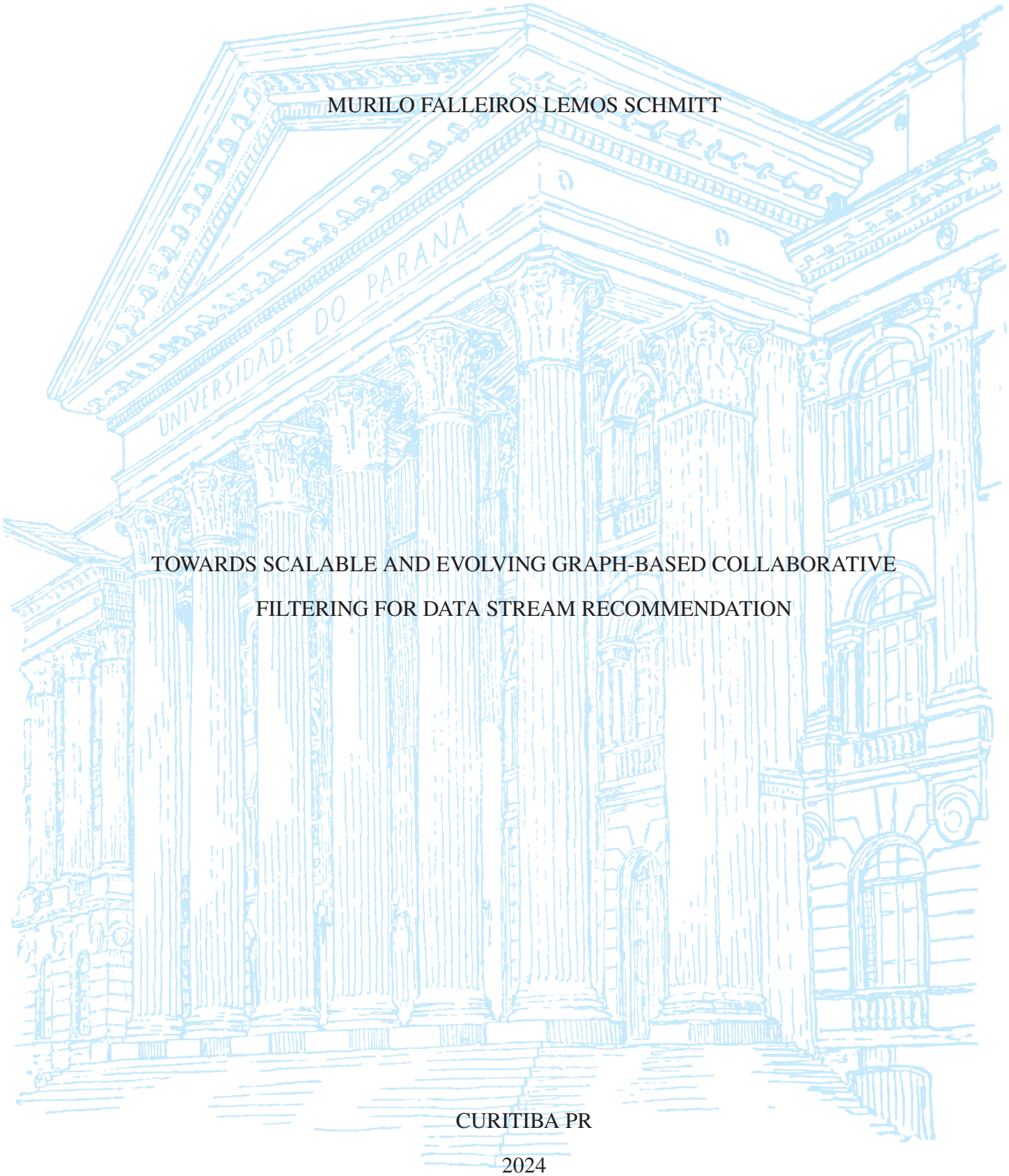
UNIVERSIDADE FEDERAL DO PARANÁ

MURILO FALLEIROS LEMOS SCHMITT



TOWARDS SCALABLE AND EVOLVING GRAPH-BASED COLLABORATIVE FILTERING FOR DATA STREAM RECOMMENDATION

CURITIBA PR

2024

MURILO FALLEIROS LEMOS SCHMITT

TOWARDS SCALABLE AND EVOLVING GRAPH-BASED COLLABORATIVE FILTERING FOR DATA STREAM RECOMMENDATION

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: Eduardo Jaques Spinosa.

CURITIBA PR

2024

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Schmitt, Murilo Falleiros Lemos
Towards scalable and evolving graph-based collaborative filtering for data stream recommendation / Murilo Falleiros Lemos Schmitt. – Curitiba, 2024.
1 recurso on-line : PDF.

Tese (Doutorado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.

Orientador: Eduardo Jaques Spinosa

1. Sistemas de recomendação (Filtragem de informações). 2. Fluxo de dados (Computadores). 3. Aprendizado do computador. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Informática. III. Spinosa, Eduardo Jaques. IV. Título.

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **MURILO FALLEIROS LEMOS SCHMITT** intitulada: **Towards Scalable and Evolving Graph-based Collaborative Filtering for Data Stream Recommendation**, sob orientação do Prof. Dr. EDUARDO JAQUES SPINOSA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 22 de Fevereiro de 2024.

Assinatura Eletrônica
27/02/2024 16:26:19.0
EDUARDO JAQUES SPINOSA
Presidente da Banca Examinadora

Assinatura Eletrônica
27/02/2024 18:50:29.0
JEAN PAUL BARDDAL
Avaliador Externo (PONTIFICIA UNIVERSIDADE CATOLICA DO PARANÁ- PUCPR)

Assinatura Eletrônica
28/02/2024 09:25:24.0
MARCOS AURÉLIO DOMINGUES
Avaliador Externo (UNIVERSIDADE ESTADUAL DE MARINGÁ)

Assinatura Eletrônica
28/02/2024 11:07:55.0
ANDRÉ LUIZ PIRES GUEDES
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

ACKNOWLEDGEMENTS

Ao longo do desenvolvimento desta Tese, contei com o apoio e contribuição de diversas pessoas, as quais gostaria de estender meus sinceros agradecimentos.

A Deus, pelo dom da vida e pelas inúmeras e contínuas bênçãos cedidas.

Aos meus pais, Taila e Sandoval, à minha irmã Marília e meu irmão Danilo. Agradeço pelo carinho, compreensão, apoio e amor incondicional. Obrigado por acreditarem em mim, por sempre incentivarem meus estudos e auxiliarem nos momentos de dificuldade. Este trabalho não teria acontecido sem vocês.

À minha namorada Isabela, por ser minha companheira e aceitar todos os momentos comigo, principalmente àqueles difíceis, aos quais seus incentivos e compreensão foram fundamentais. Agradeço também aos seus pais pelo apoio e acolhimento.

Ao meu orientador, professor Eduardo Jaques Spinosa, pela oportunidade de trabalho do Mestrado ao Doutorado, pela disponibilidade, pelos conselhos e feedbacks que foram essenciais ao trabalho, e que continuarão a impactar positivamente minha carreira. Agradeço também pela motivação, paciência, e principalmente pela confiança.

Aos professores da banca examinadora, André Luiz Pires Guedes, Jean Paul Barddal e Marcos Aurélio Domingues, pela disponibilidade, pela leitura cuidadosa da Tese, pelo encorajamento e pelas sugestões valiosas ao trabalho.

Aos amigos e colegas, agradeço o companheirismo, a ajuda, e os momentos de distração.

À todos os meus professores, do ensino básico à pós-graduação, pelo papel que tiveram na minha formação.

Por fim, à CAPES pelo apoio que permitiu o desenvolvimento deste trabalho.

RESUMO

Sistemas de recomendação são projetados para recomendar itens para usuários com base em seus interesses, a fim de aumentar seu engajamento ao interagir com sistemas online. O aumento significativo no volume de dados gerados por usuários em alta frequência resulta na necessidade de projetar sistemas de recomendação escaláveis e capazes de aprender dinamicamente, com base somente em dados recém gerados, ao menos tão rápido quanto à sua chegada, e que também sejam capazes de recomendar itens com base em informações atuais, com recursos de processamento e memória restritos. No entanto, algoritmos de recomendação bem sucedidos, que tradicionalmente dependem de re-treinamento esporádico a partir de dados previamente armazenados, não são projetados para adaptar-se de tal maneira, gerando problemas de escalabilidade e adaptabilidade. Uma alternativa é projetar sistemas de recomendação como uma tarefa de mineração de fluxos contínuos de dados, usando aprendizado incremental. Tal abordagem permite processar *feedback* de usuários continuamente e atualizar modelos de recomendação com base apenas em novos dados recebidos, sem a necessidade de armazenar estes dados, que podem então ser descartados após seu processamento. Embora sistemas de recomendação baseados em fluxos contínuos de dados estejam recentemente tornando-se um tópico ativo de pesquisa, ainda são pouco frequentes na literatura, e diversos problemas associados à sua aplicação, tais como esparsidade, precisão, escalabilidade e desvio de conceito, seguem em aberto. Neste contexto, a principal contribuição desta pesquisa é um modelo baseado em fluxo contínuo de dados, $\text{IGSI}_{\hat{\pi}^t}$, que consiste em um grafo de interações sequenciais com esquecimento para recomendação em fluxo contínuo de dados com *feedback* implícito. $\text{IGSI}_{\hat{\pi}^t}$ incorpora *feedback* em um grafo, cujos vértices representam itens, de maneira incremental, com a suposição de que o comportamento dos usuários pode ser extraído dessas sequências de interações com o passar do tempo, capturando interesses de curto prazo e de longo prazo. Nossa proposta é robusta à esparsidade, possui alta capacidade incremental e flexibilidade no processo de recomendação. A abordagem recomenda itens para usuários com base em simulações de passeios aleatórios curtos, o que permite a geração de recomendações escaláveis. Nosso trabalho também contribui com um mecanismo de esquecimento, *local neighborhood decay*, projetado especificamente para explorar as vantagens de $\text{IGSI}_{\hat{\pi}^t}$, e que pode ser generalizada para abordagens relacionadas. Tal mecanismo reutiliza as amostras de passeios aleatórios geradas originalmente para recomendação para capturar informações estruturais do grafo, e inferir a relevância dos itens. Arestas obsoletas são então eliminadas com base nessas informações e em fatores de popularidade. Avaliamos nossa proposta utilizando várias métricas e comparamos os resultados com vários algoritmos incrementais em fluxos de dados simulados. Os resultados demonstram a eficácia de nossa proposta, que em geral supera outros algoritmos em taxa de acerto, com tempos de atualização e recomendação muito competitivos. Além disso, os resultados demonstram que nossa técnica de esquecimento é capaz de aumentar escalabilidade, taxa de acerto e diversidade.

Palavras-chave: Sistemas de recomendação; Fluxo contínuo de dados; Aprendizado incremental; Esquecimento; Passeios aleatórios.

ABSTRACT

Recommender systems are designed to recommend items to users based on their interests, enhancing their engagement and satisfaction when interacting with online systems. The explosion of user-generated data at fast rates in online services leads to the need for designing scalable recommender systems that are able to learn on-the-fly. Such design requires learning from newly generated data on a single pass, at least as fast as data arrives, while also allowing the recommendation of relevant items based on up-to-date information with restricted time and processing requirements. However, successful recommendation algorithms, which traditionally rely on batch processing, are not designed to adapt to continuous flow of data, raising scalability and adaptability issues. An alternative approach is to view the recommendation problem under a data stream framework and design Stream-Based Recommender Systems, using incremental learning. This design allows continuous processing of user feedback and the update of models solely with incoming data, without requiring storage of observations, which can be discarded after processing. Although stream-based approaches are recently becoming an active topic of research, recommendation under the lens of data streams is still infrequent, and several issues still pertain, such as sparsity, accuracy, scalability and concept drift. Thus, the main contribution of this research is a stream-based model, $\text{IGSI}_{\hat{\pi}^t}$, that consists in an evolving graph of sequential interactions with forgetting for data stream recommendation with implicit feedback. $\text{IGSI}_{\hat{\pi}^t}$ incorporates feedback into an item-graph in incremental manner with the assumption that user behavior can be extracted from such sequence of interactions as time passes, capturing short-term and long-term interests. By focusing on a graph-based approach, our proposal is robust to sparsity, has natural incremental capability and flexibility on the recommendation procedure. It recommends items to users based on simulations of short random walks, which allows the generation of scalable recommendations. This work also contributes with a forgetting mechanism, local neighborhood decay, specifically designed to explore the advantages of $\text{IGSI}_{\hat{\pi}^t}$, that can be generalized to related approaches. This mechanism reuses the random walk samples originally generated for recommendation to capture structural information from the graph and infer the relevance of items. Obsolete connections are then faded based on this information and popularity factors. We evaluated our proposal under several metrics and compared the results with other related incremental algorithms on simulated data stream settings. The results suggest the effectiveness of our proposal, which generally outperforms competing algorithms in accuracy, with very competitive update and recommendation times. Also, the results suggest that our proposed forgetting technique is able to increase scalability, accuracy and diversity.

Keywords: Recommender systems; Data streams; Incremental learning; Forgetting; Random walks.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ALS	Alternating Least Squares
BPR	Bayesian Personalized Ranking
BPR-MF	Bayesian Personalized Ranking Matrix Factorization
BRISMF	Biased Regularized Incremental Simultaneous Matrix Factorization
CARS	Context-Aware Recommender Systems
CBF	Content-Based Filtering
CF	Collaborative Filtering
EASE^R	Embarrassingly Shallow Auto-Encoder
FIFO	First-In-First-Out
GAN	Generative Adversarial Network
IBPR-MF	Incremental Bayesian Personalized Ranking Matrix Factorization
IDF	Inverse Document Frequency
IGSI	Incremental Graph of Sequential Interactions
ItemKNN	Item-Based K-Nearest-Neighbors
ISGD	Incremental Stochastic Gradient Descent
KNN	K-Nearest-Neighbors
LND	Local Neighborhood Decay
MF	Matrix Factorization
PMF	Personalized Matrix Factorization
PPR	Personalized PageRank
RMFX	Stream Ranking Matrix Factorization
RS	Recommender Systems
RWR	Random Walk with Restart
SBRS	Stream-Based Recommender Systems
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition
TARS	Time-Aware Recommender Systems
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
UserKNN	User-Based K-Nearest-Neighbors
WMF	Weighted Matrix Factorization

LIST OF SYMBOLS

α	fading factor in forgetting
β	diversity parameter
γ	teleportation/restart factor on random walks
δ	upper bound on random walk sampling convergence
ε	associated error with random walk sampling
η	learning rate
λ	regularization factor
π	state probability distribution
ρ	time-window parameter
τ	acceptance factor in forgetting
ϕ	weight threshold parameter in forgetting
ω	window size parameter in forgetting

CONTENTS

1 INTRODUCTION

The ever-growing amount of information available on the Internet is overwhelming to users. In many real-world systems, such as news portals and e-commerce websites, users are interested in content that is unknown to them (Das et al., 2007), and rely on the system to present relevant content and to alleviate information overload. To that end, online systems must be able to match novel content to previous users' interests in order to improve their satisfaction and engagement when interacting with a system (Ricci et al., 2015). Systems designed to solve such recommendation tasks are known as *recommender systems*. The general goal of recommender systems is to present personalized sets of unknown and relevant items to users based on their past interests in order to increase users' satisfaction.

