques [85], RS analyze the past behavior of individual users that reveals their preferences toward certain items and that is recorded under various forms of actions like clicks, ratings, and purchases. This feedback and the underlying patterns are then used to predict items' relevances and compute recommendations matching the user profiles. Although RS apply techniques from various domains such as information retrieval and human computer interaction, the system's core can be viewed as an instance of a data mining process [18]: data related to users and items is analyzed, a model is inferred, and recommendations are produced for each user. The model is trained using all recorded observations, and the system is then queried for recommendations. CF techniques that do not rely on an inferred model to compute recommendations, i.e., memory-based approaches [45], usually compute and store some statistics or aggregated information based on the raw data, e.g., similarities between users or items, that are then used to generate recommendations in real time. Even though this introduction is focused on recommendation "models", the same ideas apply for this aggregated information that is stored under a static form in practice [93] and are further elaborated in the article.

RS are deployed online where data is continuously generated, e.g., news articles continuously published, users continuously posting on social media, watching videos, or buying products on an e-commerce platform. Recommendation models ensuring a good recommendation quality tend to suffer from very high latency. Training on large datasets of user interactions is computationally expensive in terms of space and time and cannot be performed each time a new observation is received. In practice, most RS are built using all of the observations that have been previously recorded and are only rebuilt or updated periodically, as new observations are added to the dataset.

This functioning in batch faces two limitations in particular. First, the RS cannot account for the user interactions observed after a model update until the following one is performed and cannot thus consider the continuous activity of users. Although user preferences and item perceptions are dynamic concepts that are constantly changing over time, the same model is used to deliver recommendations and is indifferent to these temporal dynamics. The inability to consider recent observations, including new items and new interactions, leads to poor recommendation quality and user experience as the system is unable to adapt to changes [47]. Second, as the dataset is continuously growing over time, the periodic retraining becomes computationally expensive. This raises scalability issues in addition to the need for additional resources to store and process data. Solution options include scaling up systems, which can be expensive; reducing the number of updates, which can worsen recommendations; and decreasing the volume of data considered, which may result in discarding valuable information for recommendation.

Advances in the RS field have focused on a part of these limitations and attempted to address each one of them using different approaches such as time-aware RS [35] and distributed models [61, 132]. However, these solutions do not address both limitations mentioned earlier, as they are not designed to handle new observations and model temporal dynamics simultaneously. Recent work [123, 138] has considered to frame the problem of recommendation as a data stream problem by designing **Stream-Based Recommender Systems (SBRS)**, i.e., RS that handle user and item data as a stream of information or observations. Observations in a data stream are expected to be received indefinitely in real time at high uncontrolled rates. A parallel can be drawn with user-generated data handled by RS that share the same characteristics. SBRS are required to deal with continuous data streams and to maintain recommendation models up to date, and rely usually on incremental learning. By adapting to changes in real time and operating independently of the number of observations, they address the previously mentioned limitations of batch RS.

Data stream mining faces the central problem of dealing with concept drifts, which happen when the definition of the target concept shifts over time [59]. The problem is even more

complicated in SBRS that are expected to handle several concepts, such as users' preferences and items' perception, that are drifting at different moments and rates and in different ways. Being unable to account for these drifts results in outdated models that do not represent the current states of users and items and fail to generate relevant recommendations. However, SBRS have to update the underlying models and compute recommendations efficiently to be able to operate in real time.

Overall, SBRS are more realistic than traditional RS in dynamic settings, and although existing work, i.e., the work surveyed in this article, has tried to address different aspects related to SBRS, research in the field remains relatively scattered and some important questions have not been addressed. In this survey, we review existing work on SBRS, discuss the differences with other families of RS, and highlight the limitations of existing approaches.

Relation with existing surveys. To the best of our knowledge, this is the first survey reviewing SBRS, which is important at this stage to centralize existing work and in an attempt to pave the way for future research on the subject. Although several surveys related to RS have been presented, only two of them are related to our work. The first one, by Vinagre et al. [140], considers the exploitation of the time dimension in CF and presents an analysis of the algorithms that take into account temporal effects in RS, e.g., drifts of user preferences and availability of new items. The second one, by Benczúr et al. [25], briefly presents the topic of SBRS in addition to the evaluation methodology of such systems and to one particular recommendation method that could be used to learn preferences and generate recommendations in such settings. In comparison to both of these surveys, this is the first work that provides a classification of existing works related to SBRS, covering a large number of proposed solutions and pointing out limitations that need to be addressed in future work.

Organization of the article. The article is organized as follows. In Section 2, we present the characteristics of SBRS and a corresponding algorithmic description, and in Section 3, we present the relations of SBRS with other families of RS and other areas. In Section 4, we present the classification of existing work that we adopt in this survey. Section 5 reviews existing approaches for SBRS, and Section 6 discusses methodologies for evaluating SBRS. Finally, Section 7 concludes the article and gives directions for future research.

2 CHARACTERISTICS OF SBRS

In Section 2.1, we provide general information about RS to contrast the classical setting with SBRS and also to support the concepts mentioned later in the survey. We then present the characteristics of SBRS in Section 2.2 and an algorithmic description in Section 2.3.

2.1 General Information About RS

Whereas the final goal of RS is to provide items that are interesting for the users, studying the problem requires a formulation that can be evaluated and quantified. One first formulation, known as *rating prediction*, consists of predicting the rating that a user will attribute to an item they did not rate and evaluating the error of predictions. A second formulation, known as *top-N recommendation*, focuses on the quality of recommended items. In both cases, the core of the RS is expected to follow the same process [10]. The recommender algorithm predicts the utility of each item, i.e., the rating or the relevance score used to rank items, for a target user. Items chosen then for recommendation are those maximizing the predicted utility.

However, the user feedback that is being collected can be of two types. *Explicit feedback* is the form of feedback that is directly reported by the user to the system and is often provided in the form of numerical ratings. *Implicit feedback* is collected by the system without actively asking the user, e.g., clicks on items or item purchases.

The development of RS has mostly focused on the batch setting where the whole dataset is available for training and where the recommendation model remains static for a certain period of time until retraining is performed. This practice has led to a gap with some real-world environments where RS are deployed and where user-generated data could arrive as a data stream.

When considering recommendation approaches that model the relationships between items in domains where those are stable and where there is enough data available, one does not necessarily need to build SBRS and take into account new interactions as soon as they are generated. However, SBRS are essential in dynamic domains such as those that rely on user-generated items, e.g., social media and news, and those where the concepts continuously evolve and where new interactions arriving at high rates should be constantly considered.

2.2 Online Adaptive Learning in the Streaming Setting

We describe here the setting in which SBRS operate and highlight the requirements it imposes on these systems.

Streaming setting. The fact that data arrives continuously, indefinitely, at very fast rates, and under the form of streams in multiple environments has led to new research challenges in terms of analysis, computation, and storage. In particular, and compared to traditional data mining, mining data streams introduces the following differences, highlighted in work of Gama [57]:

- Observations arrive online.
- The system does not control the order nor the rate in which observations arrive.
- Data streams are possibly unbounded.
- Once an observation from the stream is processed, it is either discarded or archived in a memory which size is limited and relatively small in comparison with the size of the stream. Only observations stored in the memory can be retrieved when needed.

Requirements for SBRS. These differences raise a number of requirements that a typical system handling data streams should satisfy [51]. We mention these requirements in the following, with a specific focus on the case of SBRS.

First, a single pass over user-generated data should be necessary to build the recommendation model. A small number of observations can be stored for further analysis, knowing that the main memory is bounded and its size is independent of the total number of observations processed. Therefore, incremental or online learning should be adopted [94]. The main idea is to learn user preferences and item perceptions incrementally, by processing observations one at a time. It would then be possible to update the recommendation model after receiving an observation without access to past data.

Second, handling data streams imposes some requirements related to the computing performance. In particular, observations should be processed faster than their rate of arrival and SBRS should be available to generate recommendations anytime. Distributed systems can be considered and SBRS could benefit from solutions developed for distributed data stream processing [24].

Finally, the implications of handling data streams go beyond efficiently processing and learning from data: SBRS have to rely on models that evolve over time, as the underlying represented concepts are themselves evolving and changing at different rates and in different ways. Recommendation models are thus expected to handle and adapt to drifts, leading to online adaptive learning [59].

Online adaptive learning. The online adaptive procedure consists of the following three steps, applied for each received observation:

(i) *Predict*: Recommend N items to the user u that is observed.
(ii) *Evaluate*: After observing the true interaction of the user u, evaluate the quality of the list of items that was recommended.
(iii) *Update*: The true interaction may be used to update the recommendation model.

2.3 Algorithmic Description of SBRS

In an attempt to clearly define the problem considered, we describe here the process followed by SBRS under an algorithmic form, contrasting it with the one followed by traditional or batch RS.

ALGORITHM 1: Functioning of batch RS

Input: Initial set of available interactions $\mathcal{D}^0$, Time period for model retraining T

$\mathcal{D} \leftarrow \mathcal{D}^0$
$\mathcal{M} \leftarrow f_{train}(\mathcal{D})$ ▷ f_{train} returns a model $\mathcal{M}$ trained on $\mathcal{D}$
function RETRAIN(T)
  for each time period T **do**
    $\mathcal{M} \leftarrow f_{train}(\mathcal{D})$
  end for
end function
function RECOMMEND( )
  for each received user u at time t **do**
    $list = retrieve(\mathcal{M}, u)$ ▷ $retrieve$ retrieves recommendations for u using $\mathcal{M}$
    Observe user behavior, i.e., feedback r_{ui} on item i
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{u, i, r_{ui}, t\}$ ▷ *add* the tuple to $\mathcal{D}$
  end for
end function
RECOMMEND() & RETRAIN(T)

ALGORITHM 2: Functioning of SBRS

Input: Initial set of available interactions $\mathcal{D}^0$

$\mathcal{M} \leftarrow f_{train}(\mathcal{D}^0)$ ▷ For initialization; f_{train} returns a model $\mathcal{M}$ trained on $\mathcal{D}^0$
$\mathcal{D}_{mem} \leftarrow update(\mathcal{D}^0)$ ▷ Optional; $update$ implements the storing strategy of observations
function TRAIN($\mathcal{M}, \{u, i, r_{ui}, t_{ui}\}, [\mathcal{D}_{mem}]$)
  $\mathcal{M} \leftarrow g_{train}(\mathcal{M}, \{u, i, r_{ui}, t_{ui}\}, [\mathcal{D}_{mem}])$ ▷ g_{train} incrementally updates $\mathcal{M}$ and returns it
end function
function RECOMMEND( )
  for each received user u at time t **do**
    $list = retrieve(\mathcal{M}, u)$ ▷ $retrieve$ retrieves recommendations for u using $\mathcal{M}$
    Observe user behavior, i.e., feedback r_{ui} on item i
    TRAIN($\mathcal{M}, \{u, i, r_{ui}, t\}, [\mathcal{D}_{mem}]$)
    $\mathcal{D}_{mem} \leftarrow update(\mathcal{D}_{mem} \cup \{u, i, r_{ui}, t\})$ ▷ Optional
  end for
end function
RECOMMEND()

Notations. We first provide a set of notations that we will be using throughout the article to illustrate the ideas. We denote by $\mathcal{U}$ the set of users handled by the system and by n the number

of users $|\mathcal{U}|$. We denote by $\mathcal{I}$ the set of items contained in the catalog and by m the number of items $|\mathcal{I}|$. The feedback data is encoded in a matrix $\mathbf{R}$ of size $n \times m$, called the *rating matrix*. The entry r_{ui} represents the feedback given by user u to item i, if any feedback is given. In settings where users provide explicit feedback in the form of numerical ratings, r_{ui} takes the value of the rating given by user u to item i. When considering implicit feedback, r_{ui} takes the value 1 if user u interacted with item i. Computing recommendations requires predicting the values of the missing elements from $\mathbf{R}$, and a predicted value is denoted by $\hat{r}_{ui}$. More generally, and given a matrix $\mathbf{A}$, $\mathbf{a}_x$ is a column vector that is obtained by selecting the xth column from $\mathbf{A}$. The set of users who rated item i is denoted by $\mathcal{U}_i$, and the set of items rated by user u is denoted by $\mathcal{I}_u$. Finally, we denote by $\mathcal{D}$ the set of observed interactions, having that one interaction is represented by the tuple (u, i, r_{ui}, t_{ui}), where u denotes the individual user, i the item, r_{ui} the corresponding feedback, and t_{ui} the timestamp at which the interaction happened. Whereas there are several types of interactions that one can collect to record the user feedback, e.g., clicks, likes, and purchases, we avoid introducing an additional notation regarding the interaction type, as this information will not be exploited in the article. Note that in such cases, elements of the matrix $\mathbf{R}$ represent the overall feedback when aggregating the different interactions for each couple of user and item.

Given that we are considering a dynamic setting, every introduced notation will be used with a superscript t, when applicable, denoting the time t at which we are evaluating the variable, e.g., $\mathcal{U}^t$ denotes the set of users available at time t. In particular, this will be used to distinguish between the states of one variable at different points in time, and will be omitted when dealing with one single point in time, to facilitate readability.

The overall structure followed by batch RS and SBRS is presented in Algorithm 1 and Algorithm 2, respectively. Batch RS are based on two functions that are launched in parallel. The first one, Retrain (Algorithm 1, line 3), retrains the recommendation model every period of time T based on the collected observations included in $\mathcal{D}$. The second one, Recommend (Algorithm 1, line 8), retrieves recommendations for each received user based on the last trained model and stores the user feedback in $\mathcal{D}$. In contrast, and for SBRS, each time we observe the user feedback, the model is incrementally updated based on its current state and on the user feedback (Algorithm 2, line 10). We note that it is possible to keep a limited-size memory of observations, $\mathcal{D}_{mem}$, and exploit it when updating the model. The content of this memory is handled and updated using the function *update* (Algorithm 2, lines 2 and 11).

Given the link between SBRS and multiple other topics in the RS and data mining fields, in the following section we provide the positioning of SBRS with regard to related areas.

3 RELATIONS WITH OTHER AREAS

This section exposes the relations of SBRS with, on one hand, the time dimension in RS used to model the evolution of concepts, and on the other hand, concept drifts in data stream mining.