Significant attention has been given to the field of recommender systems since Netflix launched the Netflix Prize competition in 2006, where the objective was to build algorithms to beat their CineMatch algorithm by at least 10% for a grand prize (Bennett et al., 2007). The grand prize and the availability of a dataset with more than 100 million ratings sparked interest and activity from researchers in the field. Recommender systems have since been widely researched, resulting in many important contributions, specially in collaborative filtering (Koren and Bell, 2015), and are an integral part of several online systems from different domains, such as e-commerce (Linden et al., 2003; Smith and Linden, 2017), social networks (Backstrom and Leskovec, 2011; Gupta et al., 2013), videos and movies (Baluja et al., 2008; Davidson et al., 2010; Gomez-Uribe and Hunt, 2015; Covington et al., 2016), music (Celma, 2010; Jannach et al., 2015), touristic destinations (Cheng et al., 2013; Zhang et al., 2014a) and news (Das et al., 2007; Garcin et al., 2013; Trevisiol et al., 2014).

Collaborative filtering algorithms are based on the assumption that users with prior similar interests will share common interests in the future (Goldberg et al., 1992). Collaborative filtering techniques such as k-nearest neighbors (KNN) (Sarwar et al., 2001; Deshpande and Karypis, 2004) and matrix factorization (Koren et al., 2009; Rendle et al., 2009) are effective for recommendation and modeling long-term user interests. These techniques are designed to learn user's interests and to predict unknown user preferences based on past user feedback, i.e., interactions between users and items. Traditionally, predictive models are trained on static datasets with previously collected data, and predictions are made based on the inferred model.

1.1 MOTIVATION

Although collaborative filtering models are effective in generating personalized recommendation for users, they are usually designed as batch-learning algorithms. Batch-learning approaches require static training data to always be available in order to infer a model that is used for prediction, and the model remains unchanged until new data is available for retraining. Considering that in real world systems data is generated continuously at unpredictable rate, batch-learning approaches suffer from scalability and predictive capability issues inherently related to the assumption that models can be inferred and updated with static training data.

Scalability issues are related to the amount of data that is generated. Since the amount of data will always increase potentially at fast rates, it becomes unfeasible to retrain these models as data is generated. Consequently, the impracticability of retraining models in batch also results in increasingly less relevant recommendations. Assuming that batch updates are done sporadically and that data is continuously generated, user feedback generated between updates are not taken

into consideration by the model, resulting in an outdated model that is not sensitive to changes in user preferences and that disregards new incoming users and items into the system.

As highlighted by Vinagre et al. (2015b), recommender systems share several similarities with the field of data stream mining, and a natural approximation exists between the two fields. In data streams scenarios, data is generated as a chronologically ordered sequence at unpredictable rate and are potentially unbounded, while also subject to changes over time. Algorithms that learn from data streams must process data in a single-pass at least as fast as new data arrives (Domingos and Hulten, 2001). Recommendation algorithms that acknowledge user feedback as a chronological sequence are commonly referred to as time-dependent (Shi et al., 2014; Vinagre et al., 2015b) or sequence-aware algorithms (Quadrana et al., 2018). Most research in these algorithms, however, assume batch training to stay up-to-date, which eventually leads to scalability issues since data is generated continuously.

Thus, an alternative approach to batch processing is to view the recommendation problem under a data stream framework, and design Stream-Based Recommender Systems (SBRS) (Vinagre et al., 2021; Al-Ghossein et al., 2021). SBRS assume that time-dependent (sequential) user feedback is processed continuously, new users and items are constantly added to the system, and an always-available model should learn from incoming observations as fast as they arrive (Domingos and Hulten, 2001; Gama et al., 2009). This can be achieved through *incremental approaches*, which are capable of learning and updating models with incoming data, and observations can be discarded after processing. Consequently, incremental algorithms are viable options for recommendation. However, most recommender systems are not designed to deal with data as a stream, overlooking scalability and run time complexity issues (Vinagre et al., 2015b).

Although SBRS is recently becoming an active topic of research (Al-Ghossein et al., 2021; Vinagre et al., 2022a) and some incremental algorithms have been proposed (Papagelis et al., 2005; Miranda and Jorge, 2009; Takács et al., 2009; Vinagre et al., 2014b; Matuszyk et al., 2015; Yagci et al., 2017; Anyosa et al., 2018; Al-Ghossein et al., 2018b), recommendation approached as a data stream is still infrequent, thus posing a few research gaps that could be explored.

During this research, we have identified that graph-based approaches for collaborative filtering are well suited to solve the aforementioned issues related to batch processing, as the graph structure naturally allows the incremental inclusion of continuous incoming user feedback. Traditional approaches usually model user feedback in a bipartite graph, such that each user-item interaction is represented as an edge in the graph connecting a user node to an item node. However, even though incoming user feedback can be incrementally included in these models by adding new nodes and edges, successful recommendation algorithms based on random walks (Fouss et al., 2005, 2007; Gori and Pucci, 2007; Xiang et al., 2010) are not designed to adapt to a continuous flow of incoming data, resulting in scalability issues and therefore are not suited for data stream environments.

In the work of Cooper et al. (2014), it was suggested that simulation of random walks can be used for scalable recommendations with short random walks, mainly three-steps (P^3_α algorithm), outperforming previous methods and with similar accuracy to exact random walks computations. Subsequently in the work of Christoffel et al. (2015), an extension of P^3_α called RP^3_β was proposed, that performs a re-ranking of P^3_α based on item popularity in order to increase diversity. The random walks are explored through sampling techniques, resulting in a trade-off between diversity, efficiency and accuracy. Although these approaches are suited to operate under a data stream framework, they are not evaluated in such framework, disregard sequential information and recommend items to users based on long-term interests, i.e., there is

no consideration on the chronological order of the interactions and to when each interaction was made, which disregards the shifts in concepts usually present in data streams (Domingos and Hulten, 2001; Gama et al., 2009).

In this work we study the recommendation problem under a data stream framework by considering implicit user feedback as a continuous and unbounded chronological sequence. In such a setting, we consider that every incoming observation indicates a positive interaction from a user towards an item. The algorithm must be able to update the model *solely on the current observation*, and recommend relevant items to users, also known as top-N recommendation task, with up-to-date information in scalable manner.

1.2 OBJECTIVE

The main objective and contribution of this research is to propose and evaluate recommender systems that are designed to work in data streams scenarios, where user feedback is continuously generated at unpredictable rates and is potentially unbounded. In such scenarios, incremental models must be able to update in a single-pass over the incoming observations and generate recommendations with up-to-date information. Besides the issues inherently related to recommender systems and data stream mining, such as the cold-start problem and concept drift, these considerations have direct impact in scalability and accuracy, since algorithms have limited time and resources to update and recommend relevant items on-the-fly. Additionally, how to properly evaluate recommender systems is still an open issue (Gunawardana et al., 2022), and related work usually assesses the effectiveness of algorithms by only one aspect of recommendation quality, which is accuracy.

Considering this problem scenario and the motivation described so far, the main objective of this work is to propose a stream-based recommender system designed to work in online settings, tackle the issues inherently related to recommender systems and data stream mining, such as sparsity, accuracy, scalability and concept drift, and provide alternatives to the shortcomings of previous algorithms.

To that end, our contributions are centered on our proposed stream-based model - $\text{IGSI}_{\hat{\pi}^t}$ - that consists in an evolving graph of sequential interactions with forgetting for data stream recommendation with implicit feedback. We use the term *evolving* in the sense that the model *adapts* and learns, continuously and incrementally, based on information generated along the data stream, but also removes information that is deemed as obsolete. We focus on a graph-based approach due to its robustness to sparsity, natural incremental capability and flexibility. The aforementioned motivations and objective also leads us to the following research questions:

1. *Can graph-based algorithms for collaborative filtering be used effectively in data stream scenarios? How do they perform in scalability and accuracy compared to other incremental models?*

    - Previous work show that graph-based algorithms present competitive results in stationary settings (Dacrema et al., 2021). We have implemented and experimented with incremental versions of graph-based algorithms P^3_α and RP^3_β, using simulation of random walks to provide scalable recommendations, and made comparisons with several competing incremental approaches. Our experiments, presented in Chapter 6, highlight both the effectiveness of these approaches and also its shortcomings. Mainly, that sequential information is not taken into account, i.e., short-term and long-term interests are not distinguished.

2. *How can time-related information be incorporated into graph-based models? Does such information improve the quality of the recommendations?*

    - Our proposed SBRS, IGSI$_{\hat{\pi}^t}$, defined in Chapter 5, helps addressing this question. IGSI$_{\hat{\pi}^t}$ (Schmitt and Spinosa, 2020, 2022b) incorporates implicit user feedback into an item-graph in incremental fashion with the assumption that user behavior can be extracted from the sequence of user interactions as time progresses, capturing short-term and long-term interests. To that end, edges are continuously included in the graph and their weights are updated according to the sequential user interactions, such that for each incoming user feedback in a data stream, a directed edge connects the last item interacted by the user to the current interaction, and the relevance of each sequential interaction is reinforced in the weight of the edge. Results discussed in Chapter 6 highlight the importance of incorporating time-related information.

3. *Are current incremental algorithms sufficient in providing accurate, diverse and scalable recommendations? Can graph-based approaches effectively provide recommendations that are relevant in practice?*

    - Evaluation is a key aspect of recommender systems research, and high accuracy in offline experiments does not necessary leads to relevant recommendations in practice (McNee et al., 2006; Cremonesi et al., 2010; Vinagre et al., 2015b). Thus, evaluating other metrics such as diversity is necessary. In Chapter 6 we present comparisons of several incremental algorithms under different metrics, and discuss how these metrics may conflict. Also, we propose a forgetting mechanism in Chapter 5 that specifically allows for an increase in diversity without compromising accuracy. Our evaluation in Chapter 6 highlight its effectiveness.

4. *Considering that learning incrementally based on continuous incoming data leads to ever-growing models, what techniques can be used to remove obsolete information from these models? Are evolving methods enough to deal with issues inherently related to recommender systems and data stream mining, such as concept drift*?

    - Assuming that the volume of data will always grow and that users' preferences are not static, online models must adapt to changes by incorporating new information and forgetting obsolete ones (Matuszyk et al., 2018). The challenge is how to select information to be forgotten from already very sparse models, with the goal of increasing scalability and accuracy. In Chapter 5, we propose a scalable forgetting mechanism that removes obsolete information from graph-based (and neighborhood-based) methods while allowing for increases in diversity. The incorporation of such mechanism in IGSI$_{\hat{\pi}^t}$ increased scalability, accuracy and diversity, as suggested by our results in Chapter 6.

1.3 ORGANIZATION

The remainder of this work is organized as follows. In Chapter 2, we present general background concepts on recommender systems. Chapter 3 focuses on stream-based recommender systems, defines constrains of operating in data stream scenarios and also presents algorithms designed to operate under such scenarios. Chapter 4 reviews and discusses related work on stream-based recommender systems. In Chapter 5 we present our contributions, centered on our proposed SBRS, IGSI$_{\hat{\pi}^t}$. Chapter 6 presents the experimental evaluation of our proposed algorithms,

comparisons with competing algorithms and analysis of the obtained results. Finally, we present concluding considerations and discuss future work in Chapter 7.

2 RECOMMENDER SYSTEMS

In this section we introduce relevant definitions that constitute the foundation of this work. We first define notation and terminology used throughout this thesis. Then, we formulate the problem to be explored, and then present relevant information related to recommender systems. We also introduce important preliminary information on graphs. We note that algorithms discussed throughout this chapter are batch-based, i.e., they operate under the assumptions that past data is always available and that learned concepts are stationary. We discuss the limitations of such assumptions and possible solutions in subsequent chapters.

Notation. Notation and terminology used throughout this work is summarized in Table 2.1. All *matrices* are represented by bold upper case letters, e.g., $\mathbf{A}$, $\mathbf{W}$. *Vectors* are represented by bold lower case letters, e.g. $\mathbf{p}$, $\mathbf{q}$. Unless otherwise specified, they are assumed to be row vectors. *Sets* are represented by upper case calligraphic letters, e.g., $\mathcal{U}$, $\mathcal{I}$. The i-th row of a given matrix $\mathbf{A}$ is denoted by $\mathbf{a}_i$, the j-th column is denoted by $\mathbf{a}_j^T$ and an element corresponding to the i-th row and j-th column is denoted by a_{ij}. Sets of user-item interactions are denoted by $\mathcal{D}$. Also, an estimation of a random variable x is represented as $\hat{x}$.

2.1 PROBLEM FORMULATION

Recommender systems (RS) are software tools and techniques designed to provide personalized and useful suggestions of items from a large catalog to users (Ricci et al., 2015). Typically, when users interact with online systems, they are interested in content that is unknown to them, but that matches their previous interests. However, the number of items available is usually overwhelming. Thus, RS must filter novel and relevant content to users in order to alleviate information overload and increase their engagement and experience when interacting with such systems. Hence, the impact of RS is directly related to user satisfaction and revenue (Jannach et al., 2010; Ricci et al., 2015). In general, RS provide personalized recommendations to users by detecting patterns in past user behavior encompassed through actions such as ratings, purchases and clicks, and such behavior is used to recommend relevant content based on the inferred user profiles.

As outlined by Herlocker et al. (2004), RS are useful in a variety of tasks, such as rating prediction, finding relevant items, recommending a sequence of items, recommending browsing options, among others. Although all the aforementioned tasks generally consist in recommending items to users (Ricci et al., 2015), each task has different implications in the problem formulation and evaluation. In this work, we are interested in the task of recommending relevant items to users, i.e., provide to users ranked lists of items based on their interests. This problem is usually referred to as the top-N recommendation problem (Deshpande and Karypis, 2004).

The recommendation problem can be formalized as follows (Adomavicius and Tuzhilin, 2005). Let $\mathcal{U}$ denote the set of users, m the number of users $|\mathcal{U}|$, $\mathcal{I}$ the set of items that can be recommended and n the number of items $|\mathcal{I}|$. Let $x : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{R}$ denote a utility function that measures the usefulness of item $i \in \mathcal{I}$ to user $u \in \mathcal{U}$, where $\mathcal{R}$ is a totally ordered set. Then, for a given user u, the recommendation problem consists in choosing the item i'_u that maximizes the user's utility, as defined by the following equation:

$$i'_u = \arg\max_{i \in I} x(u, i) \tag{2.1}$$

Table 2.1: Table of notation.