3.1 Time Dimension in RS

When learning user and item models based on a dataset of user interactions, traditional recommendation approaches assume that the user behavior follows a single specific pattern that is expected to reappear in the future, making prediction a feasible task. This assumption does not hold in a dynamic world where users and items are constantly evolving. Efficient RS are expected to maintain models that reflect the actual state of entities [84]. However, modeling temporal changes related to users and items is a challenging task. Changes may occur on a global scale, e.g., introduction of new items or seasonal changes, and may also be driven by local factors related to individual users, e.g., changes in the user social status.

Historical perspective. The practical value of modeling temporal changes in RS was first highlighted within the context of the Netflix Prize [26]. Koren [84] showed that user ratings exhibit temporal patterns that could be exploited for the recommendation task. The proposed solution allowed the distinction between long-term patterns and temporary ones and was evaluated on the dataset of the competition. Including temporal changes proved to be very useful to improve the recommendation quality and several approaches were further proposed, forming the category of RS known as **Time-Aware Recommender Systems (TARS)** [35]. Then, the rise of **Context-Aware Recommender Systems (CARS)** lead to investigating the possibility of integrating time as a contextual dimension. One of the first efforts following this direction [21] proposed to build contextual profiles representing the user in multiple temporal contexts, e.g., morning and evening or weekend and workday. This technique was then identified as being part of the pre-filtering paradigm for CARS. However, recent years have witnessed the emergence of a related problem, which is the problem of *sequence-aware recommendation* [113]. The increased interest was mainly fueled by industry through the release of a new dataset of session activity from a major e-commerce platform within the context of the 2015 ACM RecSys Challenge [23], and by the development of new deep learning techniques for sequence modeling [69]. Sequence-aware algorithms learn sequential patterns from user behavior and attempt to predict the user's next interaction within a session or to detect short-term tendencies. TARS and sequence-aware RS share a number of common characteristics. However, although they both rely on the order of interactions, sequence-aware RS do not focus on the exact timing of past observations and rather try to detect patterns within the sequence itself.

Time-Aware Recommender Systems. Several methods were proposed to integrate time into RS [35]. We can mainly distinguish between three categories of approaches merging time with CF methods. First, *recency-based time-aware* models assume that recent observations are more pertinent than the older ones since they are more representative of the actual reality: they should therefore be given more importance in the learning and recommendation processes. Then, *contextual time-aware* models integrate the time dimension by considering temporal information as a contextual dimension. Finally, *continuous time-aware* models represent user interactions as a continuous function of time, and the model parameters are learned from the data.

Relation to SBRS. While considering the evolution of modeled concepts over time, TARS are not adapted to a streaming environment and cannot maintain models up to date in real time. They assume that the whole dataset is available at the time of training and that multiple passes can be performed on the data, and thus cannot be used in streaming contexts. The same applies for sequence-aware RS, in addition to the fact that in such RS, the focus is on the sequence of ordered interactions that is assumed to exhibit patterns leveraged for recommendation. Whereas sequence-aware RS focus on the series of ordered interactions of each user or session individually, SBRS consider all of the ordered interactions of all users in one pass.

3.2 Data Stream Mining and Online Adaptive Learning

With the massive and growing generation of information and signals, data is commonly available as continuous streams that can only be appropriately processed in an *online* mode. In contrast with the *offline* mode where the whole dataset is available for training and where the model is only used once the learning is completed, the online setting requires the data to be processed sequentially and the models to be continuously trained when receiving new observations. Aside from the online nature of learning, handling data streams presents the challenge of detecting and adapting to concept drifts [133]. Concept drift occurs in non-stationary environments when the data distribution shifts over time and imposes the design of adaptive models.

Concept drift. The main assumption made by most predictive models is that the training data, i.e., observations used for learning, and the testing data, i.e., observations for which the model will be generating predictions, are sampled from the same distribution. This assumption does not hold in realistic environments where predictive models are meant to be developed and applied, due to the evolution of target concepts over time. Learning is often performed in non-stationary environments and thus requires adaptive models that are able to account for changes occurring unexpectedly over time.

The process of learning in non-stationary environments is formalized for the problem of supervised learning [50, 59] where the goal is to predict a target variable $y \in \mathbb{R}$ or the class y given the input features $\mathbf{X} \in \mathbb{R}^n$. Observations that are used for learning, denoted by $(\mathbf{X}, y)$, and that are generated at time t are assumed to be sampled from the joint probability distribution $p_t(\mathbf{X}, y)$. We denote by $p_t(y|\mathbf{X})$ the posterior distribution of class y and by $p_t(\mathbf{X})$ the evidence distribution. Concepts are expected to evolve over time, and consequently, distributions change dynamically. A concept drift occurring between time points t_0 and t_1 is defined as

$$\exists \mathbf{X} : p_{t_0}(\mathbf{X}, y) \neq p_{t_1}(\mathbf{X}, y). \tag{1}$$

Given that different natures of changes may be encountered, a terminology is adopted to designate where the change originates from in addition to its form [59]. Concerning the distribution from which the change is originating, we distinguish two types of drifts: *real concept drifts*, which occur when the posterior probability $p_t(y|\mathbf{X})$ changes over time, and *virtual concept drifts*, which occur when the evidence probability $p_t(\mathbf{X})$ changes. In all cases, handling concept drifts requires relying on online adaptive learning.

Framework for online adaptive learning. Gama et al. [59] proposed a general outline for an online adaptive learning algorithm, which consists of four modules, mentioned in the following. The *memory* module defines the strategies followed to select data for learning on one hand and to discard old data on the other hand. The *learning* module defines how to build predictive models and maintain them up to date given the received observations. The *change detection* module handles techniques for the active detection of drifts. The *loss estimation* module analyzes the performance of the algorithm and reports it to other modules.

Relation to SBRS. Most works related to RS consider learning predictive models in an offline mode. This setting is not adapted to real-world environments where user-generated data arrives in the form of streams and where concepts are expected to change over time. Compared to data stream mining problems that focus mainly on modeling one concept, SBRS track several entities evolving over time, i.e., users and items. In the following, we rely on the formulation of online adaptive learning and on the framework developed for data stream mining to review previous work related to SBRS.

4 CLASSIFICATION OF EXISTING WORK ON SBRS

As mentioned in previous sections, some real-world settings require formulating the problem of recommendation as a data stream problem: user-generated data is continuously received and analyzed and recommendations are delivered in real time. Recently, increased attention has been attributed to the topic of SBRS. In particular, several workshops tackling issues related to SBRS have been organized during the past few years in the context of high-profile conferences, e.g., RecProfile in 2016 [109], RecTemp in 2017 [28], ORSUM in 2018 [75], and ORSUM in 2019 [136].

In this survey, we classify existing work to ensure a better understanding of the issues that have been addressed, the solutions that have been proposed, and the aspects of SBRS that require further investigation. Whereas Section 5 focuses on the approaches proposed for learning preferences and

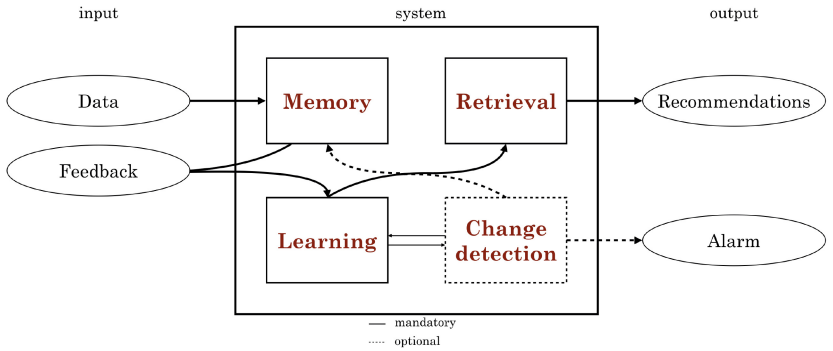

Fig. 1. A general framework for SBRS.

generating recommendations in the scope of SBRS, Section 6 discusses evaluation methodologies designed to assess the performance of SBRS.

How do we collect the papers? Practically, we believe that the topic of SBRS is related to the following problems: incremental learning, online adaptation to concept drifts, and efficiency in RS. Therefore, the major keywords we used for searching for papers included recommender system, recommendation, streaming, online, incremental, concept drift, collaborative filtering, and real-time.

We manually analyzed the first 100 results obtained by searching on the ACM Digital Library and on Google Scholar and by sorting them with respect to relevance. We also manually considered the proceedings of relevant conferences such as WWW, KDD, SIGIR, CIKM, WSDM, and RecSys, published between 2010 and 2019, to find out works related to SBRS by searching for keywords in abstracts and reading them. We also considered the related work exposed in the relevant papers.

Classification of existing work. Inspired by the work of Gama et al. [59], and to classify existing work, we consider SBRS to be designed according to the framework illustrated in Figure 1. Whereas the framework proposed by Gama et al. [59] is presented to perform supervised learning, we focus on SBRS and propose a framework that takes into account the different components of an RS.

The framework we adopt is constituted of four modules, i.e., components [13]: the *memory* module that selects the data that will be fed to the model, the *learning* module that defines how we build the recommendation model, the *change detection* module that is responsible for detecting concept drifts, and the *retrieval* module that handles the retrieval of recommendations for individual users. Given that there are only a few works focusing on *change detection* for SBRS, we discard the *loss estimation* module originally proposed by Gama et al. [59]. We consider that previous work related to SBRS attempted to come up with solutions regarding one or several modules at once, and we thus propose to review approaches for SBRS according to the main module concerned by the solution. To improve readability of the article and to avoid confusion, Table 1 provides a list of abbreviations commonly used throughout the article. In addition, Table 2 provides a summary of most of the papers considered in this work, along with their characteristics.

5 APPROACHES FOR SBRS

In this section, we review existing approaches for SBRS according to the framework presented in Figure 1. We also separately review specific frameworks that propose a solution tailored to a particular domain.

Table 1. List of Abbreviations Commonly Used Throughout the Article

Abbreviation	Meaning	Abbreviation	Meaning
CARS	Context-Aware Recommender Systems	MIPS	Maximum Inner Product Search
CF	Collaborative Filtering	RS	Recommender Systems
DCG	Discounted Cumulative Gain	SBRS	Stream-Based Recommender Systems
IMF	Incremental Matrix Factorization	SGD	Stochastic Gradient Descent
LSH	Locality Sensitive Hashing	SVD	Singular Value Decomposition
MF	Matrix Factorization	TARS	Time-Aware Recommender Systems

5.1 Memory Module

The memory module is responsible for selecting the data used for learning and to discard old data that is no longer pertinent. Assuming that recent data is more relevant than older data, predictive models have to learn from recent observations and forget old ones. This idea goes beyond being a simple intuition and is supported by findings in psychology and implemented in psychological-inspired RS [19, 87].

The *forgetting* procedure has been applied in the context of SBRS in several ways. Even though forgetting is a strategy used for handling memory in online adaptive learning [59], it is essential to note that, in the field of RS, this type of strategy was originally exploited for designing TARS [35] and its application is not proper to SBRS. Nevertheless, here we report how it is implemented in the context of SBRS in addition to its implications.

Forgetting is used for SBRS in the scope of different CF approaches, which are traditionally divided into two classes: memory-based and model-based approaches.

5.1.1 Forgetting in Memory-Based Approaches. Memory-based approaches for CF are based on the idea that similar users are interested in the same items or that similar items interest the same users, and can thus be user based or item based [45]. In a user-based approach, and to select items to suggest to user u, we refer to users who are the most similar to u. In an item-based approach [44], we recommend items that are similar to the items that u has rated in the past. In both cases, a neighborhood method first determines the neighbors of an entity using a similarity measure and then relies on the neighbors and on previous user interactions to generate recommendations. Although all data available is used to compute these similarities and the predicted scores in traditional settings, some approaches relied on *forgetting* strategies to keep the models up to date and to reduce the size of the data used for computation.

Abrupt forgetting is adopted by totally discarding observations based on their relevance and by using sliding windows: only recent observations included in the sliding window are considered. Using sliding windows requires setting its size, which can be defined in terms of time intervals, e.g., 1 day, or in terms of number of interactions, e.g., five interactions. Nasraoui et al. [107] first use a user-based neighborhood approach that relies on a sliding window containing a fixed number of recent observations. Similarities between users are thus computed only based on their recent interactions, i.e., interactions performed during the last hour. Along this line, Siddiqui et al. [123] propose a recommendation algorithm that operates in two steps. Users are first clustered with respect to their previous interactions, and a user-based neighborhood approach is then applied in each cluster to perform recommendations. A sliding window is used over the stream: only observations made during a certain time interval are considered when computing similarities and scores, and the others are definitely discarded. Experiments on synthetic and real data show that

Table 2. Categorization of Works Related to SBRS

Paper *(Year)*	**Focus**	**Techniques**	**Types of Feedback**	**Domains**
Agarwal et al. [12] *(2010)*	LRN	MF	EXP	MOV
Al-Ghossein et al. [15] *(2018)*	CHG	MF	IMP	MOV, NEWS
Al-Ghossein et al. [14] *(2018)*	CHG	KNN	IMP	MOV
Bachrach et al. [20] *(2014)*	RTR	MF	EXP	EC, MUS
Brand [31] *(2003)*	LRN	MF	EXP	Others
Chandramouli et al. [38] *(2011)*	LRN	KNN	EXP	EC, NEWS
Chang et al. [39] *(2017)*	LRN	SPM	EXP	MOV
Chen et al. [40] *(2013)*	LRN	MF	IMP	WWW
Das et al. [42] *(2007)*	LRN	CLST & KNN	IMP	NEWS
Devooght et al. [46] *(2015)*	LRN	MF	EXP	EC, MOV
Diaz-Aviles et al. [49] *(2012)*	LRN	MF	IMP	WWW
Diaz-Aviles et al. [48] *(2012)*	LRN	MF	IMP	WWW
Frigó et al. [55] *(2017)*	EVL	KNN, MF	IMP	EC, MOV, MUS, WWW
George & Merugu [62] *(2005)*	LRN	CLST & KNN	EXP	EC, MOV
He et al. [68] *(2016)*	LRN	MF	IMP	EC, POI
Huang et al. [71] *(2017)*	LRN	MF	EXP	MOV
Huang et al. [72] *(2016)*	LRN	MF	IMP	VID
Huang et al. [73] *(2015)*	LRN	KNN	IMP	EC
Jugovac et al. [76] *(2018)*	LRN	KNN, MF, NN	IMP	NEWS
Khoshneshin & Street [78] *(2010)*	LRN	CLST & KNN	EXP	MOV
Kitazawa [80] *(2016)*	LRN	MF	IMP	MOV
Koenigstein et al. [83] *(2012)*	RTR	MF	EXP	MOV, MUS
Koenigstein & Koren [82] *(2013)*	RTR	KNN, MF	IMP	EC, MOV, MUS
Li et al. [91] *(2017)*	RTR	MF	EXP	MOV, MUS, POI
Liu et al. [95] *(2010)*	LRN	KNN	EXP, IMP	MOV
Liu & Aberer [96] *(2014)*	MEM	MF	IMP	EC, WWW
Lommatzsch & Albayrak [97] *(2015)*	LRN	KNN & ENS	IMP	NEWS
Matuszyk et al. [103] *(2015)*	MEM, EVL	MF	EXP, IMP	EC, MOV, MUS
Matuszyk & Spiliopoulou [102] *(2017)*	LRN, EVL	MF	EXP	EC, MOV
Miranda & Jorge [106] *(2009)*	LRN	KNN	EXP	WWW
Nasraoui et al. [107] *(2007)*	MEM	KNN	EXP	WWW
Nathanson et al. [108] *(2007)*	MEM	CLST	EXP	WWW
Neyshabur & Srebro [110] *(2015)*	RTR	-	-	Others
Pálovics et al. [111] *(2014)*	LRN, EVL	MF	IMP	MUS
Papagelis et al. [112] *(2005)*	LRN	KNN	EXP	Others
Rendle & Schmidt-Thieme [117] *(2008)*	LRN	MF	EXP	MOV
Sarwar et al. [120] *(2002)*	LRN	MF	EXP	MOV
Shrivastava & Li [122] *(2014)*	RTR	-	-	Others