Notation	Description
$\mathcal{U}$	set of users, $\mathcal{U} = \{u_1, u_2, ..., u_m\}$
m	number of users
$\mathcal{I}$	set of items, $\mathcal{I} = \{i_1, i_2, ..., i_n\}$
n	number of items
$\mathbf{R}$	feedback matrix
$\mathcal{D}$	set of user-item interactions
$\langle u, i, t \rangle$	user-item interaction at time t
$\mathcal{V}$	set of nodes
$\mathcal{E}$	set of edges
$w(e)$	weight of edge e
$\mathbf{A}$	adjacency matrix
$\mathbf{D}$	diagonal degree matrix
$\mathbf{P}$	transition probability matrix
$RW_{s,t}$	t-step random walk starting in node s
π_s^t	t-step random walk with restart vector starting in node s
S_u	ordered list of interactions by user u
li_u	last item interacted by user u
M	number of walks in random walk sampling
N	number of recommended items

The utility function x depends on the task. For rating prediction, the utility of an item is represented by a rating, while for the top-N recommendation problem, the utility is given by a relevance score. Thus, for the top-N recommendation problem, a recommender algorithm predicts the relevance score of each item and recommends the N most relevant items. The key problem of RS lies in that utility function x is only defined for a small subset of $\mathcal{U} \times \mathcal{I}$. Thus, the utility function must be extrapolated to the whole $\mathcal{U} \times \mathcal{I}$ space (Adomavicius and Tuzhilin, 2005). In other words, users only interact with a small subset of items, and the RS must be able to estimate unknown user-item interactions and make recommendations based on these predictions. The estimation of unknown user-item interactions is performed through methods from machine learning, approximation theory and various heuristics (Adomavicius and Tuzhilin, 2005).

Recommendation approaches, also referred to in this thesis as methods, techniques and algorithms, can be classified according to the recommendation problem that they solve, the approach to address the problem and the type of feedback that is used. The commonly used classification is based on how the information regarding users and items is used to generate the recommendations, and is defined as follows (Burke, 2007):

- Content-based filtering: these approaches recommend items similar to the ones the user showed interest in the past.
- Collaborative filtering: these approaches recommend to the active user items that users with similar behavior and interests liked previously.
- Hybrid methods: these approaches combine methods from both content-based and collaborative filtering in order to avoid their shortcomings (Burke, 2007).

Table 2.2: Example of explicit feedback matrix and implicit feedback matrix, respectively.

Users / Items	i_1	i_2	i_3	i_4	i_5
u_1	4	1			5
u_2	4		2		1
u_3		5	3		
u_4	2			5	

Users / Items	i_1	i_2	i_3	i_4	i_5
u_1	1	1			1
u_2	1		1		1
u_3		1	1		
u_4	1			1	

Collaborative filtering is the most successful and widely used technique (Shi et al., 2014; Ricci et al., 2015), and is the main focus of this work, specifically collaborative filtering with implicit feedback. Thus, next we describe the types of feedback and then discuss the aforementioned recommendation approaches.

2.1.1 Types of feedback

The design of recommendation algorithms is directly dependent on the type of user feedback that is available (Jawaheer et al., 2014; Aggarwal et al., 2016). User feedback can be categorized into the two following groups.

Explicit feedback. Explicit feedback is obtained directly through the user and is often given in the form of a rating for an item, expressing the user's interest towards that item. Such rating is usually measure through a numerical scale, such as 5-star scale, which conveniently details the level of preference, and can be represented in the form $\langle u, i, r \rangle$, indicating that user u assigned rating r to item i. Thus, explicit feedback is very informative. However, explicit feedback is not always available, since users might be unwilling to provide ratings, and are usually sparse when available. Consequently, RS can make use of implicit feedback as an alternative source of user preferences.

Implicit feedback. Implicit feedback (Oard et al., 1998) is inferred from observable user behavior and is collected from actions such as click-through data, music/movie streaming and item purchases, and does not require direct user involvement. Implicit feedback can be represented in the form $\langle u, i \rangle$, indicating a positive preference from user u towards item i. We will also refer to implicit feedback in the remainder of this work as positive-only feedback. Although easier to collect, implicit feedback is inherently sensitive to noise, which is related to the tools used for capturing user feedback (Pan et al., 2008; Jawaheer et al., 2014; Jannach et al., 2018).

Table 2.2 presents an example of both explicit and implicit feedback matrices. These examples highlight the sensitivity to noise related to implicit feedback and the differences in problem formulation, since negative user preferences would be considered to be positive. With explicit feedback, for the top-N recommendation problem the algorithm can predict the missing ratings and sort items by decreasing predicted rating. With implicit feedback, the feedback matrix assumes boolean values, where true values indicate a positive preference, and false values, i.e. missing values, can be interpreted as unknown or negative (Pan et al., 2008), and the algorithm must predict and rank the positive items to generate a recommendation list.

2.2 CONTENT-BASED FILTERING

Content-based filtering (CBF) techniques (Pazzani and Billsus, 2007) recommend items by analyzing their descriptive attributes and matching these features with user profiles. These attributes can be extracted from text, images, videos, audios and tags. By matching item attributes to user profiles, the system recommends to users items that are similar to the ones they previously showed interest. For example, if a user previously showed interest towards science fiction movies, CBF systems can recommend other movies from this genre.

Content-based systems are generally composed of three main components (Aggarwal et al., 2016). The first component is feature extraction, which is one of the main challenges of content-based recommenders. This module extracts domain knowledge from various sources, including unstructured data, and converts such information into vector-space representation (Lops et al., 2011). The extraction of relevant features is crucial for the effectiveness of content-based approaches (Aggarwal et al., 2016). The second module learns the user profile. Based on user feedback and the attributes of items, vectors representing user preferences are created in the same vector-space where items are represented. The final component is the filtering, which matches the item attributes to the user profile in order to generate recommendations.

As an example, consider a CBF system that describes items through textual features, e.g., movie summaries or product description. Each text document is represented by a vector in an n-dimensional space, where each dimension represents a term from the vocabulary. The importance of each word (term) in the document is usually determined by TF-IDF, which is the product of the term frequency (TF) by the inverse document frequency (IDF). Denote a user profile by $\text{Profile}(u)$, which can be defined by aggregating the profiles of items rated by u, and an item profile as $\text{Content}(i)$. The utility function (Eq. (2.1)) for a CBF can be defined as follows (Adomavicius and Tuzhilin, 2005):

$$x(u, i) = \text{Profile}(u) \otimes \text{Content}(i) \tag{2.2}$$

where $\otimes$ is a scoring function, such as the cosine similarity measure (Baeza-Yates et al., 1999).

These techniques offer two main advantages (Lops et al., 2011). The first is transparency. The explanation behind the recommendations can be provided by the set of content features that describe the items. The second advantage is the ability to recommend new items. CBF approaches can recommend items that have yet to be interacted by any user, since items are recommended based on their descriptive features. Thus, CBF systems can operate in item cold-start scenarios, where items are not interacted by enough users in order to become a candidate for recommendation (Schein et al., 2002).

However, CBF systems have several shortcomings (Adomavicius and Tuzhilin, 2005; Lops et al., 2011). The main shortcoming is content availability. The performance of CBF techniques is dependent on the features associated with users and items. Domain knowledge is often needed to characterize items. However, such information is not always available and automatic feature extraction is not a straightforward task to perform in domains such as videos, images and audio. Another issue is overspecialization. CBF approaches recommend items that are similar to the ones that the user has already rated, which limits the novelty of the recommendations, since the recommended items are always similar to content the user has seen before. The final shortcoming is user cold-start. In order to build a user profile and provide accurate recommendations, the system must collect enough information about users. Thus, when few interactions are available, the system will not be able to make relevant recommendations (Lops et al., 2011).

To overcome the shortcomings of content availability and overspecialization inherently related to content-based systems, collaborative filtering techniques can be used to make recommendations. These techniques are designed to find correlations between users and items based solely on the available user feedback. Hence, next we describe collaborative filtering techniques, which are the focus of this work. For more information regarding content-based filtering, we refer the reader to the works of Pazzani and Billsus (2007), Lops et al. (2011) and Aggarwal et al. (2016).

2.3 COLLABORATIVE FILTERING

Collaborative filtering (CF) models try to predict future user interests and make recommendations based on all past user feedback. The key assumption behind these models is that users that shared similar interests in the past are likely to have similar interests in the future (Ekstrand et al., 2011). Thus, there is an indirect collaboration between users in order to improve the system in generating relevant recommendations (Goldberg et al., 1992). Consider a system with m users and n items. The available feedback provided by users, which can be explicit or implicit (Section 2.1.1), can be used to build a user-item feedback matrix $\mathbf{R}_{m \times n}$, as illustrated in Table 2.2. These feedback matrices are typically very sparse, i.e., users only interact with a small proportion of items available. Thus, the challenge of CF methods is to predict unknown user preferences based on the known values in $\mathbf{R}$ and make recommendations based on these predictions.

By recommending items to users based on the feedback of other users, CF models are able to overcome the content availability and overspecialization related to content-based systems (Desrosiers and Karypis, 2011). CF methods can recommend items to users when the content of items is not available. Also, the relevance of items is evaluated according to other users instead of relying on content, thus reducing the overspecialization of the recommendations. However, an issue that arises from CF methods is the item cold-start problem.

CF methods can be classified into two main classes: memory-based (or neighborhood-based) and model-based methods (Breese et al., 1998; Koren and Bell, 2015; Ning et al., 2015). Memory-based methods recommend items based directly on the known user-item interactions, while model-based methods use these ratings to learn a predictive model. Next, we provide an overview of both classes of methods.

2.3.1 Memory-based approaches

Memory-based collaborative filtering approaches, also known as neighborhood-based, use the entire user-item feedback matrix $\mathbf{R}$ to generate recommendations by computing neighborhoods of users or items using similarity measures such as cosine similarity or Pearson correlation. Considering that the rows of a user-item feedback matrix $\mathbf{R}$ corresponds to users and the columns corresponds to items, the similarity between two users u and v is obtained by their respective rows $\mathbf{r}_u$ and $\mathbf{r}_v$, while the similarity between two items i and j is obtained by their respective columns $\mathbf{r}_i^T$ and $\mathbf{r}_j^T$. After the neighborhoods are computed, these approaches combine the preferences of neighbors to produce a top-N recommendation for the active user (Sarwar et al., 2001). These algorithms are known as k-nearest-neighbors (KNN) and are divided into two classes: user-based and item-based CF.

User-based. In the user-based recommendation approach, the algorithm first finds the neighbors of a active user u based on a similarity measure. The k most similar users to u constitute the set of neighbors $\mathcal{N}_k(u)$. The predicted rating $\hat{r}_{ui}$ of u for a candidate item i is given by the weighted

average of the ratings provided by the k users that belong in the neighborhood of u towards item i (Resnick et al., 1994), where the weight of each rating is measured according to the similarity function (Ekstrand et al., 2011):

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathcal{N}_k(u)} sim(u,v) r_{vi}}{\sum_{v \in \mathcal{N}_k(u)} sim(u,v)} \tag{2.3}$$

where r_{vi} represents the rating given to item i by user v and $sim(u,v)$ is the similarity function, such as the cosine similarity, applied to users u and v:

$$sim(u,v) = cos(\mathbf{r}_u, \mathbf{r}_v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{\|\mathbf{r}_u\| \times \|\mathbf{r}_v\|} = \frac{\sum_{i \in (\mathcal{I}_u \cap \mathcal{I}_v)} r_{ui} r_{vi}}{\sqrt{\sum_{i \in \mathcal{I}_u} r_{ui}^2} \sqrt{\sum_{i \in \mathcal{I}_v} r_{vi}^2}} \tag{2.4}$$

where $\mathcal{I}_u$ and $\mathcal{I}_v$ are the set of items rated by users u and v, respectively. Thus, ratings are predicted based on the opinion of users that are similar to the active user.

Item-based. Usually, the number of users in a system is much higher than the number of items. Thus, generating recommendations based on similarities between items results in scalability improvements compared to recommendations based on user-based similarities (Sarwar et al., 2001; Linden et al., 2003; Deshpande and Karypis, 2004). Analogously to the user-based approach, to predict the rating of a user u to a candidate item i, i.e., $\hat{r}_{ui}$, the item-based algorithm finds the neighbors of item i and examine the ratings given by u to these neighbors. First, the algorithm computes the similarity between item i and every other item in $\mathcal{I}$, and then selects the k most similar items to i to constitute the set of neighbors $\mathcal{N}_k(i)$. The predicted rating $\hat{r}_{ui}$ is computed as follows:

$$\hat{r}_{ui} = \frac{\sum_{j \in \mathcal{N}_k(i)} sim(i,j) r_{uj}}{\sum_{j \in \mathcal{N}_k(i)} sim(i,j)} \tag{2.5}$$

where r_{uj} is the rating given to item j by user u and $sim(i,j)$ is the similarity function between items i and j, such as the cosine similarity:

$$sim(i,j) = cos(\mathbf{r}_i^T, \mathbf{r}_j^T) = \frac{\mathbf{r}_i^T \cdot \mathbf{r}_j^T}{\|\mathbf{r}_i^T\| \times \|\mathbf{r}_j^T\|} = \frac{\sum_{u \in (\mathcal{U}_i \cap \mathcal{U}_j)} r_{ui} r_{uj}}{\sqrt{\sum_{u \in \mathcal{U}_i} r_{ui}^2} \sqrt{\sum_{u \in \mathcal{U}_j} r_{uj}^2}} \tag{2.6}$$

where $\mathcal{U}_i$ and $\mathcal{U}_j$ are the set of users that rated items i and j, respectively. In addition to the scalability advantages compared to user-based approaches (Linden et al., 2003; Deshpande and Karypis, 2004), item-based methods often provide more relevant recommendations since they are based directly on the ratings provided by the active user, while user-based methods predict ratings based on the opinion of similar users (Aggarwal et al., 2016).

The user-based and item-based similarities defined in Eqs.(2.4) and (2.6), respectively, are designed to work with explicit feedback. They can be adapted to work with implicit feedback by considering $r_{ui} = 1$ for every known observation $\langle u, i \rangle$, and $r_{ui} = 0$ for the remaining unknown values in the feedback matrix $\mathbf{R}$. With these considerations, the cosine measure and the implementation can be simplified, since the similarities between users or items are computed based on co-occurrence counts. For the user-based approach, Eq.(2.4) can be rewritten as:

$$sim(u,v) = cos(\mathbf{r}_u, \mathbf{r}_v) = \frac{\sum_{i \in (\mathcal{I}_u \cap \mathcal{I}_v)} r_{ui} r_{vi}}{\sqrt{\sum_{i \in \mathcal{I}_u} r_{ui}^2} \sqrt{\sum_{i \in \mathcal{I}_v} r_{vi}^2}} = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{\sqrt{|\mathcal{I}_u|} \times \sqrt{|\mathcal{I}_v|}} \tag{2.7}$$

For the item-based approach, Eq.(2.6) can be simplified similarly:

$$sim(i,j) = cos(\mathbf{r}_i^T, \mathbf{r}_j^T) = \frac{\sum_{u \in (\mathcal{U}_i \cap \mathcal{U}_j)} r_{ui} r_{uj}}{\sqrt{\sum_{u \in \mathcal{U}_i} r_{ui}^2} \sqrt{\sum_{u \in \mathcal{U}_j} r_{uj}^2}} = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{\sqrt{|\mathcal{U}_i|} \times \sqrt{|\mathcal{U}_j|}} \tag{2.8}$$

Similar to the prediction with explicit feedback, the prediction with implicit feedback can be made for the user-based and item-based approaches with Eqs.(2.3) and (2.5), respectively. The main difference is that $\hat{r}_{ui}$ becomes a score prediction, and a top-N recommendation list can be generated by sorting candidate items by decreasing $\hat{r}_{ui}$.