(Continued)

Table 2. Continued

Paper *(Year)*	**Focus**	**Techniques**	**Types of Feedback**	**Domains**
Siddiqui et al. [123] *(2014)*	MEM, EVL	CLST & KNN	EXP	MOV
Song et al. [124] *(2019)*	LRN	NN	EXP	MOV
Song et al. [125] *(2015)*	LRN	MF	EXP	MOV
Subbian et al. [126] *(2016)*	MEM	KNN	EXP	EC
Takács et al. [127] *(2008)*	LRN	MF	EXP	MOV
Takács et al. [128] *(2009)*	LRN	MF	EXP	MOV
Teflioudi et al. [131] *(2015)*	RTR	MF	EXP	MOV, MUS
Vinagre et al. [139] *(2015)*	LRN	MF	IMP	MOV, MUS
Vinagre et al. [137] *(2014)*	EVL	KNN, MF	IMP	MOV, MUS
Vinagre et al. [138] *(2014)*	LRN, EVL	MF	IMP	MOV, MUS
Vinagre & Jorge [135] *(2012)*	MEM	KNN	IMP	MOV, MUS
Vinagre et al. [141] *(2018)*	LRN	MF & ENS	IMP	MOV, MUS
Vinagre et al. [142] *(2018)*	LRN	MF & ENS	IMP	MOV, MUS
Wang et al. [144] *(2018)*	LRN	NN	EXP	MOV
Wang et al. [143] *(2013)*	LRN	MF	EXP	MOV
Wang et al. [145] *(2018)*	LRN	MF	EXP, IMP	EC, MOV
Yu et al. [148] *(2016)*	LRN	MF	EXP	MOV, WWW
Zhang et al. [152] *(2014)*	LRN	MF	EXP	Others

Focus: CHG: Change detection module, EVL: Evaluation, LRN: Learning module, MEM: Memory module, RTR: Retrieval module; **Techniques:** CLST: Clustering, ENS: Ensembles, KNN: Neighborhood-based approaches, MF: Matrix factorization or other model-based approaches, NN: Neural networks, SPM: Stochastic process models; **Types of feedback:** EXP: Explicit, IMP: Implicit; **Domains:** EC: E-commerce, MOV: Movies, MUS: Music, NEWS: News, VID: Video, WWW: Web navigation or tag recommendations.

the performance of the approach, measured using the root mean squared error, the precision, and the recall [66], largely depends on the intensity of changes in concepts, e.g., changes in user preferences, in the domain considered and on the size of the sliding window.

Another approach is proposed by Nathanson et al. [108] in the context of Eigentaste 5.0, which is part of Jester, the online joke recommender. Items are clustered offline in the learning phase, and they maintain for each user the average of ratings given to items per cluster. This average rating is only computed based on the recent ratings given to items from the same cluster. Recommended items are the top-predicted items from the item cluster where the average rating is the highest. The relevance score of an item itself for a user is computed based on similar users. As more ratings are collected, the base cluster may change and we thus obtain dynamic recommendations adapted to an online setting. The authors report that the error in rating predictions of Eigentaste 5.0 is significantly lower than the one of Eigentaste 2.0, which does not consider adapting the base cluster of users in a dynamic way.

Apart from abrupt forgetting, a less extreme form of forgetting, *gradual forgetting*, can be performed by weighting observations or previously learned concepts based on their age. Fading factors have thus been used to gradually forget old information. Vinagre and Jorge [135] consider an incremental user-based neighborhood approach, where similarities and recommendations are updated for each received observation and proposed to apply fading factors. In particular, the authors multiply user similarities by a positive scalar $\alpha < 1$ before updating them with information received from new observations. The value of α controls the rate at which forgetting is performed. The authors test several values of α and consider evaluating their approach in different settings, i.e., abrupt and gradual drifts. Results show that in the scope of their experiments, the proposed

technique is less effective in the case of subtle changes than in the case of sudden changes. Such techniques can also be largely affected by the sparsity of the dataset considered.

5.1.2 Forgetting in Model-Based Approaches. Model-based approaches for recommendation [85] exploit user interactions to learn a predictive model representing user preferences and item descriptions. Among these approaches, **Matrix Factorization (MF)** became one of the most popular techniques by ensuring good scalability and predictive accuracy [86]. MF techniques model users and items by representing them in a common space of low dimensionality, the space of latent factors, where the affinity between a user and an item is determined by the inner product of the embedded vectors. Training an MF model consists of learning those representations.

Formally, each user u is associated with a vector $\mathbf{p}_u \in \mathbb{R}^K$ and each item i with a vector $\mathbf{q}_i \in \mathbb{R}^K$. The hypothesis of MF methods is that the value of a rating r_{ui} can be captured by the inner product between $\mathbf{p}_u^\top$ and $\mathbf{q}_i$:

$$\hat{r}_{ui} = \mathbf{p}_u^\top \mathbf{q}_i. \tag{2}$$

The goal is to determine the vectors $\mathbf{p}_u$ for each user u and $\mathbf{q}_i$ for each item i. MF methods then approximate the rating matrix $\mathbf{R}$ with the product of two low-rank dense matrices $\mathbf{P} \in \mathbb{R}^{K \times n}$ for user latent factors and $\mathbf{Q} \in \mathbb{R}^{K \times m}$ for item latent factors:

$$\mathbf{R} \approx \mathbf{P}^\top \mathbf{Q}, \tag{3}$$

where $K \ll \min(n, m)$.

Forgetting has been applied to manage the data used to train MF models. Matuszyk et al. [103] introduce two sets of forgetting strategies within the scope of MF: *rating-based* forgetting and *latent factor–based* forgetting. Rating-based strategies discard past ratings for each user using either sliding windows, e.g., fixed size or covering a fixed time period, or sensitivity analysis, e.g., forget ratings that imply abnormal changes in the user profile. However, latent factor–based strategies readjust the latent factors of MF to reduce the effect of old observations. These strategies include forgetting unpopular or popular items and fading user factors. Experiments on several datasets show that in the presence of explicit feedback, the latent factor–based forgetting strategy that penalizes unpopular items is particularly successful. In the presence of implicit feedback, the latent factor–based forgetting strategy that multiplies latent factors by a constant depending on the volatility of factors leads to improvements, in terms of recall [66], over a specific **Incremental Matrix Factorization (IMF)** approach described by Takács et al. [128]. The original work does not provide any additional information related to this point. Thus, it is not clear how this strategy affects the diversity of recommendations and if this result is due to the fact that the users considered are biased toward popular items.

5.1.3 Discussion. Applying forgetting strategies presents the great advantage of being able to leverage the various traditional recommendation algorithms proposed in the literature. However, most SBRS that only contribute to the memory module face one main limitation: they assume we have an *a priori* understanding concerning the way user preferences and item perceptions are drifting, and that these concepts are changing at the same rate. In fact, they postulate that old observations are no longer relevant and they necessitate the definition of a number of parameters, which, for example, are related to the shape of a fading factor or to the size of a sliding window. Given that this general assumption does not always hold and that this setting can be impractical, the performance of these methods remains limited.

One interesting direction could consider learning or automatically adjusting these parameters. Some techniques from data stream mining could be exploited to this end. As an example, Bifet and Gavalda [29] propose an algorithm that maintains a sliding window of which size is variable and changes are based on the distribution of received observations. The size of the window grows

when no changes are detected and shrinks otherwise. To the best of our knowledge, this idea of automatically tuning parameters has not been exploited for SBRS.

5.1.4 Other Approaches. Beyond forgetting techniques, other approaches were proposed to manage received observations in the scope of SBRS. Liu and Aberer [96] rely on the idea of modeling separately long-term and short-term preferences by separately processing old information and new information. They design a framework consisting of two components: the online component that is continuously updated as new observations are received and the offline component that is updated less frequently. Recommendation lists are generated by each component and merged to get the final list recommended to the user. It is worth mentioning that all received observations are eventually stored and used to train the offline component, which makes this approach not completely adapted to a streaming setting. However, Subbian et al. [126] developed a probabilistic neighborhood-based algorithm based on a min-hash technique. Similarities between users are approximately computed using min-hash indexes that are implemented using hash functions. These hash functions are stored in memory instead of storing the full history of user interactions. We note that both of these approaches give another perspective for managing observations in a streaming environment.

5.2 Learning Module

The learning module defines how recommendation models are learned and maintained up to date. Learning recommendation models in SBRS is usually based on *incremental learning*. In incremental learning, observations are processed one after the other and existing models are updated after each observation is received and may also access older observations stored in the memory or statistics related to past observations. In the more restrictive setting of *online learning*, data is processed only once, in a sequential fashion [94], without accessing information related to previous observations: only the most recent observation is used to update the current model. The *streaming* setting adds limitations on the memory and time resources required for processing data, given its high rate.

In the general case of incremental learning, concepts learned from recent data tend to erase those previously learned, as learning is carried on. Continuously updating models over time is a way to *passively* adapt to changes, by instantly analyzing new data. In the context of SBRS, traditional CF methods, i.e., memory-based and model-based approaches, have been adapted to the incremental setting.

5.2.1 Incremental Memory-Based Approaches. Applying memory-based algorithms in a streaming setting faces two major issues. First, similarity computation is performed in a costly offline phase since similarities between all user pairs or item pairs have to be evaluated. Second, the whole history of user interactions cannot be stored in memory and used entirely for recommendation. Several solutions were proposed addressing one or both of these limitations.

Incremental neighborhood-based methods were first proposed by Papagelis et al. [112]. The authors present an incremental user-based approach handling explicit feedback in the form of ratings. User similarities are stored and incrementally updated for each received rating. Nearest neighbors and predicted ratings are then computed at the time of recommendation. Similarities, denoted by sim_1, are evaluated using Pearson correlation that is split in the following way:

$$sim_1(u,v,t) = \frac{B^t}{\sqrt{C^t}\sqrt{D^t}}; B^t = \sum_{x \in \mathcal{I}_{uv}^t} (r_{ux} - \overline{r}_u^t)(r_{vx} - \overline{r}_v^t),$$
$$C^t = \sum_{x \in \mathcal{I}_{uv}^t} (r_{ux} - \overline{r}_u^t)^2, D^t = \sum_{x \in \mathcal{I}_{uv}^t} (r_{vx} - \overline{r}_v^t)^2, \quad (4)$$

where t denotes the current time at which the computation is performed.

Elements B, C, and D are stored and incrementally updated in order for new similarities to be computed in a fast manner. In fact, whenever a user u provides a new rating at time $t+1$, similarities between u and all of the other users are updated. This new provided rating is actually affecting the value of average rating, $\bar{r}_u$, in addition to the set $\mathcal{I}_{uv}$. The new similarity, $sim_1(u, v, t+1)$ is computed as follows:

$$sim_1(u, v, t+1) = \frac{B^{t+1}}{\sqrt{C^{t+1}}\sqrt{D^{t+1}}} = \frac{B^t + \epsilon_B}{\sqrt{C^t + \epsilon_C}\sqrt{D^t + \epsilon_D}}. \tag{5}$$

ϵ_B, ϵ_C, and ϵ_D are increments that are defined differently if u is updating the value of an existing rating or submitting a new one, and if user v already provided a rating to that particular item or not [112]. The main idea is that the similarity is incrementally updated by using the previously computed values B^t, C^t, and D^t, and updating them based on the received rating. Compared to classical CF, the authors prove their approach to be highly scalable.

Incremental user–based and item-based neighborhood approaches handling implicit feedback and using the cosine similarity were then presented by Miranda Jorge [106]. The formulation of cosine similarity in an implicit feedback setting is based on user and item occurrence and co-occurrence counts. Two users co-occur for every item they both interact with, and two items co-occur for each user who interacts with both of them. In this setting, when computing the similarities between users u and v and using Equation (4), B represents the number of items that both u and v interacted with, and C and D represent the number of items that each one of u and v interacted with, respectively. One counter is stored for each of these elements. For each received observation, maintaining similarities up to date boils down to incrementing these counters and recomputing the similarities' values. Similarly to Papagelis et al. [112], nearest neighbors and scores are computed at the moment of recommendation.