The main advantages of the aforementioned methods are their simplicity, straightforward implementation and explainability of recommendations. As these only require the computation of similarities and have only one parameter, k number of neighbors, their implementation and deployment is simple and straightforward. Also, it is easy to justify its underlying recommendations, specially for the item-based approach, since the system will recommend items that are similar to the ones the user previously interacted with (Linden et al., 2003; Aggarwal et al., 2016).

However, these methods suffer from scalability issues, since their space and time complexity grows with the number of users and items in the system (Sarwar et al., 2001). Another disadvantage is their sensitivity to sparsity. Since these methods require sufficient user feedback in order to characterize users and items and consequently generate relevant recommendations, the similarity computation is difficult when the occurrence of mutually interacted items between users is infrequent (Desrosiers and Karypis, 2011; Aggarwal et al., 2016).

Alternatively, graph-based models are suitable options to define similarities in memory-based methods in order to overcome sparsity limitations (Aggarwal et al., 2016). Hence, we next describe graph-based approaches.

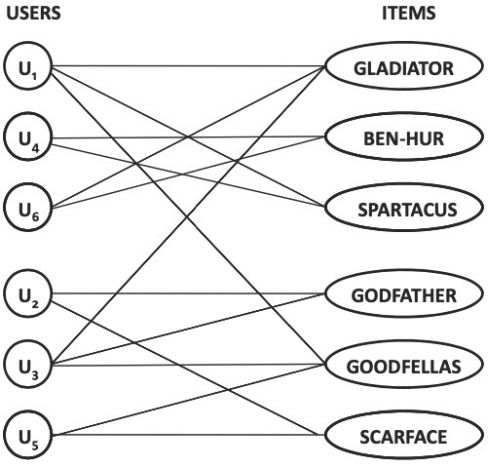
2.3.1.1 Graph-based approaches

The sparsity of the user-item feedback matrix poses a major challenge in the computation of similarity in the aforementioned memory-based methods. An alternative to overcome this challenge is to define similarities with graph-based models, where similarities are inferred with the use of either structural transitivity or ranking techniques (Aggarwal et al., 2016), which are described in this section. In graph-based models, data is represented as a graph where nodes are users, items or both, and edges connect nodes based on similarities or user-item interactions, and weights can be assigned to edges to represent the strength of the connection (Desrosiers and Karypis, 2011).

Figure 2.1 presents an example of a movie-based dataset modeled as a bipartite graph, where the two sets of nodes represent users and items, and a user u is connected by an edge to item i if u has interacted with i before. This example highlights the main difference between the user-based and item-based nearest neighbors approaches described previously, and graph-based approaches. In the former, the predicted rating $\hat{r}_{ui}$ of user u to item i is given using only the nodes connected directly to u or i. In sparse matrices, such direct connectivity exists only for a small subset of nodes, and there is no distinction on the similarity of unconnected nodes. For instance, consider item *Ben-hur*. By using a traditional nearest neighbors approach, the neighborhood of Ben-hur will include only items Gladiator and Spartacus, and no distinction can be made on the remaining items. In the latter, however, nodes that are *not* directly connected are allowed to influence each other by propagating information through the edges of the graph. The weight of

	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS	SCARFACE	SPARTACUS
U_1	1			5		2
U_2		5			4	
U_3	5	3		1		
U_4			3			4
U_5				3	5	
U_6	5		4			

(a) Example of a rating matrix.



(b) Corresponding user-item bipartite graph.

Figure 2.1: Example of a rating matrix and corresponding user-item bipartite graph for a movie-based dataset (Aggarwal et al., 2016).

an edge measures the amount of information that is allowed to pass through, and the influence of nodes that are closer to a source node should be higher than the influence of nodes that are further away in the graph, thus creating indirect connectivity between nodes (Desrosiers and Karypis, 2011; Aggarwal et al., 2016). This can be achieved for instance with random walk-based methods, described shortly after. We can see in Figure 2.1(b) that Ben-hur reaches every item on the graph if we expand the number of steps in a random walk, thus resulting in more meaningful neighborhoods. Hence, the exploitation of structural transitivity for the recommendation process reduces the impact of sparse feedback matrices (Aggarwal et al., 2016).

The structural transitivity is associated with the manner in which data is used to model the graph. In essence, items can be recommended to users with two different approaches: path-based and random walk-based strategies. In the path-based approach, the similarity between two nodes is inferred based on the number and length of paths connecting the two nodes. In the random walk-based approach, the similarity between users or items is evaluated as the probability of reaching these nodes in random walks (Desrosiers and Karypis, 2011; Aggarwal et al., 2016). Before we discuss these two approaches, we introduce some notation.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ denote a weighted graph, where $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ denotes the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges. Without loss of generality, we assume that $\mathcal{G}$ is strongly connected and directed, unless otherwise specified. Each edge e has an associated weight $w(e) \in \mathbb{R}_+$.

Let $\mathbf{A}$ denote the adjacency matrix of graph $\mathcal{G}$, where each entry a_{ij} specifies the weight of the edge connecting nodes i and j:

$$a_{ij} = \begin{cases} w(i,j) & \text{if } (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

For unweighted graphs, then $a(i,j) = 1$ if $(i,j) \in \mathcal{E}$ and 0 otherwise. Let $\mathbf{D}$ denote the diagonal degree matrix of $\mathcal{G}$ where the weighted out-degree of node i is: $d_{ii} = \sum_{(i,j)\in E} a_{ij}$.

A walk is a finite or infinite sequence of edges which joins a sequence of vertices. A path is a walk in which all vertices and therefore edges are different. A random walk is a process on $\mathcal{G}$ that begins at a given node, and at each step moves to an adjacent node selected at random. A t-step random walk starting from node $s \in \mathcal{V}$ is a random sequence of nodes $RW_{s,t} = \langle v_0, v_1, \ldots v_t \rangle$ where $v_0 = s$ and at the i-th step a node v_{i+1} is randomly chosen from the neighbors of v_i.

The probability of transitioning from node i to j is proportional to the weight of the corresponding edge (i, j):

$$p_{ij} = \begin{cases} \frac{a(i,j)}{\sum_{(i,k)\in\mathcal{E}} a(i,k)} & \text{if } (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

which can be simplified to $p_{ij} = a_{ij}/d_{ii}$. The sequence of random nodes $RW_{s,t}$ forms a Markov chain (Lovász et al., 1993; Jin, 2019). The transition matrix for a random walk is given by $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$, and the t-step random walk transition probability matrix is given by:

$$\mathbf{P}^t = (\mathbf{D}^{-1}\mathbf{A})^t \tag{2.11}$$

Let π^t denote the state probability distribution at step t, the rule of the walk can be expressed by (Lovász et al., 1993):

$$\pi^{t+1} = \pi^t P \tag{2.12}$$

Under the condition that the transition matrix $\mathbf{P}$ is row-stochastic, i.e., $\sum_{j\in\mathcal{V}} p_{ij} = 1, \forall i \in V$, this process converges to a stable distribution vector π^∞, which corresponds to the positive eigenvector of $\mathbf{P}$ with an eigenvalue of 1 (Desrosiers and Karypis, 2011). With the stationary distribution π^∞, items can be recommended according to their ranking in π^∞. This is the basis of the seminal algorithm PageRank (Page et al., 1999).

PageRank (Page et al., 1999) is an algorithm originally designed to rank pages in the Web, in order to list pages according to their relevance. PageRank is computed on the so called web-graph. Web-graph is a directed graph with n nodes, representing web pages, and edges representing hyperlinks, forming a $n \times n$ transition matrix. The underlying assumption in PageRank is that the existence of a hyperlink connecting a node u to another node v implies that page u votes for the relevance of page v, and a page that is connected to by many pages with high PageRank also receives a high rank itself. PageRank is defined as the stationary distribution of a random walk, such that the state space is the set of Web pages.

The web-graph, however, is not row-stochastic, i.e, several nodes do not have out-links. Hence, in order to make the graph connected, it is assumed that a random surfer can teleport to an arbitrary node with a small probability, rather than always follow an out-link from the current node. The new transition matrix $\mathbf{PR}$ for the graph is then written as (Avrachenkov et al., 2007):

$$\mathbf{PR} = \gamma\mathbf{P} + (1-\gamma)(1/n)\mathbf{E} \tag{2.13}$$

where $\mathbf{E}$ is a matrix such that all entries $e_{ij} = 1$, γ is the probability of selecting an adjacent node and $(1-\gamma)$ is the probability of jumping to a random page. Thus, following Eq.(2.12), PageRank vector $\pi_{\mathbf{PR}}$ can be computed as (Fogaras et al., 2005):

$$\pi^{t+1}_{\mathbf{PR}} = \gamma\pi^t_{\mathbf{PR}}\mathbf{P} + (1-\gamma)\mathbf{e}/n \tag{2.14}$$

where $\mathbf{e}$ is a vector with size n where all values are 1 and n is the number of nodes in the web-graph. The first term of the equation represents the case where the random walk follows an adjacent link from the current node with probability γ, and the second term represents the case where the random walk selects any page at random with probability $(1-\gamma)$ (Leskovec et al., 2020). Hence, a ranking of vertices can be produced with $\pi_{\mathbf{PR}}$.

Instead of selecting any node from the graph at random, it is possible to bias the probability distribution towards a set of nodes, resulting in a personalized ranking. This approach is known as Personalized PageRank (PPR) (Jeh and Widom, 2002; Haveliwala, 2003). Another possibility is to produce a ranking of nodes based on their relevance towards a source node. Then, instead of randomly selecting any node from the graph, the walk restarts on the source node. This approach is known as random walk with restart (RWR) (Leskovec et al., 2020).

In a random walk with restart, the walker moves from a node to one of its neighbors at random based on Eq.(2.10) with probability γ, and with $(1-\gamma)$ probability the walker returns back to the initial node $s \in \mathcal{V}$. Thus, nodes closer to the source node are considered to be more relevant since the walker will visit such nodes more frequently due to the restart factor. Let $\mathbf{e}_s$ denote a row vector with 1 in the column for node s and 0's elsewhere. The stationary distribution π_s^t, as $t \to \infty$, of the random walk with restart starting at node s can be expressed by (Andersen et al., 2006):

$$\pi_s^t = (1-\gamma)\mathbf{e}_s \sum_{t=0}^{\infty} \gamma^t \mathbf{P}^t \tag{2.15}$$

With these definitions, several algorithms can be used to rank vertices and consequently produce recommendations. As briefly mentioned, these algorithms can be categorized into two main approaches: path-based and random walk-based approaches. Next, we describe these methods.

Path-based strategy. In these approaches the distance between two nodes of the graph is measured as a function of the number of paths, and the lengths of these paths connecting the two nodes (Desrosiers and Karypis, 2011).

The first graph-based approach for CF computes the similarity between two users based on their shortest distance in a directed graph, termed *user-graph*, where nodes represent users and edges are determined based on the concepts of horting and predictability (Aggarwal et al., 1999). Horting is an asymmetric relation that measures the number of mutually rated items between two users. A user u horts another user v if $|\mathcal{I}_u \cap \mathcal{I}_v| \geq \chi$ or $\frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{|\mathcal{I}_u|} \geq \psi$, where $\mathcal{I}_u$ and $\mathcal{I}_v$ are the set of items rated by users u and v, respectively, and χ and ψ are predefined parameters. Predictability measures the level of similarity between the ratings of u and v, where v predicts u if u horts v and there exists a linear transformation $l(\cdot)$ such that

$$\frac{\sum_{k \in (\mathcal{I}_u \cap \mathcal{I}_v)} |r_{uk} - l(r_{vk})|}{|\mathcal{I}_u \cap \mathcal{I}_v|} \leq \omega$$

where ω is another predefined parameter. Directed edges in the graph represent the relations of predictability, such that an edge from u to v exists if v predicts u. The rating of a active user u for an item i is predicted using all the shortest directed paths from u to other users that have rated i (Aggarwal et al., 1999, 2016).

Data can also be modeled as an unweighted and undirected bipartite graph, defined as $\mathcal{G} = (V = \mathcal{U} \cup \mathcal{I}, \mathcal{E})$. We refer to such graphs throughout this thesis as *user-item bipartite graphs* or simply *user-item graphs* . In such a graph, the number of paths between a user u and an

item i can be a measurement of their compatibility (Huang et al., 2004). More specifically, it is possible to use the weighted number of distinctive paths between u and i, whose length is smaller than a given threshold K, in order to determine the affinity between them. Considering that the graph is bipartite, K should be an odd number, i.e., a path starts in a user node and stops at an item node. To decrease the influence of longer paths, the weight given to a path of length k is ϕ^k, where $\phi \in [0, 1]$, and the number of k length paths between pairs of nodes is given by A^k. Thus, the weighted number of k-paths between two nodes, also referred to as Katz measure, results in a user-item score matrix $\mathbf{S}^K$ given by (Desrosiers and Karypis, 2011; Aggarwal et al., 2016):

$$\mathbf{S}^K = \sum_{k=1}^{K} \phi^k \mathbf{A}^k \tag{2.16}$$

and recommendations can be made to a active user u by sorting candidate items based on the descending values in $\mathbf{s}_u^K$.

Random walk-based strategy. In these approaches, the similarity between two nodes is evaluated as the probability of reaching these nodes in a random walk. Several algorithms based on random walks have been proposed for the recommendation problem. To measure distance between two nodes in a graph, the hitting time and commute time measures can be used. Hitting time is the average number of steps needed in a random walk starting from a node i to reach a node j for the first time. The hitting time h_{ij} can be expressed recursively as (Sarkar et al., 2008):

$$h_{ij} = \begin{cases} 0 & \text{if } i = j \\ 1 + \sum_k p_{ik} h_{kj} & \text{otherwise} \end{cases} \tag{2.17}$$

A limitation of hitting time is that it is not symmetric. A related measure that overcomes this issue is the commute time, defined as $c_{ij} = h_{ij} + h_{ji}$ (Fouss et al., 2005; Sarkar et al., 2008), and corresponds to the average number of steps required to reach a node j from a node i, and go back to i. In a user-item bipartite graph, recommendations are made by sorting candidate items based on these metrics.

It is also possible to model user-item relations based on the correlation between items. In such a case, a weighted and directed graph is constructed where each node corresponds to an item from the catalog, and each edge corresponds to a relationship between two items. We refer to such graphs throughout this thesis as *item-graphs*. ItemRank (Gori and Pucci, 2007) is an effective algorithm based on PageRank (Page et al., 1999) that ranks the preferences of a active user u for candidate items i as the probability of u to visit i in a random walk on a directed item-graph, where nodes represent the items and edges connect items that have been rated by similar users. The edge weights are defined as the number of users that have rated both items, and a normalized correlation matrix $\mathbf{C}$ is defined, such that the correlation between two items is proportional to the number of users that rated both items, resulting in asymmetric weights.

Similar to PageRank, the random walk can, at any step t, visit an adjacent node defined in $\mathbf{C}$ with probability γ, or teleport to another node according to a probability distribution $\mathbf{e}_u$ with probability $(1 - \gamma)$. Thus, for an active user u, a personalized ranking vector $\mathbf{IR}_u$ can be iteratively computed until convergence as follows:

$$\mathbf{IR}_u^{t+1} = \gamma \cdot \mathbf{IR}_u^t \cdot \mathbf{C} + (1 - \gamma) \cdot \mathbf{e}_u \tag{2.18}$$

Finally, the algorithm recommends to the active user u items with the highest ranking in $\mathbf{IR}_u^{\infty}$. Experiments show that ItemRank is superior to other bipartite based algorithms, including hitting time, commute time and Katz measure (Fouss et al., 2005, 2007).

The main advantage of graph-based approaches is that these methods are more effective for sparse matrices, since they are able to capture indirect connectivity between nodes, specially with random walk-based methods (Aggarwal et al., 2016). Several approaches have been designed to rank nodes based on random walks (Baluja et al., 2008; Xiang et al., 2010; Vahedian et al., 2017; Eksombatchai et al., 2018; Nikolakopoulos and Karypis, 2019). Despite their advantages, the computation of stationary distributions is expensive and scalability issues arise for large datasets. An alternative to deal with this issue is to generate recommendations based on short-step random walks.

Cooper et al. (2014) proposed three ranking algorithms based on short-step random walks on bipartite graphs: $\mathbf{P}^3$, $\mathbf{P}^5$ and $\mathbf{P}^3_\alpha$. Let $\mathcal{G} = (\mathcal{V} = (\mathcal{U} \cup \mathcal{I}), \mathcal{E})$ denote a bipartite graph, where $\mathcal{U}$ and $\mathcal{I}$ represent the set of users and items, respectively, and $\mathcal{E}$ represents the set of edges, such that an unweighted edge exists between user u and item i if u interacted with i. This formulation results in a $(|\mathcal{U}| + |\mathcal{I}|) \times (|\mathcal{U}| + |\mathcal{I}|)$ adjacency matrix, where $a_{ui} = 1$ if u interacted with i and 0 otherwise. The ranking of items is made with the t-step random walk transition probability matrix $\mathbf{P}$ defined in Eq.(2.11), and in the bipartite graph t must be an odd number, so that the walk starts in a user node and finishes in an item node. Each entry p^t_{ui} gives the probability that user u reaches item i after t steps. Thus, $\mathbf{P}^3$ and $\mathbf{P}^5$ are based on the distribution of the random walk after three and five steps. Algorithm $\mathbf{P}^3_\alpha$ raises every entry from $\mathbf{P}^3$ to the power of a parameter α. Experiments show that taking longer steps deteriorates accuracy, and that $\mathbf{P}^3$ performs better than ItemRank (Gori and Pucci, 2007) and other bipartite graph-based algorithms (Fouss et al., 2005, 2007), and further improvements can be obtained with $\mathbf{P}^3_\alpha$.

Additionally, the work of Cooper et al. (2014) also highlighted the impracticability of implementing the aforementioned algorithms through matrix multiplication, since they require $O(|\mathcal{U}| + |\mathcal{I}|) \times (|\mathcal{U}| + |\mathcal{I}|)$ in space. Hence, the paper explores an alternative approach, which is to approximate the distributions through simulations of random walks. This method is much more memory efficient, since it requires $O(|\mathcal{V}| + |\mathcal{E}|)$ and the graphs are typically very sparse, albeit at the expense of accuracy. Recommendations to a given user u are made by ranking items based on information collected from samples of random walks starting from node u. Thus, the number of samples in the simulation results in a trade-off between scalability and accuracy.

Christoffel et al. (2015) argues that the ranking of items produced by algorithm $\mathbf{P}^3$ is highly influenced by their popularity. To compensate for the influence of item-popularity, they proposed algorithm $\mathbf{RP}^3_\beta$, which is based on $\mathbf{P}^3$. In order to increase diversity in the recommendations, $\mathbf{RP}^3_\beta$ re-ranks items based on their popularity. Let p^3_{ui} denote the probability that a 3-step random walk starting in user u ends in item i. $\mathbf{RP}^3_\beta$ re-weight the score as follows:

$$p^{\sim 3}_{ui} = \frac{p^3_{ui}}{d^\beta_{ii}} \tag{2.19}$$

where d_{ii} is the degree of node i and $\beta \in \mathbb{R}_+$ is a parameter that controls the influence of popularity, such that when $\beta = 0$, $\mathbf{RP}^3_\beta$ produces the same score as $\mathbf{P}^3$. Experiments show that $\mathbf{RP}^3_\beta$ increases diversity and also accuracy compared to $\mathbf{P}^3$ and $\mathbf{P}^3_\alpha$. Another contribution from Christoffel et al. (2015) is a sampling procedure to approximate algorithms $\mathbf{P}^3$, $\mathbf{P}^3_\alpha$ and $\mathbf{RP}^3_\beta$ with simulations of random walks as a Bernoulli process. We discuss this procedure in more detail in Chapter 3.

Overall, algorithms $\mathbf{P}^3_\alpha$ and $\mathbf{RP}^3_\beta$ are shown to be viable and competitive for the top-N recommendation problem (Dacrema et al., 2021). Next, we discuss model-based algorithms, which are established as state-of-the-art.

2.3.2 Model-based approaches

Model-based collaborative filtering approaches use the collection of user interactions to infer a predictive model that represents user preferences and item characteristics through a set of model parameters. These parameters are learned from the available training data and then the predictive model is used to provide recommendations (Adomavicius and Tuzhilin, 2005; Koren and Bell, 2015).

The main advantages of model-based algorithms in comparison to memory-based methods are space-efficiency, since the size of the inferred model is typically much smaller than the rating matrix, and also training and prediction speed, as memory-based methods must compute the similarities and then generate the recommendations, while model-based methods use the learned model to make predictions efficiently (Aggarwal et al., 2016). Despite these advantages, similar to memory-based approaches, model-based methods are sensitive to sparsity.

Several model-based approaches have been explored, such as Bayesian classifiers (Chien and George, 1999; Miyahara and Pazzani, 2000), clustering algorithms (Breese et al., 1998; Ungar and Foster, 1998), Markov decision processes (Shani et al., 2005; Rendle et al., 2010) and neural networks (Salakhutdinov et al., 2007; Wang et al., 2015; Wu et al., 2016; Zhang et al., 2019). The most successful techniques among model-based approaches are matrix factorization methods (Koren et al., 2009). Hence, next we discuss such techniques.

2.3.2.1 Matrix factorization approaches

Matrix Factorization (MF) methods (Koren et al., 2009) model both users and items to a common latent factor space of dimensionality k where the interest from a user towards an item is measured by the inner product of their embedded vectors. MF methods associates each user u with a vector $\mathbf{v}_u \in \mathbb{R}_k$, and each item i is associated with a vector $\mathbf{w}_i \in \mathbb{R}_k$. The latent factors measure the characteristics that an item possesses and the interest of users in these features. These features are inferred from past user behavior. Thus, the premise of MF methods is that the interest of user u towards an item i is captured by the dot product between their corresponding latent factors. The estimated value of the rating $\hat{r}_{ui}$ given by u to i is defined as:

$$\hat{r}_{ui} = \mathbf{v}_u \cdot \mathbf{w}_i^T \tag{2.20}$$

The number of latent features k is a parameter that controls the model's size. Thus, MF methods decompose the original rating matrix $\mathbf{R} \in \mathbb{R}_{m \times n}$ with m users and n items into two low-rank matrices $\mathbf{V} \in \mathbb{R}_{m \times k}$ and $\mathbf{W} \in \mathbb{R}_{n \times k}$ representing user and item latent factors respectively, and $\mathbf{R}$ is approximated as:

$$\mathbf{R} \approx \mathbf{V}\mathbf{W}^T \tag{2.21}$$

The major challenge is to compute the vectors $\mathbf{v}_u$ and $\mathbf{w}_i$ for each user u and item i, respectively. The learned latent factors can then be used to predict unobserved ratings in $\mathbf{R}$, and recommendations are computed based on these predictions.

Earlier approaches relied on single value decomposition (SVD) for matrix decomposition. SVD decomposes the rating matrix $\mathbf{R}$ into $\mathbf{R} = \mathbf{V}\Sigma\mathbf{W}^T$, where $\mathbf{V} \in \mathbb{R}_{m \times k}$ is the matrix of left singular vectors, $\mathbf{W} \in \mathbb{R}_{n \times k}$ is the matrix of right singular vectors and $\Sigma \in \mathbb{R}_{k \times k}$ is a diagonal

matrix containing the k singular values. By selecting the $d < k$ largest singular values and the corresponding singular vectors, matrix $\mathbf{R}$ can be approximated as $\mathbf{R} \approx \hat{\mathbf{R}}_d = \mathbf{V}_d \Sigma_d \mathbf{W}_d^T$. Since SVD requires a full matrix for decomposition, and matrices in the context of recommender systems are typically very sparse, SVD approaches for CF relied on value imputation to fill missing values in the rating matrix (Sarwar et al., 2000; Kurucz et al., 2007). However, imputation makes the rating matrix dense, raising scalability problems, and introduces bias in the data (Aggarwal et al., 2016).

An alternative to SVD is to learn the parameters $\mathbf{V}$ and $\mathbf{W}$ using only the observed ratings (Funk, 2006; Takács et al., 2007; Koren, 2008). To learn the vectors $\mathbf{v}_u$ and $\mathbf{w}_i$, training is performed by minimizing the regularized squared error for known ratings in $\mathbf{R}$ and the predicted rating (Koren et al., 2009):

$$\min_{\mathbf{v}^*, \mathbf{w}^*} \sum_{(u,i) \in \mathcal{D}} (r_{ui} - \mathbf{v}_u \cdot \mathbf{w}_i^T) + \lambda(\|\mathbf{v}_u\|^2 + \|\mathbf{w}_i\|^2) \tag{2.22}$$

where $\mathcal{D}$ is the set of user-item pairs for which r_{ui} is known and λ is a parameter that controls the amount of regularization in order to avoid overfitting. The two most popular methods to minimize this objective function are Stochastic Gradient Descent (SGD) (Funk, 2006; Koren, 2009) and Alternating Least Squares (ALS) (Bell and Koren, 2007). SGD-based optimization generally performs better in terms of accuracy and run time performance than ALS, although ALS benefits from parallel execution (Koren, 2009).

Stochastic gradient descent. Stochastic gradient descent (SGD) is an iterative optimization algorithm that approximates the true gradient based on an estimation computed from a randomly selected subset of the available observations. SGD first initializes the user and item latent factors $\mathbf{V}$ and $\mathbf{W}$ randomly, usually following a Gaussian distribution. Then, the algorithm performs several iterations, also known as epochs, until reaching a stopping criteria, such as convergence or maximum number of epochs. At each epoch, SGD loops over all observations $\langle u, i, r \rangle$, i.e., the rating given by user u to item i, and modifies the corresponding parameters $\mathbf{v}_u$ and $\mathbf{w}_i$ in the opposite direction of the gradient of the error by a magnitude proportional to $\eta \leq 1$, a hyperparameter known as learning rate. For each given training observation $\langle u, i, r \rangle$, the corresponding error is calculated as:

$$err_{ui} = r_{ui} - \mathbf{v}_u \cdot \mathbf{w}_i^T \tag{2.23}$$

and the latent factors $\mathbf{v}_u$ and $\mathbf{w}_i$ are updated as:

$$\begin{aligned} \mathbf{v}_u &= \mathbf{v}_u + \eta(err_{ui}\mathbf{w}_i - \lambda \mathbf{v}_u) \\ \mathbf{w}_i &= \mathbf{w}_i + \eta(err_{ui}\mathbf{v}_u - \lambda \mathbf{w}_i) \end{aligned} \tag{2.24}$$

Algorithm 1 describes the training procedure of SGD (Funk, 2006). MF models are considered to be state-of-the-art in recommender systems (Aggarwal et al., 2016), and several extensions (Koren, 2008; Takács et al., 2009; Koren, 2009) and algorithms have since been proposed, such as Personalized Matrix Factorization (PMF) (Mnih and Salakhutdinov, 2007), Weighted Matrix Factorization (WMF) (Hu et al., 2008), Bayesian PMF (Salakhutdinov and Mnih, 2008) and Bayesian Personalized Ranking Matrix Factorization (BPR-MF) (Rendle et al., 2009).

The SGD procedure described in Algorithm 1 is designed for explicit feedback, where the training data is a set of observations $\langle u, i, r \rangle$, each representing a rating r given by a user u to an item i. Algorithm 1 can be easily adapted to work with implicit feedback by assuming

Algorithm 1 Batch SGD

Require:
$\mathcal{D} = \{(\langle u, i, r\rangle)_1, ..., (\langle u, i, r\rangle)_n\}$: dataset
k: number of latent factors
it: number of iterations
λ: regularization factor
η: learning rate

```
for u ∈ users(D) do
    v_u ← initializeVector(k)
    v_u ~ N(0, 0.1)
for i ∈ items(D) do
    w_i ← initializeVector(k)
    w_i ~ N(0, 0.1)
for count ← 1 to it do
    D ← Shuffle(D)
    for ⟨u, i, r⟩ ∈ D do
        err_ui ← r_ui − v_u · w_i^T
        v_u = v_u + η(err_ui w_i − λ v_u)
        w_i = w_i + η(err_ui v_u − λ w_i)
```

that $r = 1$ for all cases (Vinagre et al., 2014b). This assumption impacts the training data, as the algorithm will assume $r = 1$ for every known $\langle u, i\rangle$, and the error calculation. For implicit feedback, the error calculation (Eq.(2.23)) is adapted to $err_{ui} = 1 - \mathbf{v}_u \cdot \mathbf{w}_i^T$. Hence, instead of predicting a rating, the prediction $\hat{r}_{ui}$ becomes a value representing the preference level of user u towards item i, measured as:

$$f_{ui} = |1 - \hat{r}_{ui}| \tag{2.25}$$

and candidate items for each user are sorted by descending proximity to 1 according to f_{ui}. In other words, Eq.(2.25) measures the proximity of the predicted rating to 1, which would indicate a positive preference of u towards i (Vinagre et al., 2014b).