In an attempt to adapt to user and item changes in the scope of a neighborhood-based method, Liu et al. [95] introduce the online evolutionary CF framework. It relies on an incremental version of the item-based neighborhood method and proposes instance weighting techniques to reduce the weight of old observations when computing similarities and scores. In fact, the authors propose to assign a weight f_{ui}^{α} for each rating r_{ui} depending on its age, defined as follows: $f_{ui}^{\alpha}(t) = e^{-\alpha(t-t_{ui})}$, where α is a parameter controlling the decaying rate. This weight intervenes in computing similarities and prediction scores. Similarities between items i and j are defined based on the Pearson correlation, similarly to Equation (4), as follows:

$$sim_2(i, j, t) = \frac{P_{ij}^t}{\sqrt{Q_i^t}\sqrt{Q_j^t}}; P_{ij}^t = \sum_{x \in \mathcal{U}_{ij}^t} (f_{ui}^{\alpha}(t) \cdot r_{ui})(f_{uj}^{\alpha}(t) \cdot r_{uj}), Q_i^t = \sum_{x \in \mathcal{U}_i^t} (f_{ui}^{\alpha}(t) \cdot r_{ui})^2. \tag{6}$$

Similarly to Papagelis et al. [112], the authors derive incremental updates for similarity computation and score prediction. As an example, P_{ij}^{t+1} is expressed based on P_{ij}^t as follows: $P_{ij}^{t+1} = e^{-2\alpha} \cdot P_{ij}^t + \Delta P_{ij}$, where ΔP_{ij} is a term computed based on the newly received observation. The authors report that the use of temporal relevance weighting leads to more significant improvements for the task of top-N recommendation than for the task of rating prediction. Improvements are measured using the root mean square error and the mean average precision [66].

Other approaches for incremental CF built on co-clustering methods for recommendation [62, 78]. In its simplest form, George and Merugu [62] propose parallel and incremental versions of the co-clustering algorithm that is used for CF by simultaneously clustering users and items. Predicted ratings are estimated based on the average rating of co-clusters and users' and items' biases. Given an initial co-clustering of users and items computed based on the whole available data, the newly

received ratings are directly used to update the clusters' statistics, including the average ratings, thus affecting the rating prediction. If a new (unclassified) user is received, it is temporarily associated to a global cluster before being attributed to another one during the next execution of the co-clustering algorithm. Whereas George and Merugu [62] rely on the Bregman co-clustering [22], Khoshneshin and Street [78] introduce a number of revisions to improve its performance.

However, efficient frameworks, implementations, and architectures of SBRS were presented based on neighborhood-based approaches. *StreamRec* [38] is based on a stream processing system and implements a scalable item-based CF approach while handling explicit ratings for the recommendation of movies and news. The framework models an RS as a complex event processing application using incremental streaming operators and is implemented using the Microsoft CEP [16]. The authors describe the architecture and the practical implementation of the RS, the type of streaming events handled, and the operations executed for building the model and generating recommendations, while focusing on the database dimension of the streaming system.

TencentRec [73] is another framework also relying on an item-based CF approach but dealing with implicit feedback and designed to serve recommendations for a wide range of applications having different requirements. It is built on Storm with a data access and a data storage components. The incremental CF approach implemented is similar to the one proposed by Miranda and Jorge [106]. However, additional details related to its practical and efficient implementation are provided. The authors also propose to perform pruning in real time to reduce the computation cost, by discarding items that are most likely not part of the set of candidates for the most similar items.

5.2.2 IMF Approaches. Due to the great success of MF and model-based approaches in the context of recommendation [86], several efforts have been made to adapt them to the incremental and online settings. These efforts historically followed the evolution and advances of MF approaches. Whereas some of them focused on integrating newly observed users and items into an existing model, the others managed to update parts of learned models based on current observations.

Singular Value Decomposition. Early efforts on IMF for RS were introduced by Sarwar et al. [120] and were based on **Singular Value Decomposition (SVD)**, which is a well-known technique used for matrix decomposition. In fact, SVD is closely related to the MF problem [119] and decomposes the matrix $\mathbf{R}$ as follows:

$$\mathbf{R} = \mathbf{P}^\top \Sigma \mathbf{Q} = \sum_{s=1}^{d} \sigma_s \mathbf{p}_s^\top \mathbf{q}_s, \tag{7}$$

where $\mathbf{P} \in \mathbb{R}^{d \times n}$ and $\mathbf{Q} \in \mathbb{R}^{d \times m}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{d \times d}$ is a diagonal matrix containing the d singular values on its diagonal. $\mathbf{R}$ can then be approximated by only retaining the largest K singular values from Σ, therefore obtaining the matrices $\mathbf{P}_K$, Σ_K, and $\mathbf{Q}_K$.

In the work of Sarwar et al. [120], incremental updates of SVD are made through the *fold-in* projection technique: latent vectors corresponding to new users or new items are computed based on the current decomposition and are then appended to the existing matrices [27]. To fold a new user represented by its vector of ratings $\mathbf{x}$ into an existing model, we compute its projection $\hat{\mathbf{x}}$, to be added to $\mathbf{P}_K$, as follows:

$$\hat{\mathbf{x}} = \mathbf{x}^\top \mathbf{Q}_K^\top \Sigma_K^{-1}. \tag{8}$$

Along this line, Brand [31] defines incremental operations allowing the addition, update, and removal of data already incorporated into the SVD model. This is made possible through the basic algebraic properties of SVD. Although these techniques lead to high efficiency, SVD suffers from

several shortcomings, such as the need to have an initial matrix with no missing elements, resulting in a limited performance on sparse datasets [88].

Point-wise approaches. Due to the limitations of SVD for CF, the MF framework later adopted the idea of learning user and item latent matrices by minimizing an objective function, which is the squared error between observed and predicted ratings over known ratings [86]:

$$\mathcal{L}(\mathbf{P}, \mathbf{Q}) = \sum_{(u,i)\in\mathcal{D}} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda\Omega(\mathbf{P}) + \lambda\Omega(\mathbf{Q}). \tag{9}$$

$\Omega(\mathbf{P})$ and $\Omega(\mathbf{Q})$ are the regularization terms and λ is the parameter controlling the relative importance of least square fitting and regularization.

One way to optimize this objective function is to rely on **Stochastic Gradient Descent (SGD)**, iteratively updating user and item latent factors in the direction that reduces the objective function and using its gradient. SGD offers a natural way of performing incremental updates of parameters using a new received observation by applying one step of the algorithm and has thus been extensively used in this context.

With the rise of the MF framework for the problem of rating prediction, Rendle and Schmidt-Thieme [117] introduce regularized kernel MF as a generalization of regularized MF. It provides a flexible method to derive new MF methods by transforming the product of the factor matrices, $\mathbf{P}$ and $\mathbf{Q}$, using kernel functions. Within this context, an online algorithm was proposed to allow the incremental addition of new users and items to the existing MF model. Concretely, the algorithm retrains the whole feature vector corresponding to the new user or item and keeps the other feature vectors fixed. For each new received rating r_{ui}, the vectors $\mathbf{p}_u$ and $\mathbf{q}_i$ are retrained using the stored observations. Along this line, Takács et al. [127, 128] propose *BRISMF*, standing for Biased Regularized Incremental Simultaneous MF, supporting incremental updates of latent factors given an initially trained model and integrating user and item biases expected to model user and item tendencies with respect to ratings. For each received rating, features of the corresponding user are retrained considering all of the ratings the user previously made. However, and in comparison with Rendle and Schmidt-Thieme [117], item features are kept fixed to avoid a larger retraining process and considering that they are more stable than user features.

Whereas Rendle and Schmidt-Thieme [117] and Takács et al. [128] rely on SGD to learn latent factors, other IMF approaches exploit another method, **Alternating Least Squares (ALS)**, to benefit from its advantages. The idea is to repeatedly iterate over the parameters, fix some of them, and solve the optimization problem. In the scope of MF, the item latent matrix is iteratively fixed and the user latent matrix optimized, and vice versa. In the context of incremental learning for MF, Yu et al. [148] propose one-sided least squares for updating one side, i.e, user side or item side, of an existing MF model. It is applied when a new user or a new item is encountered and is also proposed to update both sides of the model when required. To update a user vector or an item vector, we follow the classical procedure of ALS to learn the features and fix all other parameters. We note that the complexity of the solution offered by ALS is higher than the one offered by SGD.

Aside from the rating prediction problem in the presence of explicit feedback, IMF methods were proposed for the implicit feedback setting. In particular, Vinagre et al. [138] proposed an IMF framework for the top-N recommendation problem and handling positive-only feedback. A user or an item that is observed for the first time is initially randomly initialized and added to the model. Then, for each received observation r_{ui}, the user and item latent factors contained in $\mathbf{p}_u$ and $\mathbf{q}_i$ are incrementally updated following SGD, i.e., using the gradient of the objective function (Equation (9)) for the corresponding observation. Vinagre et al. [138] allow slight updates of latent factors, i.e., one update operation per latent factor for user u and item i, instead of total retraining like in approaches mentioned previously.

Handling missing observations. One limitation of the approach proposed by Vinagre et al. [138] is related to the problem of dealing with implicit feedback and concerns the lack of negative observations or the way missing observations should be interpreted. In fact, only considering positive feedback for learning the model may result in an accuracy degradation. To deal with this problem, Vinagre et al. [139] extend their earlier work [138] by artificially introducing negative feedback into the stream of observations. For each positive observation causing a model update, negative items are considered for the current user and result in additional updates compared to their earlier work [138]. Instead of only learning from positive observations, the idea is to try to exploit unobserved items for a user and consider some of them as negative items. Negative items are selected based on the recency of item occurrences: the oldest items appearing in the stream of positive observations are considered as negative feedback. Other methods were also introduced to deal with missing observations in an incremental setting. Experiments on several datasets show an improvement of the recall [66] compared to the approach only relying on positive observations [138].

Devooght et al. [46] extend several loss functions, e.g., squared and absolute losses, to take into account an explicit prior on unknown ratings and derived online learning algorithms to update the model. The objective function (Equation (9)) now has an additional term related to the missing ratings, driving the model toward a prior defined for these missing ratings. Online updates of the model are based on the same ideas as those of Vinagre et al. [138]: when a new rating is received, only the corresponding user and item factors are updated by performing a gradient step, assuming that a new received rating only significantly affects the corresponding user and item. Experiments show that the area under the curve [66] increases over time as new ratings are being considered to update the model. Whereas the work of Devooght et al. [46] assumes that missing entries are equally likely to be negative observations, He et al. [68] introduce a weighting strategy for missing observations based on item popularity. The authors also propose a new learning algorithm based on ALS, *eALS* for element-wise ALS, and handling variably weighted missing data. The learning algorithm optimizes the parameters at the element level: at each iteration, we optimize one coordinate of the latent vector while fixing all of the other parameters. A fast learning procedure is enabled by caching some of the used terms at every iteration instead of naively computing them multiple times. The authors derive an incremental strategy supporting online learning: when a new rating is received, the corresponding user and item latent factors are retrained using *eALS* until convergence.

Pairwise approaches. With the evolution of the recommendation problem beyond the matrix completion task, several objective functions were considered for training recommendation models. In particular, pairwise functions approximate the loss by considering the relative ranking of predictions for pairs of items [116]. The assumption made in such settings is that positive items, i.e., items that have been observed for the user, should be ranked higher than negative items, i.e., items that have not been observed. Adopting these pairwise functions appears to be particularly beneficial for the task of top-N recommendation in the presence of implicit feedback.

In the scope of online recommendation, Diaz-Aviles et al. [49] present Stream Ranking MF or RMFX, which relies on a pairwise approach to MF and uses a selective sampling strategy based on active learning ideas to perform incremental model updates. Instead of updating the model for each received observation, thus allowing the model to quickly forget past observations, the approach maintains a reservoir containing a fixed number of observations sampled continuously from the stream of observations. Model updates are then performed by iterating through the reservoir instead of going through the entire stream. At each interaction, we sample one observation from the reservoir, construct a small buffer containing negative items for the considered user, select one of those items based on the gain of information it provides, and update the model by minimizing the pairwise objective function.

Diaz-Aviles et al. [48] rely on the same approach and further investigate additional sampling strategies. In addition to the reservoir sampling, the authors explore retaining the most recent observations per user. The approach is evaluated on a dataset of tweets for the task of hashtag recommendation. Results show that using a buffer of observations or a reservoir for updating the model leads to better performance—measured using the recall [66]—than updating the model for each received observation. The recommendation quality is also boosted when increasing the size of the buffer.

Further advances in IMF. The approach introduced by Huang et al. [71] proposes a framework for the incremental update of MF models once they are initially learned in batch, based on a linear transformation of user and item latent factors. In fact, the transformation of a latent feature matrix, which is expected to happen after new observations are received, is represented as a linear function of its previous value and of the incremental latent feature matrix that results from the factorization of the incremental matrix of newly received observations. Ratings are thus handled batch by batch for updating models. The performance of the approach is evaluated on two datasets in the context of rating prediction and is measured using the root mean square error [66], the cumulative runtime, and the storage of rating samples. The proposed approach show better results compared to two other incremental approaches, including that of Rendle and Schmidt-Thieme [117].

However, and instead of only updating vectors related to the current received observations, Wang et al. [143] explore multi-task learning and proposed to update additional vectors according to a user interaction matrix. This user interaction matrix, inspired by ideas from multi-task learning, encodes the degree of interaction or similarities between users. The authors report that they are able to learn more accurate models in comparison with two other approaches that only update one item and one user for each observation, at the cost of being less efficient as the required running time is much higher than the other approaches.

In contrast with most approaches mentioned earlier that rely on vector retraining, a space retraining model is proposed by Song et al. [125]. The feature space is retrained using, on one hand, auxiliary feature learning and, on the other hand, matrix sketching strategies. The residual error of newly received observations is computed, normalized, and added to the existing feature matrices under the form of auxiliary features, by exploiting online matrix extension. Given that this operation results in continuously increasing the number of dimensions of the feature space, a sketch of lower feature size and retaining most of the information is maintained and user for re-learning the rest of the features. However, this approach is only meant to handle the new user and new item cases, i.e., model users and items that are new to the system. In this particular context, and when measuring the root mean square error [66] for rating prediction, experiments show that the proposed approach performs better than SVD and the approach of Rendle and Schmidt-Thieme [117] on two datasets.

5.2.3 Beyond MF Approaches. Relatively few efforts have been made to adapt methods other than MF to the online setting of recommendation. We mention some of them in the following.

Tensor factorization. Zhang et al. [152] propose an incremental version of tensor factorization for the problem of web service recommendation. Tensor factorization has been one of the most successful methods for integrating contextual information into the recommendation model [77]. It can be seen as a generalization of MF in which we factorize a multidimensional cube instead of a 2-dimensional matrix. Zhang et al. [152] rely on a 3-order tensor to represent the user, item, and time dimensions and derive an incremental version of tensor factorization based on the incremental SVD [120].