Alternating least squares. Alternating least squares (ALS) alternates between fixing one of the two unknown parameters, $\mathbf{V}$ and $\mathbf{W}$, and repeatedly iterating over the remaining ones until convergence or satisfying a stopping criterion (Bell and Koren, 2007). All entries of matrices $\mathbf{V}$ and $\mathbf{W}$ are restricted to be non-negative, which avoids overfitting and guarantees convergence. Denote by $\mathbf{I}$ the identity matrix. In essence, ALS repeatedly executes the following two steps until convergence (Hu et al., 2008; Takács and Tikk, 2012):

1. Fix item matrix $\mathbf{W}$ and solve the least-square problem for the user latent matrix $\mathbf{V}$, defined as (Hu et al., 2008):

$$\mathbf{V} = (\mathbf{W}\mathbf{W}^T + \lambda\mathbf{I})^{-1}\mathbf{W}\mathbf{R}^T \tag{2.26}$$

2. Fix user matrix $\mathbf{V}$ and solve the least-square problem for the item latent matrix $\mathbf{W}$, defined as (Hu et al., 2008):

$$\mathbf{W} = (\mathbf{V}\mathbf{V}^T + \lambda\mathbf{I})^{-1}\mathbf{V}\mathbf{R}^T \tag{2.27}$$

2.3.2.2 *Learning to rank*

Another possible formulation for the recommendation problem besides the classical matrix completion one, is to generate a list of *ranked* items. In other words, instead of predicting the value of given ratings, the recommendation problem is posed as providing a ranking of items to users based on their perceived interests, i.e., top-N recommendation problem. The most popular ranking approach is the pairwise one, which considers the relative ranking of predictions for pairs of items. Such popularity is influenced by the proposal of Rendle et al. (2009).

BPR-MF. Rendle et al. (2009) proposed a Bayesian optimization method that generates personalized ranking on a set of items on a user basis, which is then applied on both matrix factorization and k-nearest-neighbors models. Their method, Bayesian Personalized Ranking (BPR), uses item pairs as training data (implicit feedback), and provides to each user u a personalized total ranking $>_u \subset \mathcal{I} \times \mathcal{I}$, satisfying the three properties of a total order. An important assumption in their method is that all items i that were consumed by u precede all items j that were *not* consumed by u in the total ranking $>_u$. Then, BPR is modeled to optimize the posterior distribution $p(\theta| >_u)$, where θ denotes the parameters of the underlying model, such as matrix factorization, resulting in algorithm BPR-MF, where $\theta = (\mathbf{V}, \mathbf{W})$, and $\mathbf{V}$ and $\mathbf{W}$ represent user and item latent factors, respectively.

Learning in BPR-MF relies on SGD, and uniform sampling to select so-called negative items (unseen) and artificially introduce negative feedback in the dataset. Observations in the dataset are defined as $\langle u, i, j \rangle$, where $\langle u, i \rangle$ represent an actual observation, and j represents an item sampled from the set of unobserved items by u. Relative ranking of i and j by u is predicted by:

$$\hat{r}_{uij} = \mathbf{v}_u \cdot \mathbf{w}_i^T - \mathbf{v}_u \cdot \mathbf{w}_j^T \tag{2.28}$$

For each observation $\langle u, i, j \rangle$, latent factors for user u and items i and j are updated as:

$$\Theta \leftarrow \Theta + \eta \left(\frac{e^{-\hat{r}_{uij}}}{1 + e^{-\hat{r}_{uij}}} \times \frac{\partial}{\partial_\Theta} \hat{r}_{uij} + \lambda_\Theta \Theta \right) \tag{2.29}$$

where

$$\frac{\partial}{\partial_\Theta} \hat{r}_{uij} = \begin{cases} \mathbf{w}_i - \mathbf{w}_j & \text{if } \Theta = \mathbf{v}_u \\ \mathbf{v}_u & \text{if } \Theta = \mathbf{w}_i \\ -\mathbf{v}_u & \text{if } \Theta = \mathbf{w}_j \\ 0 & \text{otherwise} \end{cases} \tag{2.30}$$

2.4 HYBRID METHODS

As discussed in the previous sections, CBF and CF techniques rely on different sources of data to make predictions and provide recommendations. As such, CBF and CF have their own advantages and shortcomings. CBF provides transparent recommendations and can operate in cold-start scenarios, while it suffers from overspecialization and content availability. CF on the other hand can recommend items when content is not available by estimating relevance based on the preferences of other users, while it suffers from cold-start issues.

Hybrid methods aim to combine CBF and CF to take advantage of its strengths and address its shortcomings. By combining different sources of data into a single model, the

limitation of content-based and collaborative systems, when used separately, can be reduced (Adomavicius and Tuzhilin, 2005; Burke, 2007). In fact, there is no restriction on combining recommender systems, and it is also possible to combine recommenders from the same class of technique (Burke, 2007). A categorization for hybrid RS that combines CBF and CF methods is presented as follows (Adomavicius and Tuzhilin, 2005):

- *Combining separate recommenders*. Separate collaborative and content-based systems can be implemented and their predictions combined to provide one final recommendation, using either linear combination, voting schemes, or selecting one of the recommenders based on a prediction metric (Pazzani, 1999; Adomavicius and Tuzhilin, 2005).

- *Adding content-based characteristics to CF models*. Content-based profiles for users can be maintained and used for similarity computation to reduce sparsity issues related to collaborative approaches (Pazzani, 1999). These approaches can reduce the impact of cold-start in CF methods (Burke, 2007).

- *Adding collaborative characteristics to CBF models*. Collaborative filtering models can be used on a group of content-based profiles to perform dimensionality reduction (Adomavicius and Tuzhilin, 2005).

- *Developing a single unifying recommendation model combining CBF and CF*. Many approaches have been devised following this approach, such as a unified probabilistic method for combining collaborative and content-based recommendations (Schein et al., 2002) and Bayesian regression models (Adomavicius and Tuzhilin, 2005).

2.5 CONTEXT-AWARE COLLABORATIVE FILTERING

In several applications, *contextual information* affects the recommendation process as the current circumstances impact the expectation of users and items' suitability for recommendation. For example, current user location is highly relevant when recommending points of interests. Similarly, in travel recommendations, appropriate destinations change according to the season and time of year. Another example is music consumption, where the preferences of user depend on the activity at hand or time of day. Finally, social information (company of other people) can affect the choice of movie to be watched. The assumption that *context* (location, temporal, social information) should be taken into consideration when providing recommendations further advanced the RS field. Recommender systems that somehow exploit such information are termed context-aware RS (CARS).

CARS model and predict user preferences by integrating contextual information into the recommendation process with additional dimensions of data besides user and item ones. The utility function formalized in Section 2.1, which models ratings as the function of only users and items, is adapted to $x : \mathcal{U} \times \mathcal{I} \times C \rightarrow \mathcal{R}$, where C covers the context domain, and represents the set of contextual features. Adomavicius et al. (2021) categorizes three paradigms to incorporate contextual information:

- *Contextual pre-filtering*. Context information filters data that are selected to be used as input for the recommendation model. An example is to split user profiles into several micro-profiles, each representing the given user in a particular context, such as time span (morning, evening, weekdays) (Baltrunas and Amatriain, 2009).

- *Contextual post-filtering*. Context is initially ignored and predictions are made with a conventional model. Then, the resulting recommendation list is adjusted to the active user considering the contextual information, e.g., removing items that do not match the current context or adjusting the ranking accordingly (Panniello and Gorgoglione, 2012).

- *Contextual modeling*. Directly uses contextual features at the learning module, resulting in multidimensional recommendation models, as defined by CARS' utility function. Existing neighborhood-based (Sarwar et al., 2001) and model-based approaches (Koren, 2009) can be extended to include contextual information (Adomavicius et al., 2021).

The major advantage of contextual pre-filtering and post-filtering is that it allows the usage of any two dimensional recommendation technique, i.e., modeled according to the original utility function. Contextual modeling, on the other hand, essentially represents predictive models that incorporate contextual information in the learning process in addition to the user and item data (Adomavicius et al., 2021).

For more information on CARS, see Adomavicius et al. (2021). In this thesis, we make use of temporal information by acknowledging that user feedback follows a naturally chronological order. Intrinsically to this premise is the assumption that user preferences and item dynamics, e.g. popularity, change over time at different rates based on different factors. Also, new users and new items enter the system while old ones leave. These factors should have directly influence on the design of recommender systems, as the user-item relations that these models aim to learn are actually dynamic (Vinagre et al., 2015b). Moreover, temporal information is generally easy to collect through timestamps and requires no additional effort from users. These assumptions somewhat intersects with two subareas of CARS: time-aware RS and sequence-aware RS. Although our work and contributions fundamentally differs from these two areas, similarities are shared based on these premises. Hence, we discuss them next.

2.5.1 Time-Aware Collaborative Filtering

Time-aware recommender systems (TARS) are a specialized type of CARS, with its main feature being the usage of time context information at some point of the prediction process, while being able to provide appropriate recommendations depending on the target recommendation time, based on previous preferences expressed by users at similar time contexts. In other words, time-aware models exploit time information related to past user preferences, such as timestamps associated with ratings/interactions (Campos et al., 2014).

Campos et al. (2014) identifies two ways of representing time variable: as a continuous contextual variable, e.g. timestamps, and as categorical contextual variables, e.g. day of week or season of year. Approaches that use timestamps - continuous time-aware approaches - include adapting classic models such as k-nearest-neighbors (Sarwar et al., 2001) and matrix factorization (Koren, 2009) to explicitly consider the target time in the predictions. Approaches that use categorical variables include the aforementioned use of micro-profiles (Baltrunas and Amatriain, 2009).

One of the most successful approaches for time-aware recommendation is tensor factorization (Karatzoglou et al., 2010; Rendle and Schmidt-Thieme, 2010; Xiong et al., 2010). Tensor factorization is similar to traditional MF. However, instead of factorizing a matrix, it factorizes a multidimensional tensor that includes extra dimensions, which represents contextual information, such as time features. This way, ratings are directly modeled based on the time that they occur, such that user, item and time latent factors are learned in the process. The rating matrix is extended into a three dimensional tensor $\mathbf{R} \in \mathbb{R}_{|\mathcal{U}|\times|\mathcal{I}|\times|\mathcal{T}|}$, where dimensions cover

the set of users $\mathcal{U}$, the set of items $\mathcal{I}$ and the set of time features $\mathcal{T}$. Similar to the MF model, users, items and time are modeled by latent factor vectors of k dimensions by minimizing the prediction error on known ratings. Denote by $\mathbf{x}_t$ the feature vector of time step t. The rating matrix is approximated as:

$$R \approx \sum_{j=0}^{k} \mathbf{v}_u \cdot \mathbf{w}_i \cdot \mathbf{x}_t \tag{2.31}$$

Tensor factorization in general outperforms static MF models (that do not account for time information) in terms of accuracy at the expense of higher time complexity (Xiong et al., 2010). For more information on TARS, we refer the reader to the survey of Campos et al. (2014).

2.5.2 Sequence-Aware Collaborative Filtering

Sequence-aware CF (Quadrana et al., 2018), also termed as time-dependent CF in the literature (Shi et al., 2014; Vinagre et al., 2015b), aim to capture and detect features attached to continuous temporal dynamics. These features include drifts in user preferences, changes in item dynamics such as popularity, short-term trends motivated by external factors and the emergence of new users and items. This is achieved by looking at user generated data as a chronologically ordered sequence of events (Vinagre et al., 2015b).

Sequence-aware CF shares several similarities to TARS, in that both are concerned with time-related phenomena. The main difference is that TARS is often concerned with the *exact* point of time of past user interactions, while sequence-aware is simply interested in the sequential order of events (Quadrana et al., 2018), which naturally captures the aforementioned features related to time, and try to detect patterns in the sequence itself.

Inherently related to sequence-aware CF are the notions of short-term and long-term user preferences/profiles. Whereas long-term represents the preferences of a given user learned from her past interactions, short-term represents her current interests, or even the interests of an anonymous/unregistered user (Ricci et al., 2021). Short-term preferences are related to limited periods of time when users interact with the systems called *sessions*.

Hence, two RS fields that intersect between time-aware and sequence-aware RS are session-based and session-aware recommendation. While session-aware RS assume that the system has knowledge about all past user sessions, thus being able to model long-term user preferences, session-based RS considers only the last session, and users are generally anonymous. One main advantage of modeling both short-term and long-term user preferences, when possible, is that it allows detection of the continuous temporal dynamics previously discussed. Our work investigates online incremental algorithms in a more generic setting, without explicitly considering sessions, but treat user feedback as a sequence. Surveys related to sequence-aware RS can be found in Vinagre et al. (2015b) and Quadrana et al. (2018), and a survey regarding session-based recommendation can be seen in Wang et al. (2021).

On the lines of TARS, algorithms for sequence-aware recommendation include adaptations of neighborhood-based and matrix factorization methods. One way to adapt these approaches to this setting is to emphasize more recent observations with less relevance given to past ones (Ding and Li, 2005; Liu et al., 2010; Vinagre and Jorge, 2012). Other algorithm approaches include frequent pattern mining, k-nearest-neighbors and first-order Markov chains (Ludewig and Jannach, 2018), reinforcement learning (Li et al., 2010; Pereira et al., 2019) and deep learning (Hidasi et al., 2016; Wu et al., 2019; Zhang et al., 2019). We note here that despite their simplicity, k-nearest neighbors approaches are among the most effective approaches for sequence-aware CF (Ludewig and Jannach, 2018).

Finally, treating data as a chronologically ordered sequence allows formulating the recommendation problem under a *data stream* setting (Domingos and Hulten, 2001; Gama, 2012; Vinagre et al., 2015b). Addressing the recommendation issue in such setting is the main focus of our work, and we discuss it in greater detail in the following chapters.

2.6 DISCUSSION

In this chapter, we introduced several concepts related to the topic of recommender systems, specially collaborative filtering techniques, which constitute the main focus of this work. These techniques are based on the assumption that users that shared similar interests in the past are likely to have similar interests in the future, thus creating an indirect collaboration between users in order to improve the recommendations.

Although CF models are generally efficient in modeling long-term user preferences, and short-term ones in the case of sequence-aware RS and its variants, they are traditionally designed to learn from batch-data, i.e., they require retraining to account for new data, and disregard runtime performance. This assumption results in several shortcomings, mainly related to scalability and adaptability, since the amount of data always increases, becoming unfeasible to retrain these models as data is generated.

To address these issues, we formulate the top-N recommendation problem under a data stream setting (Domingos and Hulten, 2001; Gama, 2012), which assumes that user feedback incomes continuously at fast rates, new users and items enter the system, known users change their opinion and tastes over time (Koychev and Schwab, 2000; Koychev, 2000) and an always-available and up-to-date model must provide recommendations when required. In the next chapter, we discuss such setting and its approaches, termed stream-based recommender systems (Vinagre et al., 2021; Al-Ghossein et al., 2021).

3 STREAM-BASED RECOMMENDER SYSTEMS

So far, we have discussed some of the most relevant collaborative filtering techniques. These techniques are batch-based, i.e., they operate under the assumption that past data is always available and that the learned concepts are stationary. Although they are generally effective for modeling user preferences, they also have several shortcomings. In real-world systems, user feedback is generated continuously at fast rates, unpredictable order, and is potentially unbounded.

Generally, most recommender systems built an initial model following a large static dataset that is then rebuilt periodically according to the arrival of new sets of data that are added to the original dataset. In order to incorporate such new information, batch-based algorithms require frequent retraining to stay up-to-date. Such batch-based procedure faces two main limitations, related to scalability and predictive capability.

First, considering that the amount of data will always increase, scalability issues arises and it becomes unfeasible to retrain these models as data is generated. As such, one way of overcoming this problem is to reduce the number of updates and only do so periodically. Conversely, this approach raises a *second* limitation. Performing only periodical updates limits the ability of these models in incorporating new information. In other words, the system disregards user feedback generated between updates and only considers it after the next one. This restricts its capabilities of tracking *temporal dynamics*, such as current user preferences and item popularity, even though users continuously provided more feedback. Disregarding continuously generated information means that the recommendation system is immediately outdated. An outdated model fails to quickly adapt to changes and to consider recent feedback, which negatively affects predictive performance, as recent user actions are not taken into consideration before recommendation.

An alternative approach to overcome the limitations of batch algorithms is to address recommendation as a data stream problem (Domingos and Hulten, 2001). In fact, recommender systems process data that is generated similarly to a data stream. New users and new items continuously enter the system, user feedback is generated at unpredictable rates and order, and previously known concepts are subject to changes over time, i.e., concept drift (Vinagre et al., 2015b). Therefore, scalability to deal with increasing amounts of data is a fundamental requirement for data stream settings.

In data stream mining, algorithms are designed to work incrementally. Incremental algorithms learn solely from incoming observations, and these can be discarded after processing. This premise addresses the shortcomings in scalability and missing adaptability related to batch algorithms, since new concepts will be incorporated in the model without retraining. Also, incremental updates offers a natural way to account for drifts in user preferences, as these will immediately be incorporated into the underlying recommendation model. Finally, by immediately incorporating information, the model is always maintained up-to-date. This leads to online adaptive learning (Gama et al., 2014). Essentially, online adaptive learning is formally defined based on three steps as follows (Gama et al., 2014):

1. *Predict.* When a new example x_t arrives, a prediction $\hat{y}_t$ is made using the current model;

2. *Evaluate.* After receiving the true label y_t, the loss function $f(\hat{y}_t, y_t)$ can be estimated;

3. *Update.* The example (x_t, y_t) can be used to update the model.

Table 3.1: Summary of the main differences between batch-based and data stream processing (Gama, 2012).

	Batch-based	Data streams
Data access	Random	Sequential
Number of passes	Multiple	Single
Processing time	Unlimited	Restricted
Available memory	Unlimited	Fixed
Result	Accurate	Approximate

In an online setting, such steps are repeated continuously. The main issue is that there is no control over the rate at which data is generated, which usually is very fast, nor the order that they arrive in. Also, data streams are potentially unbounded. Thus, in order to keep up with a stream, data needs to be discarded once processed as computational resources are restricted, since the model must be updated on the current example before performing the next prediction.

In order to operate under data streams, incremental algorithms must have the following prerequisites, as outlined by Domingos and Hulten (2001):

- It must process each observation faster than the rate of arrival of subsequent elements;
- It must use only a fixed amount of main memory, independently of the number of observations;
- It must be able to build a model with a single pass over the available data;
- A usable model must be available at any point in time, as opposed to when it finishes processing the data, since such processing may never end;
- The model must adapt to changes and stay up-to-date, but also maintain information that is not outdated.

The main characteristics of batch-based learning and data stream mining are summarized in Table 3.1 (Gama, 2012). Whereas in batch-based learning it is acceptable to perform multiple passes on the available data with random access, data stream processing must learn solely from sequentially incoming data on a single pass with restricted processing time and available memory. This difference directly affects predictive performance when we consider the three steps of online adaptive learning.

In batch-based learning, it is impossible to update the underlying model for every new example. As previously discussed, to overcome this problem a solution is to reduce the number of updates, only doing so sporadically. By only updating the model periodically, the *predict* and *evaluate* steps would consider an static model trained at a previous time point. Examples generated between updates are disregard until the next update, and would require storage to be used in the next update. Incremental learning, on the other hand, would immediately update the model with the new example that could then be discarded, thus being more suitable for real-world online scenarios.

Under a data stream framework, the top-N recommendation problem can be formalized as follows. Let $\mathcal{U} = \{u_1, u_2, ...\}$ denote the increasing set of users and $\mathcal{I} = \{i_1, i_2, ...\}$ the increasing set of items. User feedback is modeled as a data stream, where each observation is defined as $\langle u, i, t \rangle$, indicating that user u interacted with item i at time t. These interactions are

naturally a chronological sequence. The goal of an incremental model is to learn and update itself based on the current observation $\langle u, i, t \rangle$ and recommend relevant unknown items to users based on up-to-date information, allowing the continuous inclusion of new incoming users and items into the model in scalable manner. Thus, for each received observation, the aforementioned three steps of online adaptive learning (Gama et al., 2014) are adapted as follows for the top-N recommendation problem (Al-Ghossein et al., 2021):

1. *Predict.* Recommend a ranked list of N items for the active user with the current model;
2. *Evaluate.* After collecting the actual interaction of user u, evaluate the quality of the recommended list;
3. *Update.* The interaction $\langle u, i, t \rangle$ can be used to update the recommendation model.

In this work, we focus on implicit feedback, since in real-world system they can be easily collected from all users (Jannach et al., 2018), e.g., from click-through data. Hence, each incoming observation $\langle u, i, t \rangle$ indicates a positive preference from user u towards item i at time t, and no distinction can be made on the remaining unseen data, which can be negative or simply unknown (Pan et al., 2008).

We refer to RS designed to work on the data stream setting as *stream-based recommender systems* (SBRS) (Al-Ghossein et al., 2021). Although recommendation approached as a data stream is still infrequent (Vinagre et al., 2015b; Matuszyk et al., 2018), the field of SBRS (which is also posed as online recommender systems) has received increasingly interest in recent years (Vinagre et al., 2022a,b), and different incremental algorithms have been proposed (Papagelis et al., 2005; Miranda and Jorge, 2009; Takács et al., 2009; Vinagre et al., 2014b; Matuszyk et al., 2015; Yagci et al., 2017; Anyosa et al., 2018; Al-Ghossein et al., 2018b). In the following sections, we describe classes of incremental algorithms designed to learn from implicit feedback that are the most relevant within the scope of this thesis. We discuss related work on SBRS in the next chapter.

3.1 INCREMENTAL MEMORY-BASED APPROACHES

In Section 2.3.1, we described two traditional memory-based approaches: user-based and item-based KNN. These algorithms have been adapted to work incrementally, by maintaining a similarity matrix in memory and updating the values based on new incoming interactions. A user-based incremental KNN for explicit feedback is described in Papagelis et al. (2005), and extended in Miranda and Jorge (2009) to learn with implicit feedback. Miranda and Jorge (2009) also proposed an item-based algorithm for implicit feedback. We describe the item-based algorithm since it is more scalable and more relevant to this work. We note that the user-based algorithm can be obtained following the same formulation. In the remainder of this thesis, we refer to the user-based KNN as UserKNN and to the item-based KNN as ItemKNN.

ItemKNN. The cosine similarity function for implicit feedback defined in Eq.(2.8) measures the similarities between items based on user co-occurrence counts. The algorithm stores a matrix **IC** with the number of users that interacted with each pair of items, such that the diagonal of **IC** contains the number of users that interacted with each individual item. For each incoming observation $\langle u, i, t \rangle$, the corresponding counts are incrementally updated, and based on these counts, the similarities between items are recomputed and stored in an item-item similarity matrix **SM**:

$$SM_{ij} = sim(i, j) = \frac{IC_{ij}}{\sqrt{IC_{ii}} \times \sqrt{IC_{jj}}} \tag{3.1}$$

The algorithm for the incremental update of the similarity matrix **SM** is presented in Algorithm 2. As the algorithm is based on co-occurrence counts, the update procedure basically consists in incrementing these counts based on the current observation and previous user interactions (lines 2-4), updating the similarity matrix (line 5) and the rating matrix (line 6).

The recommendation procedure for ItemKNN is outlined in Algorithm 3. To provide a recommendation list for the active user u, based on updated similarity matrices obtained in the training procedure (Algorithm 2), a set of candidate items C is defined, containing all items that u has not interacted with before (line 1). The algorithm then computes a relevance score for each item $i \in C$, calculated with Eq.(2.5), based on the k nearest neighbors of i and the items previously interacted with by u (lines 2-4). Finally, candidate items are sorted by score and the N with highest score are recommended (lines 5-6).

Algorithm 2 Training procedure for incremental ItemKNN (Miranda and Jorge, 2009)

Require:
$\mathcal{D} = \{(< u, i, t >)_1, (< u, i, t >)_2, ...\}$: data stream
R: user-item rating matrix
1: **for** $< u, i, t > \in \mathcal{D}$ **do**
2: **for** $j \in \{k | R_{uk} = 1\}$ **do**
3: $IC_{ij} \leftarrow IC_{ij} + 1$
4: $IC_{ii} \leftarrow IC_{ii} + 1$
5: $SM_{ij} \leftarrow \frac{IC_{ij}}{\sqrt{IC_{ii}} \times \sqrt{IC_{jj}}}$ // Eq.(3.1)
6: $R_{ui} \leftarrow 1$

Algorithm 3 Recommendation procedure for incremental ItemKNN (Miranda and Jorge, 2009)

Require:
SM: item-item similarity matrix
R: user-item rating matrix
u: active user
k: number of neighbors
N: number of items to be recommended
1: $C \leftarrow \{x | R_{ux} = 0\}$ // Candidate items
2: **for** $i \in C$ **do**
3: $K_i \leftarrow \text{getKNN}(i, k)$
4: $\text{score}_i \leftarrow \frac{\sum_{j \in K_i} SM_{ij} R_{uj}}{\sum_{j \in K_i} SM_{ij}}$ // Eq.(2.5)
5: $\text{sorted}_u \leftarrow \text{sortByScore}(C)$
6: **return** $\text{topN}(\text{sorted}_u, N)$

Despite its simplicity, ItemKNN is a strong competitor in several stream-based recommendation scenarios (Lommatzsch and Albayrak, 2015; Jugovac et al., 2018; Ludewig and Jannach, 2018). The major shortcoming of this algorithm is that it requires the re-computation of the k-nearest neighbors and hence the relevance scores after similarities are incrementally updated. These computations can be performed at update or recommendation level, and this

decision therefore is dependent on the application. In our experiments, presented in Chapter 6, we have included ItemKNN as a baseline and opted to compute neighborhoods after similarity values are updated.

3.1.1 Incremental graph-based approaches

Although graph-based approaches for CF are well suited for sparsity problems and incremental updates, since the graph structure naturally allows the incremental inclusion of continuous incoming user feedback, successful recommendation algorithms are not designed to adapt to a continuous flow of incoming data, resulting in scalability issues and therefore are not suited for data stream environments.

$\mathbf{P}^3$, $\mathbf{P}^3_\alpha$ and $\mathbf{RP}^3_\beta$. As discussed in Section 2.3.1.1, algorithms $\mathbf{P}^3$, $\mathbf{P}^3_\alpha$ (Cooper et al., 2014) and $\mathbf{RP}^3_\beta$ (Christoffel et al., 2015) rank items based on 3-step random walks in a user-item bipartite graph, as defined in Eq.(2.11), outperforming previous graph-based methods (Fouss et al., 2007; Gori and Pucci, 2007). Additionally, Cooper et al. (2014) highlighted the impracticability of implementing these algorithms through matrix multiplication, and proposed to approximate the distributions through simulations of random walks, which is more time and memory efficient with only limited impact on accuracy. Christoffel et al. (2015) presents a sampling procedure to approximate algorithms $\mathbf{P}^3$, $\mathbf{P}^3_\alpha$ and $\mathbf{RP}^3_\beta$ modeled as a Bernoulli process.

While the sampling procedure is not evaluated on a data stream scenario, it is applicable in such scenario and thus these algorithms can be implemented incrementally. We have adapted these algorithms to operate in stream-based settings (Schmitt and Spinosa, 2022b) and consider them as baselines in Chapter 6. The effectiveness of these methods has been highlighted in recent comparative evaluation work (Dacrema et al., 2021). In the following, we describe this sampling procedure (Christoffel et al., 2015), which is outlined in Algorithm 4. We denote by $\hat{\mathbf{P}}^3$, $\hat{\mathbf{P}}^3_\alpha$ and $\hat{\mathbf{RP}}^3_\beta$ the approximate versions of $\mathbf{P}^3$, $\mathbf{P}^3_\alpha$ and $\mathbf{RP}^3_\beta$, respectively.

Algorithm 4 Estimating item scores of $\mathbf{P}^3$, $\mathbf{P}^3_\alpha$ and $\mathbf{RP}^3_\beta$ with random walk sampling (Christoffel et al., 2015).