Factorization machines. Another way of integrating additional dimensions into recommendation models is based on factorization machines [114]. Kitazawa [80] introduce an incremental version

adapted to the online setting. The author proposes to update the parameters of factorization machines using SGD, for each received observation. Regularization parameters are also incrementally updated, as it is proposed in the original work applying factorization machines for RS [115]. When dealing with a dataset of movie ratings, the features considered in this work are related to user demographics, the movie genre, the time at which the rating was provided, and the time elapsed since the last rating was provided by the user. We note that this is one of the few works considering additional features in the context of incremental recommendation approaches, but the evaluation is only demonstrated on one real dataset. In addition, the approach is only compared to incremental MF that does not consider the additional features [138] and to other static approaches. Nevertheless, the work underlines the improvements obtained in terms of recall [66] when considering online models instead of static ones.

Online regression. Agarwal et al. [12] rely on online regression and introduce an online bilinear model to learn user or item factors. The framework is specifically designed for applications where new users and items frequently appear. The authors formulate the problem as a regression task and the framework combines offline training and online updates. Linear projections are also computed for dimensionality reduction, which leads to a fast and scalable procedure.

Ensemble learning. Ensemble methods are known to enhance the accuracy of algorithms by combining predictions or results from several learners or models, using different techniques. In particular, bagging [32] and boosting [54] have been used in the scope of online recommendation.

In the context of bagging, we train several models, each one on a bootstrap sample of the dataset. Vinagre et al. [141] propose to rely on bagging and train a certain number of IMF models that are used later for generating recommendations. Practically, each received observation is used to train a subset of these models, selected in such a way that guarantees that each model is being trained on a bootstrap sample of the dataset. Results show that bagging can improve the recommendation quality, in terms of recall [66], compared to a setting where only one IMF model is used, especially when the number of recommended items is greater than 10. Therefore, this technique may be less beneficial in applications where only a couple of items can be recommended.

The same authors later looked into boosting and its application for online recommendation [142]. The boosting ensemble method proposes to handle several learners where each one attempts to rectify the deficiencies of the previous one. Vinagre et al. [142] propose two algorithms for boosting incremental RS based on stochastic gradient boosting. Both algorithms maintain M learners over which we iterate in the learning step: the first learner tries to learn the value of the target variable and passes the residual to the next learner in order for it to learn this value, repeatedly until the end of the iteration. Results show that some gain—also in terms of recall [66]—can be obtained compared to only relying on one IMF model, especially when the number of learners is low.

Semi-supervised learning. Matuszyk and Spiliopoulou [101, 102] present a semi-supervised framework for SBRS. The idea is to address the problem of sparsity in RS, where only a limited number of items are being rated by each user and resulting in a large amount of unlabeled information. Several recommendation models are trained in batch, and once the streaming mode is started, supervised and unsupervised learning are performed in a sequential manner. During the supervised learning phase, predictions are performed and the feedback of users is used to update the models: the functioning is thus similar to other SBRS that have been reviewed. During the unsupervised learning phase that takes place periodically in the stream, a number of unlabeled user-item pairs are selected and predictions are performed by each learner. Based on a reliability measure that assesses the prediction quality of each learner, reliable learners provide the label that is used by unreliable learners to incrementally train their models. Experiments with and without

the semi-supervised learning setting measuring the recall [66] show the potential of using the proposed framework.

Neural networks. Although deep learning techniques are becoming popular for designing RS [151], here we only consider the application of those techniques for SBRS, which is still not too common. In fact, a recent survey on deep learning–based RS [151] states that the future work around neural RS should focus on the problem of incremental learning for streaming and non-stationary data. Following this direction, Wang et al. [144] recently proposed Neural Memory Recommender Networks (NMRN) to address the problem of recommendation in streaming environments. The recommender model relies on external memories that are meant to record long-term and short-term user preferences. To this end, the authors leverage key-value memory networks [105]. They also rely on Generative Adversarial Nets (GANs) [65] to address the limitations of negative sampling approaches and improve parameter inference. The idea is to select informative negative items while also adapting to the changes in similarities between users and items. Finally, it is also worth mentioning the work of Souza et al. [56] that is based on recurrent neural networks and presents a contextual hybrid approach that leverages side information, e.g., popularity and recency, for news recommendation. Although this approach is not specifically designed for a streaming setting, the model is trained using mini-batches, thus supporting online learning, and is evaluated in a setting that is similar to a streaming setting.

Learning from few observations in cold-start conditions. In an environment where new users are regularly arriving to use the service for the first time, the question of generating recommendations and learning a user model when only a few information is available can be challenging and is known as the *cold-start* problem [81]. Efficient RS are required to deliver relevant recommendations under such conditions. Most of the works surveyed here do not specifically focus on the cold-start problem in the context of SBRS, and no special treatment is applied to new users. For example, in an IMF model [138], new users are assigned a random model which is then incrementally updated as more observations are collected. Modeling users and generating recommendations when only a few information is available is also a problem that occurs in session-aware RS [113], when dealing with anonymous users who are only identified for the duration of one session. Online solutions applied in the context of session-aware RS can therefore inspire the design of SBRS for handling new users and addressing the cold-start problem.

Examples of such solutions include the class of incremental algorithms proposed in the work of Garcin et al. [60] based on context trees. Context trees partition the space of contexts in a hierarchy, where each context is defined as a sequence or distribution of topics and associated with a prediction model. As more information about the user is gathered, it is possible to define the context more precisely that is then used to generate context-dependent recommendations by combining the results of different prediction models. Parameters used to perform this combination are sequentially updated based on Bayes' theorem.

5.2.4 User and Item Dynamics. Beyond efficiently updating models and integrating new users and items, online recommendation should account for the evolution of users and items over time. Recent efforts have tried to address this problem in several ways. In contrast to previously presented approaches that continuously update models at a constant speed for each received observation, these efforts explicitly take into account the dynamics of users and items by modeling their evolution.

First, Chang et al. [39] present a framework that handles streams via a continuous-time random process. Markov processes are meant to model each time-varying user and item factors in an attempt to capture the dynamics occurring in such settings. The authors propose a variational Bayesian method for efficient inference in online settings. Then, another framework is introduced

by Wang et al. [145] and is based on Bayesian Personalized Ranking (BPR) [116]. It handles users' interest drifts, new users and items, and the system overload occurring in a streaming setting. It also relies on samples stored in a reservoir to represent users' long-term preferences instead of only learning from recent observations.

Song et al. [124] explore Deep Bayesian Learning and proposed a Coupled Variational Recurrent CF (CVRCF) framework to address the streaming recommendation problem. The authors claim to combine the advantages of deep neural networks for RS on one side and probabilistic models for modeling uncertainties on the other side. User preferences and item popularities are modeled as the combination of a stationary factor, encoding the long-term aspects that vary slowly, and a dynamic factor, encoding the short-term aspects that evolve quickly. A variational inference algorithm is then proposed based on variational recurrent neural networks to ensure an efficient optimization. The experiments conducted explore, in particular, the drifting patterns learned and the stability of the framework.

5.3 Change Detection Module

The change detection module implements techniques for the active detection of drifts. In fact, continuously updating recommendation models over time is a way to *passively* adapt to changes. This is actually happening when performing incremental learning. Passive approaches face the limitation of being slow in reaction to changes: learning is done at a constant speed independently from the changes occurring in the environment. Therefore, *active* approaches have been proposed where drifts are explicitly detected, triggering a learning mechanism to adapt the models [52]. Drift detection is handled by the *change detection* module, and once a drift is reported, several strategies with regard to learning and adaptation can be performed.

Concretely, concept drifts can take different forms [59]. The concept drift can happen either *abruptly* resulting in a sudden change, *incrementally* consisting of the existence of intermediate states between the old concept and the new one, or *gradually* where the change is not abrupt but we keep returning to the previous concept for some period of time. Although new concepts may be introduced at each change, previously observed concepts may recur, resulting in *recurrent* drifts.

These ideas have rarely been exploited in the context of recommendation and SBRS. A limited number of efforts, mentioned in the following, have been made in this direction.

Al-Ghossein et al. [14] introduce dynamic local models that learn from streams of user interactions and adapt to user drifts or changes in user interests. Initially, users are distributed into several clusters and a neighborhood-based local model is learned for each cluster and used to generate recommendations. The idea is then to automatically detect the drift of preferences causing the user to follow a behavior that is similar to users grouped in a different cluster and to update the corresponding models.

In another work, Al-Ghossein et al. [15] propose a hybrid RS that uses textual content to model new items received in real time in addition to user interactions to model their preferences. The approach proposed detects item drifts or changes occurring in item descriptions, which is done by using topic modeling of items [30] and a drift detection technique [29], widely used in the data stream mining community. Experiments show the importance of using a change detection technique to maintain an updated model that is able to generate relevant recommendations.

5.4 Retrieval Module

The retrieval module is responsible for generating recommendations for a user, which usually includes computing the relevance score of each item, ordering them, and selecting the N items scoring the highest. Without loss of generality, the problem of recommendation can be divided into two tasks: learning a utility function and estimating the ratings or relevance scores of items for a

target user. In an online context, the utility function is constantly updated and scores have to be computed on the fly. In an attempt to be compliant with a streaming setting, specific methods have to be developed and adopted to ensure an efficient retrieval of recommendations. In the following, we only review the main methods proposed, given that they were not meant to work in a fully online setting and were not evaluated to this end.

The most common form of the problem of efficiently retrieving recommendations was framed as the problem of **Maximum Inner Product Search (MIPS)**. Formally, given a query $\mathbf{p} \in \mathbb{R}^K$ corresponding to the user and given a large collection of m items $\mathcal{I} = \{\mathbf{q} \in \mathbb{R}^K : 1, \ldots, m\}$, the goal in MIPS is to identify the set of items having the largest values of the inner product with $\mathbf{p}$. The naive solution to solve MIPS, which constitutes in a linear search, requires $O(mK)$ operations to compute the inner products and $O(m \log m)$ to order the m items. As this solution requires expensive computations and is not scalable, a number of efficient alternatives have been proposed, leading to exact or approximate solutions. Here we only consider those that do not affect the learning procedure and that do not depend on it.

Hashing methods. A first set of solutions applies hashing techniques to reduce the cost of computations, leading to approximate solutions and introducing a trade-off between the quality of recommendations and the efficiency of computations. The main idea is to divide the space into several partitions where elements, i.e., users and items, from the same partition get the same hash-code. Given a specific user and its corresponding bucket, we only consider the items from the same bucket as candidates for recommendation. By using this technique, we reduce the number of operations given that the space of items is pruned. If designed appropriately, it is thus expected to significantly reduce the runtime of the process generating recommendations.

Along that line, Shrivastava and Li [122] and Neyshabur and Srebro [110] considered using the **Locality Sensitive Hashing (LSH)** technique [63], which has been useful to solve the problem of nearest neighbor search. Whereas Shrivastava and Li [122] rely on an asymmetric hashing that was considered essential to address MIPS, Neyshabur and Srebro [110] later show the existence of a symmetric LSH that can be used, improving the empirical performance.

Learning to hash techniques were also used in this context [153, 155]. The main idea is to hash user and item vectors into binary codes and then to recommend items which codes are within a small distance from those of the user. Zhou and Zha [155] propose to learn these codes in a way that the Hamming distance between codes try to capture the inner product between user and item vectors. Building on this work, Zhang et al. [153] define the binary codes by focusing on maintaining the ranking of inner products between pairs of users and items. It is worth mentioning that these approaches are definitely not appropriate for the online recommendation setting: it is not possible to dynamically update the binary hash codes of users and items once defined.

Tree-based methods. A second set of solutions relied on tree-based approaches. Koenigstein et al. [83] propose to use metric trees, i.e., binary space-partitioning trees, to index item vectors, and use a branch and bound algorithm to obtain an exact solution. The technique is applied for neighborhood-based methods [82] and MF methods [83]. An approximate faster solution relying on a clustering of users is also proposed [83]. Recommendations are computed for the cluster centers and presented as an approximate result to all users belonging to the corresponding cluster. Another approximate approach was proposed by Bachrach et al. [20]. It consisted of, first, transforming the item vectors by applying an SVD based transformation, indexing the first few elements of the vectors by a PCA-Tree, and then using a neighborhood search approach.

Sequential scan. The last set of solutions are based on a sequential scan of candidates. An exact high-performing solution is proposed by Teflioudi et al. [130, 131] by introducing the LEMP framework. The idea adopted is to filter item vectors that are not qualified to be potential solutions,

using the Cauchy-Schwarz inequality. The items are grouped into clusters of similar lengths, and we search in each one of them for the smallest cosine similarity. Partial inner products are also computed over a limited number of dimensions and used as upper bounds to incrementally prune the search space. FEXIPRO [91] is another approach proposed to address the same problem. FEXIPRO performs an SVD transformation to the item matrix, in order for the first few dimensions to hold a large part of the inner product. This is eventually improving the upper bounds and leading to a more efficient pruning procedure.

As mentioned before, there has been no focus in previous work on retrieving recommendations in a fully online setting, i.e., in a setting where user models and item models are dynamically changing. Nevertheless, this module remains an essential part of SBRS.

5.5 Applications and Frameworks

Specific frameworks were developed to address the streaming recommendation problem for a particular application or domain.

News recommendation. SBRS were first developed to address the problem of news recommendation [53]. In fact, Das et al. [42] present one of the first applications to tackle the problem of online recommendation to deliver news recommendations in real time. One has to notice that in the context of news recommendation, the item catalog has the particular characteristic of undergoing changes every few minutes through the insertion and deletion of news articles. Das et al. [42] propose a scalable approach combining different recommendation methods and using parallel computation. They also rely on a time-decaying function for user interactions and on a time-based window to evaluate co-visits of news. In a different study, Lommatzsch and Albayrak [97] analyze the stream of user interactions on several news portals, provided in the context of the ACM RecSys News Challenge 2013.[1] The authors present the characteristics of the stream of interactions and develop and compare a number of approaches optimized for providing news recommendations under the strict time constraints of the streaming setting. These approaches include recommending the most popular articles, recommending recently requested articles, CF-based methods, and ensemble methods. In the same scope, Jugovac et al. [76] introduce *StreamingRec*, an open source framework for evaluating SBRS for the application of news recommendation and for ensuring replicability. The framework is based on a replay evaluation protocol allowing the update of models in real time and ensuring the streaming setting. *StreamingRec* implements a series of baseline algorithms for session-based RS [113]. The authors report the performances of these algorithms on two datasets. A neighborhood-based method taking into account the recency of events [74] outperforms the other considered baselines, including several CF approaches.

Recommendation on social networks. Pálovics et al. [111] consider the problem of online music recommendation in a social media service. They propose an approach based on MF that exploits temporal and social influences between users to improve recommendations. Users who are socially connected are expected to exhibit similar preferences in close periods of time. Also in the context of social media, Chen et al. [40] propose *TeRec*, a service for hashtag recommendation over the tweet stream, helping users to choose hashtags when posting tweets. When the user feedback is received, a reservoir that stores past entries is updated and is then used to update the latent factors of a MF model. The authors proposed a new sampling strategy for filling and updating the reservoir, which is based on the idea of representing users' real-time interests rather than overall preferences. Learning the model is based on sampling negative examples based on active

[1] https://recsys.acm.org/recsys13/nrs/.

learning principles. The authors report improvements over the method proposed in the work of Diaz-Aviles [49] on one dataset of tweets.

Video recommendation. Finally, Huang et al. [72] consider the problem of video recommendation in real time. They develop a scalable IMF algorithm that relies on an adaptive updating strategy. Different types of implicit feedback, e.g., click, watch and comment, are considered, and each one of them is assigned a different confidence level regarding the degree of user interest that it exhibits. These confidence levels affect the value of the learning rate used to update the IMF model. We also mention the work of Amatriain and Basilico [17], exposing a case study of the implementation of an RS at Netflix, from an industry perspective. The authors present a high-level architecture combining offline and online components to perform recommendation and providing technical insights on how to implement such a framework.

6 EVALUATION OF SBRS

This section reviews methodologies and metrics used for evaluating SBRS. Compared to traditional RS, SBRS introduces a setting where the chronological order of observations should be considered and where the models continuously evolve, leading to a different framework for evaluation.

Although evaluation methods for RS include offline and online evaluation [66], this section only focuses on offline evaluation, given that, to the best of our knowledge, online evaluation has not been thoroughly investigated for SBRS. Nevertheless, this remains an important area to explore, as online evaluation provides the strongest evidence regarding the value of an RS. As a starting point, one could consider combining the ideas from the A/B testing methodology [66] and from prequential evaluation, introduced in the following. Online evaluation requires having an online environment serving recommendations to real users and reporting the performance in real time.

It is worth mentioning one of the works along this direction, which is the CLEF News Recommendation Evaluation Lab (NewsREEL) of 2016 [70, 98] that allowed participants to compete in a living lab that queried for news recommendations and evaluated them on real users in real time. The evaluation lab was based on services from plista,[2] a company that provides recommendation services to online news publishers. When a real user visited a news page, the query was sent to a randomly selected team that had to provide a list of recommendations under a tight time constraint and that was evaluated based on the feedback of the real user.

Performing online evaluation could also mitigate the limitations of applying prequential evaluation in an offline setting, as will be mentioned in the following.

6.1 Offline Evaluation of RS

Offline evaluation is among the most popular settings for evaluating RS, as it does not require any interaction with real users and allows replicability and comparison of approaches at low cost [66]. However, offline experiments cannot measure the influence of the RS on the user behavior, which calls for other evaluation methodologies including user studies and online evaluation.

The basic schema of offline evaluation relies on the train-test and cross-validation techniques, widely used in machine learning. The data is usually divided into two distinct parts: the *training set*, used to estimate the utility function, and the *test set*, used to measure the RS performance. There are several ways to split the data, including the random split and the given-n split, i.e., split where the training set is constituted of n interactions for each user, among others [66].

The traditional batch mode of offline evaluation involving hold-out methods is widely used to evaluate RS; however, it faces several limitations in the streaming setting [137]. When data is randomly sampled and divided across the training and the testing tests, the time dimension is

[2] https://www.plista.com/.

ignored and the original ordering of observations is shattered. Recommendation algorithms designed to handle ordered data cannot thus be evaluated with these methods. In addition, shuffling observations may result in illogical operations such as using future interactions to predict past interactions. Learning from shuffled observations would also prevent capturing dynamics occurring within users and items. Whereas batch offline evaluation expects models to be static during the recommendation phase, SBRS continuously update models as recommendations are delivered.

6.2 Offline Evaluation of SBRS

With a new formulation for the recommendation problem came the need to design new appropriate evaluation methodologies. Ideas for evaluating RS in streaming settings appear elsewhere [100, 111, 123, 137, 138]. In an attempt to address the limitations faced by batch offline evaluation, most of them rely on the prequential methodology [58].

6.2.1 Evaluation Methodologies. Prequential evaluation consists of a *test-then-learn* procedure repeated for each received observation [137]. Considering, for example, the implicit feedback setting where received observations are in the form of (u, i), i.e., user u interacted with item i, the prequential evaluation methodology iterates over the following steps for each observation:

(1) Use the current recommendation model to recommend N items for user u;
(2) Evaluate the quality of recommendation given the item i observed;
(3) Update the recommendation model using the observation (u, i) *(optional step)*.

The prequential methodology offers several benefits. Aside from respecting the chronological order of events, it allows the continuous tracking of the RS performance in addition to its evolution over time. Recommendation algorithms can also exploit real-time statistics returned by prequential evaluation for drift detection and model adaptation. However, and in addition to the shortcomings inherited from the fact of being an offline evaluation method, this methodology suffers from one particular limitation. The recommendation list generated by the RS is evaluated against a single item, i.e., item i, failing to acknowledge other potentially interesting recommendations occurring in the list. Therefore, it is not possible to distinguish between cases where the list holds relevant items that will be chosen at a later time and cases where it holds totally irrelevant items. Vinagre et al. [137] report that after performing experiments where the recommendation list is evaluated against all future observations of the target user instead of the current observation only, the overall computed metrics do not improve significantly. Nevertheless, a possible solution consists of adopting hybrid evaluation methods such as in the work of Siddiqui et al. [123]. The authors proposed to use a portion of the ratings in each batch for hold-out evaluation and the rest for prequential evaluation.

Since SBRS usually require an initial phase of training, the operating environment is not exclusively a streaming environment. The evaluation should thus consider the switch between learning an initial model in batch and moving into the streaming setting. To this end, Matuszyk and Spiliopoulou [100] propose the following evaluation protocol. The observations are first sorted chronologically, and the dataset is then split into the three following parts:

(1) The *Batch Train* part, which is obtained by selecting the first 30% of the dataset. The observations contained in this part are used to train the models in batch and to obtain a first initialized version of the models.
(2) The *Batch-Test–Stream Train* part, which is obtained by selecting the next 20% of the dataset. The observations contained in this part are used to test the recommendation models initially learned in (1). They are also used to incrementally update the models,

ensuring that all recorded observations are used for training. This part can also be used as a validation set.

(3) The *Stream Test and Train* part, which is obtained by selecting the last 50% of the dataset. The observations contained in this part are used for prequential evaluation [137].

6.2.2 Evaluation Metrics. Adopting the prequential evaluation has consequences on the metrics' definitions used to assess the quality of recommendations. The main characteristic in the streaming setting is that we are evaluating against a single item: the number of relevant items for the user is always one item, given that we evaluate the recommendation quality every time we receive a new interaction. This is in contrast to batch RS where we are evaluating once for every user against the whole test set: the number of relevant items could thus be greater than 1, given that the test set could contain several interactions per user.

In prequential evaluation, metrics are evaluated and reported for each received observation (u, i). The overall performance of the system is obtained by averaging over all observations, and its evolution in time can also be reported. We present some of these metrics in the following as defined in the scope of prequential evaluation [55], and we denote by $rank_u(i)$ the rank of item i in the recommendation list of size N for user u:

- The *precision* measures the fraction of relevant recommended items. In the streaming setting, it is defined as follows:

$$Precision@N(u,i) = \begin{cases} 0 & \text{if } rank_u(i) > N \\ \frac{1}{N} & \text{otherwise.} \end{cases} \tag{10}$$

- The *recall* measures which proportion of the relevant items are present in the recommendation list. In the streaming setting, it is defined as follows:

$$Recall@N(u,i) = \begin{cases} 0 & \text{if } rank_u(i) > N \\ 1 & \text{otherwise.} \end{cases} \tag{11}$$

- The **Discounted Cumulative Gain (DCG)** is a ranking measure, measuring the quality of ranking in the recommendation list. In the streaming setting, it is defined as follows:

$$DCG@N(u,i) = \begin{cases} 0 & \text{if } rank_u(i) > N \\ \frac{1}{\log_2(rank_u(i)+1)} & \text{otherwise.} \end{cases} \tag{12}$$

- The **Mean Reciprocal Rank (MRR)** computes the reciprocal of the rank of the first relevant item in the full ordered list of items. In the streaming setting, it is defined as follows:

$$MRR@N(u,i) = \begin{cases} 0 & \text{if } rank_u(i) > N \\ \frac{1}{rank_u(i)} & \text{otherwise.} \end{cases} \tag{13}$$

Given that there is only one relevant item when applying the prequential evaluation methodology, no normalization is needed for the DCG metric. The Normalized Discounted Cumulative Gain (NDCG) is therefore not adopted, in opposition to the batch evaluation setting [66].

6.2.3 Statistical Significance. To assess the statistical significance of results in a streaming environment, Vinagre et al. [137] propose to use the McNemar test over a sliding window. The authors used the McNemar test for the recall metric, which takes the values of 0 or 1 in the streaming setting. When comparing two recommendation approaches on a sliding window, the idea is to track the quantity $n_{0,1}$, which is the number of observations for which the recall is 0 for the first algorithm and 1 for the second, and the quantity $n_{1,0}$, which is the number of observations for which

recall is 1 for the first algorithm and 0 for the second. The McNemar test can then be computed for each received observation as follows:

$$M = sign(n_{0,1} - n_{1,0}) \times \frac{(n_{0,1} - n_{1,0})^2}{n_{0,1} + n_{1,0}}. \tag{14}$$

The null hypothesis, i.e., both algorithms are not statistically different, is rejected for $|M| > 6.635$ with a confidence level of 99%, as M follows a χ^2 distribution with a degree of freedom equal to 1. The sign of M then indicates which recommendation approaches perform better.

6.2.4 Datasets. The release of several datasets of user interactions has fueled the research in the field of RS in academic environments. Although multiple datasets related to different domains are currently available, not all of them can be used to evaluate SBRS. In fact, appropriate datasets for evaluating SBRS should verify two main characteristics. First, interactions should be labeled with the corresponding timestamps. Datasets that do not verify this constraint are thus obsolete for the particular problem of SBRS. Second, the timestamp should indicate the point in time where the item has actually been "consumed," e.g., movie watched, song listened to, or product purchased, and not the point in time where the item has been rated. Although such datasets have already been used to evaluate SBRS, it is our opinion that they are not adapted to the task considered here.

Table 3 provides a list of public datasets that have been used to evaluate SBRS in the papers mentioned in this survey, along with their characteristics. The SBRS column indicates if the corresponding dataset is appropriate for use in a streaming context (Y) or not (N), as previously defined. It is important to note that this is rather a difficult classification to carry out: following a first attempt to label the datasets, we only provide this information as a suggestion, acknowledging that it can benefit from further investigation. In fact, some datasets, such as *MovieLens* and *Netflix*, are clearly not ideal for SBRS: they record the user feedback that is provided independently of the time at which the user actually watched the movies. There is even no guarantee that the order in which the movies are rated is the same as the one in which the movies were watched. However, datasets such as *Last.fm* and *Plista* can be used for evaluating SBRS since they record the real-time feedback, collected while users were listening to songs or browsing news. Regarding other datasets, such as *Jester*, this question may not be so straightforward. The classification in Table 3 is provided as a suggestion and relies on the following idea: datasets considered as appropriate for evaluating SBRS are those recording the user feedback that is captured as soon as an item is being considered, i.e., consumed or browsed, thus reporting the current feedback and maintaining the order in which items are considered.

In all cases, we point out the necessity of collecting and releasing new appropriate datasets to further investigate the problem of SBRS.

7 SUMMARY AND FUTURE DIRECTIONS

The problem of recommendation in streaming environments is of high importance in real-world applications. Hence, it is becoming essential to adopt the online setting for recommendation, especially given that it would invalidate several recommendation techniques that perform well in batch. Although we review in this previous work related to SBRS in this article, several essential research questions remain open, some of which are mentioned in the following.

The change detection module. Little effort has been made in the area of drift detection for SBRS. As shown in the field of data stream mining, active methods for drift detection can be very useful and improve prediction performance. Several types of drifts should be considered, such as gradual and abrupt drifts. In particular, recurrent drifts tend to appear in online RS. In fact, users can be affected by recurring factors derived from seasonal changes, for example. Handling such drifts

Table 3. Public Datasets Used for Evaluating SBRS

Dataset	Users	Items	Events	Reference	SBRS	Used in the Papers
Amazon Books	2.5M	929k	12.8M	[104]	N	[55]
Amazon Electronics	884k	96k	1.3M	[104]	N	[55]
Amazon Food	112k	23k	154k	[104]	N	[46]
Amazon Movies	1.3M	235k	8.5M	[104]	N	[46, 55, 68]
BookCrossing	278k	271k	1.1M	[156]	N	[62, 126]
Dating	194k	194k	11.7M	[33]	Y	[126, 143]
Delicious	1.8k	69k	105k	[34]	Y	[96]
Douban	36k	726k	5.3M	[154]	Y	[96]
Epinions	50k	140k	660k	[99]	N	[102, 103]
Flixster	147k	48k	8.1M	[4]	N	[102]
Globo	314k	46k	3M	[7, 43]	Y	[43, 56]
Jester	59k	140	1.7M	[64]	Y	[108, 143, 148]
Last.fm-600k	164	65k	493k	[37, 138]	Y	[103, 137–139, 141]
Last.fm-30M	40k	5.6M	31M	[134]	Y	[55]
MovieLens-100k	1k	1.7k	100k	[67]	N	[15, 39, 42, 62, 78, 80, 102, 103, 143]
MovieLens-1M	6k	4k	1M	[67]	N	[12, 14, 46, 71, 102, 103, 117, 123, 125, 137–139, 141, 143, 145]
MovieLens-10M	72k	10k	10M	[67]	N	[14, 39, 55, 71, 124, 143, 148]
MovieLens-20M	138k	27k	20M	[67]	N	[144, 148]
MovieTweetings	58k	33k	794k	[8]	N	[124]
Netflix	480k	17k	100M	[3]	N	[39, 102, 103, 117, 124, 127, 128, 144, 145]
Outbrain	700M	560	2B	[5]	Y	[76]
Palco Principal	7.5k	30k	588k	[6]	Y	[141]
Plista	220k	835	1.1M	[79]	Y	[15, 76, 97]
Taobao	1.1M	462k	13.7M	[9]	Y	[145]
Twitter	413k	37k	35M	[146]	Y	[48, 49]
Yahoo! Music-6k	6k	127k	476k	[1, 141]	N	[141]
Yelp	25.k	25.8k	731k	[2]	N	[68]

The SBRS column denotes if the dataset is adapted to the streaming setting for recommendation (Y) or not (N) (more details are provided in Section 6.2.4).

allows the RS to benefit from previous observations corresponding to the same concepts. It requires managing a memory of past states to leverage previously acquired knowledge. Future work should consider developing SBRS that account for these types of drifts. Specific methods should also be designed to handle all types of drifts occurring on the user and item levels at once and in one framework. Studying this problem could be made easier with the availability of synthetic and real-world datasets, annotated with the occurrence of drifts. Such datasets would cover more information about the points in time when drifts happened, their types, and how they affected users and items. This would help in developing specific algorithms that can detect drifts in a real-world setting or even anticipate them, and adapt the models accordingly.

The retrieval module. The problem of efficiently retrieving recommendations has emerged in recent work, given that real-world systems handle large numbers of users and items. The proposed solutions have been focusing on efficiently retrieving recommendations once the model is trained. These approaches cannot be applied for online RS where models are constantly being updated. Nevertheless, adopting similar strategies is essential in the streaming setting that is subject

to limitations in terms of resources and computation time. Future work should consider the development of efficient methodologies to retrieve recommendations in an online setting. However, learning techniques that reduce the retrieval time, such as indexable MF, have also been proposed for batch RS [89, 150]. Future work could consider extending such approaches for SBRS.

Evaluation of SBRS. The prequential evaluation faces several limitations in the context of online recommendation [137]. In particular, recommendations are only evaluated against the single received observation. Relevant recommended items that were selected in the near past or that will be selected in the near future can be penalized. On another note, previous work related to SBRS only considers the evaluation of the accuracy and ranking of recommendations, e.g., using recall and DCG. Metrics related to other criteria such as diversity and novelty have to be adapted to the online setting and evaluated for existing online recommendation approaches. In addition, the evolution of metrics over time in a sequential setting has to be assessed. The feedback received from these evaluations could be used to adapt recommendations in real time. It could also be interesting to explore recent advances in debiasing the evaluation of RS, e.g., handling selection biases due to the natural user behavior or to the exposure to the output of RS, in a streaming setting [121, 147].

Connection with other areas in RS. An interesting direction of research could involve connections between online adaptive recommendation and other areas in RS. On one hand, one could consider introducing side information to SBRS, as it has been done in the scope of hybrid RS, for example [15], and context-aware RS [11]. On the other hand, it would be interesting to look into the design of cross-domain online adaptive RS. Cross-domain RS are known for their ability to enrich a target domain by transferring knowledge from auxiliary domains [36]. Knowledge acquired in real time could then benefit the target domain and could help in anticipating events or drifts. Further advances could consider the design of such methods and the evaluation of their performance.

Another important question that should be investigated further is related to the role of **Reinforcement Learning (RL)** techniques in the context of SBRS. Recent years have witnessed the proliferation of RS based on RL techniques, in particular based on multi-armed bandits [92, 129], where each item is represented by an arm. With such a formulation, the model progressively learns a strategy for the selection of arms while, on one hand, maximizing the user satisfaction, i.e., exploitation, and on the other hand, gathering more information about the relevance of arms, i.e., exploration. At every step, the algorithm selects an arm and receives feedback from the user that is then used to update the strategy. Although multi-armed bandits have been explored for several other problems such as clinical trials and adaptive routing in networks, their application in the scope of RS has benefited from further developments to include ideas based on CF [41], CARS [92], and non-stationarity [90].

The sequential functioning of RL techniques, where the feedback is collected at each step and used to update the model, aligns with the one on which SBRS are based. However, and although these techniques can cover at least two out of the three requirements of SBRS from Section 2.2, i.e., the online learning and non-stationarity requirements, further investigations should review their applicability in a real-time setting, e.g., computational performance and time or number of interactions until convergence, and, more generally, explore RL techniques in the scope of SBRS.

REFERENCES

[1] Yahoo! n.d. Ratings and Classification Data. Retrieved November 1, 2020 from https://webscope.sandbox.yahoo.com/catalog.php?datatype=r.

[2] Kaggle. Yelp Dataset. Retrieved April 30, 2021. https://www.kaggle.com/yelp-dataset/yelp-dataset.

[3] Kaggle. 2009. Netflix Prize Data. Retrieved November 1, 2020 from https://www.kaggle.com/netflix-inc/netflix-prize-data/data.

[4] Social Computing Data Repository at ASU - Flixster Dataset. Retrieved on April 30, 2021 from http://datasets.syr.edu/datasets/Flixster.html.
[5] Kaggle. 2016. Outbrain Click Prediction. Retrieved November 1, 2020 from https://www.kaggle.com/c/outbrain-click-prediction/data.
[6] INESC TEC. 2017. Music Streaming and Playlisting Activity from Music Streaming Service. Retrieved November 1, 2020 from https://rdm.inesctec.pt/dataset/cs-2017-003.
[7] Kaggle. 2018. News Portal User Interactions by Globo.com. Retrieve November 1, 2020 from https://www.kaggle.com/gspmoreira/news-portal-user-interactions-by-globocom.
[8] GitHub. 2018. MovieTweetings. Retrieved November 1, 2020 from https://github.com/sidooms/MovieTweetings.
[9] Tianchi. 2018. Welcome to Tianchi Data Sets. Retrieved November 1, 2020 from https://tianchi.aliyun.com/dataset/.
[10] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data*6 (2005), 734–749.
[11] Gediminas Adomavicius and Alexander Tuzhilin. 2015. Context-aware recommender systems. In *Recommender Systems Handbook*, F. Ricci and B. Shapira (Eds.). Springer, 191–226.
[12] Deepak Agarwal, Bee-Chung Chen, and Pradheep Elango. 2010. Fast online learning through offline initialization for time-sensitive recommendation. In *Proc. of KDD*. 703–712.
[13] Marie Al-Ghossein. 2019. *Context-Aware Recommender Systems for Real-World Applications*. Ph.D. Dissertation. Université Paris-Saclay (ComUE).
[14] Marie Al-Ghossein, Talel Abdessalem, and Anthony Barré. 2018. Dynamic local models for online recommendation. In *Companion Proc. of TheWebConf*. 1419–1423.
[15] Marie Al-Ghossein, Pierre-Alexandre Murena, Talel Abdessalem, Anthony Barré, and Antoine Cornuéjols. 2018. Adaptive collaborative topic modeling for online recommendation. In *Proc. of RecSys*. 338–346.
[16] Mohamed H. Ali, Ciprian Gerea, Balan Sethu Raman, Beysim Sezgin, Tiho Tarnavski, Tomer Verona, Ping Wang, et al. 2009. Microsoft CEP server and online behavioral targeting. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1558–1561.
[17] Xavier Amatriain and Justin Basilico. 2015. Recommender systems in industry: A Netflix case study. In *Recommender Systems Handbook*, F. Ricci and B. Shapira (Eds.). Springer, 385–419.
[18] Xavier Amatriain and Josep M. Pujol. 2015. Data mining methods for recommender systems. In *Recommender Systems Handbook*, F. Ricci and B. Shapira (Eds.). Springer, 227–262.
[19] John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. 2004. An integrated theory of the mind. *Psychological Review* 111, 4 (2004), 1036–1060.
[20] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the Xbox recommender system using a Euclidean transformation for inner-product spaces. In *Proc. of RecSys*. 257–264.
[21] Linas Baltrunas and Xavier Amatriain. 2009. Towards time-dependant recommendation based on implicit feedback. In *Proc. of RecSys*. 25–30.
[22] Arindam Banerjee, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S. Modha. 2007. A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. *Journal of Machine Learning Research* 8 (Aug. 2007), 1919–1986.
[23] David Ben-Shimon, Alexander Tsikinovsky, Michael Friedmann, Bracha Shapira, Lior Rokach, and Johannes Hoerle. 2015. RecSys challenge 2015 and the YOOCHOOSE dataset. In *Proc. of RecSys*. 357–358.
[24] András A. Benczúr, Levente Kocsis, and Róbert Pálovics. 2018. Online machine learning in big data streams. arXiv:1802.05872
[25] András A. Benczúr, Levente Kocsis, and Róbert Pálovics. 2019. Recommender systems over data streams. In *Encyclopedia of Big Data Technologies*, Sherif Sakr and Albert Zomaya (Eds.). Springer, 1–9.
[26] James Bennett and Stan Lanning. 2007. The Netflix Prize. In *Proc. of KDDCup*.
[27] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. 1995. Using linear algebra for intelligent information retrieval. *SIAM Review* 37, 4 (1995), 573–595.
[28] Mária Bieliková, Veronika Bogina, Tsvi Kuflik, and Roy Sasson. 2017. The 1st international workshop on temporal reasoning in recommender systems. In *Proc. of RecSys*. 368–369.
[29] Albert Bifet and Ricard Gavalda. 2007. Learning from time-changing data with adaptive windowing. In *Proc. of SDM*. 443–448.
[30] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3 (Jan. 2003), 993–1022.
[31] Matthew Brand. 2003. Fast online SVD revisions for lightweight recommender systems. In *Proc. of SDM*. 37–46.
[32] Leo Breiman. 1996. Bagging predictors. *Machine Learning* 24, 2 (1996), 123–140.
[33] Lukas Brozovsky and Vaclav Petricek. 2007. Recommender system for online dating service. arXiv:cs/0703042

[34] Peter Brusilovsky, Iván Cantador, Yehuda Koren, Tsvi Kuflik, and Markus Weimer. 2010. Workshop on information heterogeneity and fusion in recommender systems. In *Proc. of HetRec@RecSys*.
[35] Pedro G. Campos, Fernando Díez, and Iván Cantador. 2014. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction* 24, 1–2 (2014), 67–119.
[36] Iván Cantador, Ignacio Fernández-Tobías, Shlomo Berkovsky, and Paolo Cremonesi. 2015. Cross-domain recommender systems. In *Recommender Systems Handbook*, F. Ricci and B. Shapira (Eds.). Springer, 919–959.
[37] Óscar Celma. 2010. *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer.
[38] Badrish Chandramouli, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. 2011. StreamRec: A real-time recommender system. In *Proc. of SIGMOD*. 1243–1246.
[39] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A. Hasegawa-Johnson, and Thomas S. Huang. 2017. Streaming recommender systems. In *Proc. of WWW*. 381–389.
[40] Chen Chen, Hongzhi Yin, Junjie Yao, and Bin Cui. 2013. Terec: A temporal recommender system over tweet stream. *Proceedings of the VLDB Endowment* 6, 12 (2013), 1254–1257.
[41] Konstantina Christakopoulou and Arindam Banerjee. 2018. Learning to interact with users: A collaborative-bandit approach. In *Proc. of SDM*. 612–620.
[42] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: Scalable online collaborative filtering. In *Proc. of WWW*. 271–280.
[43] Gabriel de Souza Pereira Moreira, Felipe Ferreira, and Adilson Marques da Cunha. 2018. News session-based recommendations using deep neural networks. In *Proc. of DLRS@RecSys*. 15–23.
[44] Mukund Deshpande and George Karypis. 2004. Item-based top-*N* recommendation algorithms. *ACM Transactions on Information Systems* 22, 1 (2004), 143–177.
[45] Christian Desrosiers and George Karypis. 2011. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor (Eds.). Springer, 107–144.
[46] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic matrix factorization with priors on unknown values. In *Proc. of KDD*. 189–198.
[47] M. Benjamin Dias, Dominique Locher, Ming Li, Wael El-Deredy, and Paulo J. G. Lisboa. 2008. The value of personalised recommender systems toe-business: A case study. In *Proc. of RecSys*. 1–23.
[48] Ernesto Diaz-Aviles, Lucas Drumond, Zeno Gantner, Lars Schmidt-Thieme, and Wolfgang Nejdl. 2012. What is happening right now . . . that interests me? Online topic discovery and recommendation in Twitter. In *Proc. of CIKM*. 1592–1596.
[49] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme, and Wolfgang Nejdl. 2012. Real-time top-*N* recommendation in social streams. In *Proc. of RecSys*. 59–66.
[50] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. 2015. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine* 10, 4 (2015), 12–25.
[51] Pedro M. Domingos and Geoff Hulten. 2001. Catching up with the data: Research issues in mining data streams. In *Proc. of DMKD*.
[52] Ryan Elwell and Robi Polikar. 2011. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks* 22, 10 (2011), 1517–1531.
[53] Elena Viorica Epure, Benjamin Kille, Jon Espen Ingvaldsen, Rebecca Deneckere, Camille Salinesi, and Sahin Albayrak. 2017. Recommending personalized news in short user sessions. In *Proc. of RecSys*. 121–129.
[54] Yoav Freund and Robert E. Schapire. 1996. Experiments with a new boosting algorithm. In *Proc. of ICML*. 148–156.
[55] Erzsébet Frigó, Róbert Pálovics, Domokos Kelen, Levente Kocsis, and András A. Benczúr. 2017. Online ranking prediction in non-stationary environments. In *Proc. of RecSys*.
[56] P. Moreira Gabriel De Souza, Dietmar Jannach, and Adilson Marques Da Cunha. 2019. Contextual hybrid session-based news recommendation with recurrent neural networks. *IEEE Access* 7 (2019), 169185–169203.
[57] João Gama. 2010. *Knowledge Discovery from Data Streams*. Chapman & Hall.
[58] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2013. On evaluating stream learning algorithms. *Machine Learning* 90, 3 (2013), 317–346.
[59] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys* 46, 4 (2014), 1–37.
[60] Florent Garcin, Christos Dimitrakakis, and Boi Faltings. 2013. Personalized news recommendation with context trees. In *Proc. of RecSys*. 105–112.

[61] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proc. of KDD*. 69–77.
[62] Thomas George and Srujana Merugu. 2005. A scalable collaborative filtering framework based on co-clustering. In *Proc. of IEEE ICDM*. 625–628.
[63] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *Proc. of VLDB*, 518–529.
[64] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. 2001. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval* 4, 2 (2001), 133–151.
[65] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proc. of NIPS*. 2672–2680.
[66] Asela Gunawardana and Guy Shani. 2015. Evaluating recommender systems. In *Recommender Systems Handbook*, F. Ricci and B. Shapira (Eds.). Springer, 265–308.
[67] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems* 5, 4 (2016), 1–19.
[68] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proc. of SIGIR*. 549–558.
[69] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. 2016. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proc. of RecSys*. 241–248.
[70] Frank Hopfgartner, Torben Brodt, Jonas Seiler, Benjamin Kille, Andreas Lommatzsch, Martha A. Larson, Roberto Turrin, and András Serény. 2015. Benchmarking news recommendations: The CLEF NewsREEL use case. *SIGIR Forum* 49, 2 (2015), 129–136.
[71] Xunpeng Huang, Le Wu, Enhong Chen, Hengshu Zhu, Qi Liu, and Yijun Wang. 2017. Incremental matrix factorization: A linear feature transformation perspective. In *Proc. of IJCAI*. 1901–1908.
[72] Yanxiang Huang, Bin Cui, Jie Jiang, Kunqian Hong, Wenyu Zhang, and Yiran Xie. 2016. Real-time video recommendation exploration. In *Proc. of SIGMOD*. 35–46.
[73] Yanxiang Huang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu. 2015. TencentRec: Real-time stream recommendation in practice. In *Proc. of SIGMOD*. 227–238.
[74] Dietmar Jannach and Malte Ludewig. 2017. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proc. of RecSys*. 306–310.
[75] Alípio Jorge, João Vinagre, Pawel Matuszyk, and Myra Spiliopoulou. 2018. ORSUM chairs' welcome and organization. In *Companion Proc. of TheWebConf*.
[76] Michael Jugovac, Dietmar Jannach, and Mozhgan Karimi. 2018. StreamingRec: A framework for benchmarking stream-based news recommenders. In *Proc. of RecSys*. 269–273.
[77] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. 2010. Multiverse recommendation: *n*-Dimensional tensor factorization for context-aware collaborative filtering. In *Proc. of RecSys*. 79–86.
[78] Mohammad Khoshneshin and W. Nick Street. 2010. Incremental collaborative filtering via evolutionary co-clustering. In *Proc. of RecSys*. 325–328.
[79] Benjamin Kille, Frank Hopfgartner, Torben Brodt, and Tobias Heintz. 2013. The plista dataset. In *Proc. of NRS.*. 16–23.
[80] Takuya Kitazawa. 2016. Incremental factorization machines for persistently cold-starting online item recommendation. arXiv:1607.02858
[81] Daniel Kluver and Joseph A. Konstan. 2014. Evaluating recommender behavior for new users. In *Proc. of RecSys*. 121–128.
[82] Noam Koenigstein and Yehuda Koren. 2013. Towards scalable and accurate item-oriented recommendations. In *Proc. of ReSys*. 419–422.
[83] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proc. of CIKM*. 535–544.
[84] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proc. of KDD*. 447–456.
[85] Yehuda Koren and Robert Bell. 2015. Advances in collaborative filtering. In *Recommender Systems Handbook*, F. Ricci and B. Shapira (Eds.). Springer, 77–118.
[86] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *IEEE Computer*8 (2009), 30–37.
[87] Dominik Kowald, Elisabeth Lex, and Markus Schedl. 2020. Utilizing human memory processes to model genre preferences for personalized music recommendations. In *Proc. of HUMANIZE@IUI*.
[88] Miklós Kurucz, András A. Benczúr, and Károly Csalogány. 2007. Methods for large scale SVD with missing values. In *Proc. of KDDCup*, Vol. 12. 31–38.
[89] Dung D. Le and Hady W. Lauw. 2017. Indexable Bayesian personalized ranking for efficient top-*k* recommendation. In *Proc. of CIKM*. ACM, New York, NY, 1389–1398.

[90] Chang Li and Maarten De Rijke. 2019. Cascading non-stationary bandits: Online learning to rank in the non-stationary cascade model. In *Proc. of IJCAI*. 2859–2865.
[91] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: Fast and exact inner product retrieval in recommender systems. In *Proc. of SIGMOD*. 835–850.
[92] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proc. of WWW*. 661–670.
[93] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80.
[94] Nick Littlestone. 1988. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning* 2, 4 (1988), 285–318.
[95] Nathan N. Liu, Min Zhao, Evan Xiang, and Qiang Yang. 2010. Online evolutionary collaborative filtering. In *Proc. of RecSys*. 95–102.
[96] Xin Liu and Karl Aberer. 2014. Towards a dynamic top-*N* recommendation framework. In *Proc. of RecSys*. 217–224.
[97] Andreas Lommatzsch and Sahin Albayrak. 2015. Real-time recommendations for user-item streams. In *Proc. of SAC*. 1039–1046.
[98] Cornelius A. Ludmann. 2017. Recommending news articles in the CLEF news recommendation evaluation lab with the data stream management system odysseus. In *Working Notes of CLEF 2017—Conference and Labs of the Evaluation Forum*, Vol. 1866. CLEF.
[99] Paolo Massa and Paolo Avesani. 2006. Trust-aware bootstrapping of recommender systems. In *Proc. of ECAI*. 29–33.
[100] Pawel Matuszyk and Myra Spiliopoulou. 2014. Selective forgetting for incremental matrix factorization in recommender systems. In *Proc. of DS*. 204–215.
[101] Pawel Matuszyk and Myra Spiliopoulou. 2015. Semi-supervised learning for stream recommender systems. In *Proc. of DS*. 131–145.
[102] Pawel Matuszyk and Myra Spiliopoulou. 2017. Stream-based semi-supervised learning for recommender systems. *Machine Learning* 106, 6 (2017), 771–798.
[103] Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. 2015. Forgetting methods for incremental matrix factorization in recommender systems. In *Proc. of SAC*. 947–953.
[104] Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proc. of RecSys*. ACM, New York, NY, 165–172.
[105] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. arXiv:1606.03126
[106] Catarina Miranda and Alípio Mário Jorge. 2009. Item-based and user-based incremental collaborative filtering for web recommendations. In *Proc. of EPIA*. 673–684.
[107] Olfa Nasraoui, Jeff Cerwinske, Carlos Rojas, and Fabio Gonzalez. 2007. Performance of recommendation systems in dynamic streaming environments. In *Proc. of SDM*. 569–574.
[108] Tavi Nathanson, Ephrat Bitton, and Ken Goldberg. 2007. Eigentaste 5.0: Constant-time adaptability in a recommender system using item clustering. In *Proc. of RecSys*. ACM, New York, NY, 149–152.
[109] Rani Nelken. 2016. RecProfile'16: Workshop on profiling user preferences for dynamic, online, and real-time recommendations. In *Proc. of RecSys*. 411–412.
[110] Behnam Neyshabur and Nathan Srebro. 2015. On symmetric and asymmetric LSHs for inner product search. In *Proc. of ICML*. 1926–1934.
[111] Róbert Pálovics, András A. Benczúr, Levente Kocsis, Tamás Kiss, and Erzsébet Frigó. 2014. Exploiting temporal influence in online recommendation. In *Proc. of RecSys*. 273–280.
[112] Manos Papagelis, Ioannis Rousidis, Dimitris Plexousakis, and Elias Theoharopoulos. 2005. Incremental collaborative filtering for highly-scalable recommendation algorithms. In *Proc. of ISMIS*. 553–561.
[113] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-aware recommender systems. *ACM Computing Surveys* 51, 4 (2018), 1–36.
[114] Steffen Rendle. 2010. Factorization machines. In *Proc. of IEEE ICDM*. 995–1000.
[115] Steffen Rendle. 2012. Learning recommender systems with adaptive regularization. In *Proc. of WSDM*. 133–142.
[116] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proc. of UAI*. 452–461.
[117] Steffen Rendle and Lars Schmidt-Thieme. 2008. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proc. of RecSys*.
[118] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor (Eds.). Springer, 1–35.

[119] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2000. Application of dimensionality reduction in recommender system—A case study. In *Proc. of WebKDD*.
[120] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2002. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proc. of CIT*.
[121] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as treatments: Debiasing learning and evaluation. In *Proc. of ICML*. 1670–1679.
[122] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Proc. of NIPS*. 2321–2329.
[123] Zaigham Faraz Siddiqui, Eleftherios Tiakas, Panagiotis Symeonidis, Myra Spiliopoulou, and Yannis Manolopoulos. 2014. xStreams: Recommending items to users with time-evolving preferences. In *Proc. of WIMS*. 1–12.
[124] Qingquan Song, Shiyu Chang, and Xia Hu. 2019. Coupled variational recurrent collaborative filtering. In *Proc. of KDD*. 335–343.
[125] Qiang Song, Jian Cheng, and Hanqing Lu. 2015. Incremental matrix factorization via feature space re-learning for recommender system. In *Proc. of RecSys*. 277–280.
[126] Karthik Subbian, Charu Aggarwal, and Kshiteesh Hegde. 2016. Recommendations for streaming data. In *Proc. of CIKM*. 2185–2190.
[127] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. 2008. Investigation of various matrix factorization methods for large recommender systems. In *Proc. of IEEE ICDM*. 553–562.
[128] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. 2009. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research* 10 (March 2009), 251–258.
[129] Liang Tang, Yexi Jiang, Lei Li, and Tao Li. 2014. Ensemble contextual bandits for personalized recommendation. In *Proc. of RecSys*. 73–80.
[130] Christina Teflioudi and Rainer Gemulla. 2017. Exact and approximate maximum inner product search with LEMP. *ACM Transactions on Database Systems* 42, 1 (2017), 5.
[131] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. 2015. LEMP: Fast retrieval of large entries in a matrix product. In *Proc. of SIGMOD*. 107–122.
[132] Christina Teflioudi, Faraz Makari, and Rainer Gemulla. 2012. Distributed matrix completion. In *Proc. of IEEE ICDM*. 655–664.
[133] Alexey Tsymbal. 2004. *The Problem of Concept Drift: Definitions and Related Work*. Technical Report. Department of Computer Science, Trinity College Dublin, Ireland.
[134] Roberto Turrin, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 2015. 30Music listening and playlists dataset. In *Proc. of RecSys*, Vol. 1441.
[135] João Vinagre and Alípio Mário Jorge. 2012. Forgetting mechanisms for scalable collaborative filtering. *Journal of the Brazilian Computer Society* 18, 4 (2012), 271–282.
[136] João Vinagre, Alípio Mário Jorge, Albert Bifet, and Marie Al-Ghossein. 2019. ORSUM 2019 2nd workshop on online recommender systems and user modeling. In *Proc. of RecSys*. 562–563.
[137] João Vinagre, Alípio Mário Jorge, and João Gama. 2014. Evaluation of recommender systems in streaming environments. In *Proc. of RecSys*.
[138] João Vinagre, Alípio Mário Jorge, and João Gama. 2014. Fast incremental matrix factorization for recommendation with positive-only feedback. In *Proc. of UMAP*. 459–470.
[139] João Vinagre, Alípio Mário Jorge, and João Gama. 2015. Collaborative filtering with recency-based negative feedback. In *Proc. of SAC*. 963–965.
[140] João Vinagre, Alípio Mário Jorge, and João Gama. 2015. An overview on the exploitation of time in collaborative filtering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5, 5 (2015), 195–215.
[141] João Vinagre, Alípio Mário Jorge, and João Gama. 2018. Online bagging for recommender systems. *Expert Systems* 35, 4 (2018), e12303.
[142] João Vinagre, Alípio Mário Jorge, and Joao Gama. 2018. Online gradient boosting for incremental recommender systems. In *Proc. of DS*. 209–223.
[143] Jialei Wang, Steven C. H. Hoi, Peilin Zhao, and Zhi-Yong Liu. 2013. Online multi-task collaborative filtering for on-the-fly recommender systems. In *Proc. of RecSys*. 237–244.
[144] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. 2018. Neural memory streaming recommender networks with adversarial training. In *Proc. of KDD*. 2467–2475.
[145] Weiqing Wang, Hongzhi Yin, Zi Huang, Qinyong Wang, Xingzhong Du, and Quoc Viet Hung Nguyen. 2018. Streaming ranking based recommender systems. In *Proc. of SIGIR*. 525–534.
[146] Jaewon Yang and Jure Leskovec. 2011. Patterns of temporal variation in online media. In *Proc. of WSDM*. 177–186.
[147] Longqi Yang, Yin Cui, Yuan Xuan, Chenyang Wang, Serge Belongie, and Deborah Estrin. 2018. Unbiased offline recommender evaluation for missing-not-at-random implicit feedback. In *Proc. of RecSys*. 279–287.

[148] Tong Yu, Ole J. Mengshoel, Alvin Jude, Eugen Feller, Julien Forgeat, and Nimish Radia. 2016. Incremental learning for matrix factorization in recommender systems. In *Proc. of IEEE BigData*. 1056–1063.
[149] Markus Zanker, Laurens Rook, and Dietmar Jannach. 2019. Measuring the impact of online personalisation: Past, present and future. *International Journal of Human-Computer Studies* 131 (2019), 160–168.
[150] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proc. of SIGIR*. 325–334.
[151] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys* 52, 1 (2019), 1–38.
[152] Wancai Zhang, Hailong Sun, Xudong Liu, and X. Guo. 2014. An incremental tensor factorization approach for web service recommendation. In *Proc. of ICDMW*. IEEE, Los Alamitos, CA, 346–351.
[153] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *Proc. of SIGIR*. 183–192.
[154] Erheng Zhong, Wei Fan, Junwei Wang, Lei Xiao, and Yong Li. 2012. ComSoc: Adaptive transfer of user behaviors over composite social network. In *Proc. of KDD*. ACM, New York, NY, 696–704.
[155] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *Proc. of KDD*. 498–506.
[156] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *Proc. of WWW*. 22–32.

Received August 2019; revised December 2020; accepted February 2021