1: **procedure** EstimateItemScore(v_u, α, β) // v_u: starting user vertex, α: hyperparameter, β: diversity hyperparameter
2: $\mathbf{s} \leftarrow$ a score array with initial values 0
3: $d_{uu} \leftarrow$ getDegree(v_u)
4: **while not** converged(s) **do**
5: $v_c \leftarrow$ getRandomNeighbor(v_u)
6: $d_{jj} \leftarrow$ getDegree(v_c)
7: $v_c \leftarrow$ getRandomNeighbor(v_c)
8: $d_{vv} \leftarrow$ getDegree(v_c)
9: $v_c \leftarrow$ getRandomNeighbor(v_c)
10: $d_{ii} \leftarrow$ getDegree(v_c)
11: $c_{RW} = \frac{(d_{uu} d_{jj} d_{vv})^{\alpha-1}}{d_{ii}^{\beta}}$
12: $\mathbf{s}[v_c] \leftarrow \mathbf{s}[v_c] + c_{RW}$
13: **return** c

Since algorithms $\mathbf{P}^3$ and $\mathbf{RP}^3_\beta$ can be directly obtained through $\mathbf{P}^3_\alpha$ by considering $\alpha = 1$ and $\beta = 0$, respectively, we outline the sampling procedure for $\mathbf{P}^3_\alpha$. Algorithm $\mathbf{P}^3_\alpha$ raises every

value in the transition probability matrix to the power of a parameter α. From Eqs.(2.10) and (2.11), each value in matrix $\mathbf{P}^1$ raised to the power of α is calculated as $(p^1_{ui})^\alpha = (a_{ui}/d_{uu})^\alpha$, such that a_{ui} is an entry in the adjacency matrix and d_{uu} is the degree of vertex u. The transition probability $(p^3_{ui})^\alpha \in \mathbf{P}^3_\alpha$ of user u reaching item i after a 3-step random walk is defined as:

$$(p^3_{ui})^\alpha = \sum_{v=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|}(p^1_{uj})^\alpha(p^1_{jv})^\alpha(p^1_{vi})^\alpha = \sum_{v=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|}\left(\frac{a_{uj}}{d_{uu}}\right)^\alpha\left(\frac{a_{jv}}{d_{jj}}\right)^\alpha\left(\frac{a_{vi}}{d_{vv}}\right)^\alpha \tag{3.2}$$

Since the graph is bipartite, all values in the adjacency matrix are 1 or 0, and Eq.(3.2) can be simplified to:

$$(p^3_{ui})^\alpha = \sum_{v=1}^{|\mathcal{I}|}\sum_{j=1}^{|\mathcal{U}|}\frac{a_{uj}a_{jv}a_{vi}}{(d_{uu}d_{jj}d_{vv})^\alpha} \tag{3.3}$$

In Eq.(3.3), the term $a_{uj}a_{jv}a_{vi}$ is 1 if a path from user u to item i through item j and user v exists in the graph and 0 otherwise. Thus, $(p^3_{ui})^\alpha$ is the aggregate of all paths with length 3 from u to i, where each path $RW_u = \{\mathcal{U}_u, \mathcal{I}_j, \mathcal{U}_v, \mathcal{I}_i\}$ contributes $c_{RW}^{(p^3_{ui})^\alpha} = \frac{1}{(d_{uu}d_{jj}d_{vv})^\alpha}$ to the total transition probability. The sampling procedure aims to estimate the distribution based on the contributions of each individual path.

Considering that a random walk sampling is more likely to follow paths traversing nodes with low degree than to traverse nodes with high degree, it is necessary to increase the contribution of paths with low probabilities and penalize the contribution of paths with high probabilities. This is done by weighting a path contribution c_{RW} with the inverse of its occurrence probability:

$$c_{RW} = \frac{1}{(d_{uu}d_{jj}d_{vv})^\alpha} * d_{uu}d_{jj}d_{vv} = (d_{uu}d_{jj}d_{vv})^{1-\alpha} \tag{3.4}$$

Finally, $\hat{\mathbf{P}}^3_\alpha$ can be approximated with Algorithm 1 by performing M random walks starting from a given user node u, and assigning the contribution of each walk, defined in Eq.(3.4), into the score of the destination node. Considering that $\mathbf{P}^3$ is equivalent to $\mathbf{P}^3_\alpha$ with $\alpha = 1$, the contribution of each random walk for $\hat{\mathbf{P}}^3$ can be simplified to $(d_{uu}d_{jj}d_{vv})^0 = 1$, such that the estimation becomes simply based on the number of visits to the destination nodes. To estimate the item ranking of $\mathbf{RP}^3_\beta$, the contribution of the walk is divided by the degree of the destination node raised to the power of parameter β.

The aforementioned sampling procedure can be modeled as a Bernoulli process by sampling M independent random walks starting from each user u. Christoffel et al. (2015) uses Hoeffding's inequality and Union bound to show that the rate of convergence is exponential, and that the $\varepsilon-$approximate estimate for any user is less than δ, which provides a lower bound for M as $\frac{\psi^2}{2\varepsilon^2}\log(\frac{2|U|}{\delta})$, where ψ is the maximum possible value for c_{RW}. Thus, the number of walks required increases with ψ, which is defined by the algorithm at use.

3.2 INCREMENTAL MATRIX FACTORIZATION APPROACHES

Earlier incremental matrix factorization approaches relied on performing incremental updates on models initialized in batch. One of the earliest work on this topic is presented in Sarwar et al. (2002), where an incremental SVD algorithm was proposed, such that the calculation of

new latent vectors are based on the current decomposition and appended to their corresponding matrices through the application of the fold-in method (Deerwester et al., 1990). However, as discussed in Section 2.3.2.1, SVD for recommendation has important accuracy and scalability limitations.

BRISMF. Takács et al. (2009) proposed incremental updates on a SGD model. The proposed algorithm, Biased Regularized Incremental Simultaneous Matrix Factorization (BRISMF), is similar to the batch-based SGD procedure defined in Algorithm 1, with the addition of user and item biases. Biases are meant to capture how the rating scale is used by users to evaluate items. The main difference from BRISMF to the procedure defined in Algorithm 1 is that observations are treated sequentially. After initially training BRISMF in batch, the algorithm retrains only the user latent vectors *every time* new ratings are made available to each active user, while keeping the item latent vectors fixed. Although incremental, this approach requires computation of an initial model in batch, and the algorithm does not update item latent vectors, limiting its ability to capture item dynamics. Thus, retraining is necessary to include new items into the model, which in turn requires storing the rating matrix.

ISGD. To address the limitations related to maintaining models in batch, Vinagre et al. (2014b) proposed an incremental version of the batch-based SGD procedure defined in Algorithm 1, designed to learn from implicit feedback and update the latent factors $\mathbf{V}$ and $\mathbf{W}$ based solely on the current observation $\langle u, i, t \rangle$. This algorithm, incremental stochastic gradient descent (ISGD), is defined in Algorithm 5. Despite the similarities with batch-based SGD, ISGD presents two fundamental differences. First, it requires only a single pass over the incoming data, as the adjustments to the latent factors $\mathbf{v}_u$ and $\mathbf{w}_i$ are performed in a single step, which allows the discarding of observations after they are processed. Second, ISGD does not shuffle the training data and learns sequentially from the data as it is made available. This in turn allows ISGD to continuously include new users and items into the system without the need of retraining. Since the algorithm is designed to learn from implicit feedback, the error is calculated as $err_{ui} = 1 - \mathbf{v}_u \cdot \mathbf{w}_i$, and updates are done following Eq.(2.24). To provide a recommendation list for an active user u, candidate items are sorted based on Eq.(2.25), and the top-N items are then recommended to u.

Algorithm 5 ISGD (Vinagre et al., 2014b)

Require:
$\mathcal{D} = \{(< u, i, t >)_1, (< u, i, t >)_2, ...\}$: data stream
k: number of latent factors
λ: regularization factor
η: learning rate

1: **for** $< u, i, t > \in \mathcal{D}$ **do**
2: **if** $u \notin U$ **then**
3: $\mathbf{v}_u \leftarrow$ initializeVector(k)
4: $\mathbf{v}_u \sim \mathcal{N}(0, 0.1)$
5: **if** $i \notin I$ **then**
6: $\mathbf{w}_i \leftarrow$ initializeVector(k)
7: $\mathbf{w}_i \sim \mathcal{N}(0, 0.1)$
8: $err_{ui} \leftarrow 1 - \mathbf{v}_u \cdot \mathbf{w}_i^T$
9: $\mathbf{v}_u = \mathbf{v}_u + \eta(err_{ui}\mathbf{w}_i - \lambda\mathbf{v}_u)$
10: $\mathbf{w}_i = \mathbf{w}_i + \eta(err_{ui}\mathbf{v}_u - \lambda\mathbf{w}_i)$

Experiments show that ISGD is able to update its concepts very fast compared to a UserKNN (Miranda and Jorge, 2009) and presents competitive accuracy compared to algorithms UserKNN and BPR-MF (Rendle et al., 2009). Another important contribution from Vinagre et al. (2014b) is an experimental prequential protocol designed to evaluate incremental algorithms. We discuss such protocol in more detail in Chapter 6.

RAISGD. A limitation of ISGD is that it learns solely from feedback considered to be positive, leading the model to converge to the positive class and eventually degrading accuracy. To overcome this issue, a follow-up algorithm artificially introduces negative feedback into the model by maintaining a global FIFO (First-In-First-Out) queue of all items seen in the stream, ordered according to the recency of item occurrences. For every new positive observation in the stream, the l oldest items are selected from the queue to be considered as negative feedback by the user, and the corresponding user latent factor is updated based on such negative feedback. After updating the model based on the current positive observation, this observation is inserted into the tail of the queue. In order to avoid penalizing infrequent items repeatedly, items that are used as negative feedback are also moved to the tail of the queue. This procedure, recency-adjusted ISGD (RAISGD) (Vinagre et al., 2015a), is presented in Algorithm 6.

Algorithm 6 RAISGD (Vinagre et al., 2015a)

Require:
$\mathcal{D} = \{(< u, i, t >)_1, (< u, i, t >)_2, ...\}$: data stream
k: number of latent factors
λ: regularization factor
η: learning rate
l: number of items to be considered as negative feedback

$Q \leftarrow$ queue()
for $< u, i, t > \in \mathcal{D}$ **do**
  if $u \notin U$ **then**
    $\mathbf{v}_u \leftarrow$ initializeVector(k)
    $\mathbf{v}_u \sim \mathcal{N}(0, 0.1)$
  if $i \notin I$ **then**
    $\mathbf{w}_i \leftarrow$ initializeVector(k)
    $\mathbf{w}_i \sim \mathcal{N}(0, 0.1)$
  for $k \leftarrow 1$ to $\min(l, \#Q)$ **do**
    $j \leftarrow$ dequeue(Q)
    $err_{uj} \leftarrow 0 - \mathbf{v}_u \cdot \mathbf{w}_j^T$
    $\mathbf{v}_u = \mathbf{v}_u + \eta(err_{uj}\mathbf{w}_j - \lambda\mathbf{v}_u)$
    enqueue(Q, j)
  $err_{ui} \leftarrow 1 - \mathbf{v}_u \cdot \mathbf{w}_i^T$
  $\mathbf{v}_u = \mathbf{v}_u + \eta(err_{ui}\mathbf{w}_i - \lambda\mathbf{v}_u)$
  $\mathbf{w}_i = \mathbf{w}_i + \eta(err_{ui}\mathbf{v}_u - \lambda\mathbf{w}_i)$
  if $i \in Q$ **then**
    remove(Q, i)
  enqueue(Q, i)

Essentially, RAISGD gives greater importance to more recent events. The experimental results show that RAISGD significantly outperforms ISGD, user-based KNN (Miranda and Jorge, 2009) and BPR-MF (Rendle et al., 2009) in music domain datasets in terms of time and accuracy, but is outperformed in accuracy in movie domain datasets. These results highlights

the importance of considering short-term (recent) and long-term interests when generating recommendations.

Incremental BPR-MF. Algorithm BPR-MF (Rendle et al., 2009), defined in Section 2.3.2.2, was originally designed for learning in batch. However, it can be implemented in incremental manner (Gantner et al., 2011; Vinagre et al., 2014b; Viniski et al., 2021). Incremental BPR-MF (IBPR-MF) is presented in Algorithm 7. Such implementation is similar to the update procedure of ISGD: as each new observation in the stream is made available (line 1), new users and items are included in the system without needing retraining (lines 2-7), and only affected user and item latent factors are updated following the ranking described in Section 2.3.2.2. The update procedure for a new observation $\langle u, i, t \rangle$ first uniformly samples an unseen item j by the active user u to be used as negative feedback (line 8). Then, the relative ranking of i, the actual observed item, and the negative item j is predicted based on Eq.(2.28) (line 9). Finally, latent factors for user u ($\mathbf{v}_u$) and items i and j ($\mathbf{w}_i$ and $\mathbf{w}_i$, respectively) are updated following Eqs.(2.29) and (2.30) (lines 10-12). We note that the update procedure (lines 8-12) can be performed multiple times for a single new observation, at the expense of higher computational cost.

Algorithm 7 IBPR-MF (Rendle et al., 2009; Gantner et al., 2011; Vinagre et al., 2014b; Viniski et al., 2021)

Require:
$\mathcal{D} = \{(< u, i, t >)_1, (< u, i, t >)_2, ...\}$: data stream
k: number of latent factors
$\lambda_{u,i,j}$: regularization factors
η: learning rate

1: **for** $< u, i, t > \in \mathcal{D}$ **do**
2: **if** $u \notin U$ **then**
3: $\mathbf{v}_u \leftarrow$ initializeVector(k)
4: $\mathbf{v}_u \sim \mathcal{N}(0, 0.1)$
5: **if** $i \notin I$ **then**
6: $\mathbf{w}_i \leftarrow$ initializeVector(k)
7: $\mathbf{w}_i \sim \mathcal{N}(0, 0.1)$
8: $j \leftarrow$ uniformSample($\{j | \langle u, j \rangle \notin \mathcal{D}\}$)
9: $\hat{r}_{uij} \leftarrow \mathbf{v}_u \cdot \mathbf{w}_i^T - \mathbf{v}_u \cdot \mathbf{w}_j^T$ // Eq.(2.28)
10: $\mathbf{v}_u \leftarrow \mathbf{v}_u + \eta \left(\frac{e^{-\hat{r}_{uij}}}{1+e^{-\hat{r}_{uij}}} \times (\mathbf{w}_i - \mathbf{w}_j) + \lambda_u \mathbf{v}_u \right)$ // Eqs.(2.29) and (2.30)
11: $\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta \left(\frac{e^{-\hat{r}_{uij}}}{1+e^{-\hat{r}_{uij}}} \times \mathbf{v}_u + \lambda_i \mathbf{w}_i \right)$ // Eqs.(2.29) and (2.30)
12: $\mathbf{w}_j \leftarrow \mathbf{w}_j + \eta \left(\frac{e^{-\hat{r}_{uij}}}{1+e^{-\hat{r}_{uij}}} \times -\mathbf{v}_u + \lambda_j \mathbf{w}_j \right)$ // Eqs.(2.29) and (2.30)

3.3 CONCEPT DRIFT

In dynamic and non-stationary settings, the assumption that data distribution is static does not hold, raising the problem of *concept drift* (Widmer and Kubat, 1996). Such problem relates to changes that occur unexpectedly over time, where the conditional probability of the output

changes given the input, but the input distribution may stay unchanged (Gama et al., 2014). Specifically, a concept drift occurs when a previously known concept change between any two timestamps. Concept drift is innate to recommender systems, as the relationships between users and items are not static and change over time (Koychev and Schwab, 2000; Koren, 2009; Gama et al., 2014; Matuszyk et al., 2018).

User dynamics include changes in preferences, which may be permanent or temporary based on some contextual influence, while item dynamics are highly influenced by popularity and user perception. Predictive performance is put at risk if a model is unable to quickly detect and adapt to drifts in previously known concepts. In other words, a model that correctly predicts user preferences at a certain time may fail to do so at a later point if it is incapable of adapting.

Matuszyk et al. (2018) argues that there are two complementary ways of adapting recommender systems to changes in previously known concepts:

1. incorporating new information into the underlying model;
2. *forgetting* obsolete information.

The first way is naturally accomplished by incremental algorithms, such as the ones previously discussed. The learning procedure updates the model based solely on the most recent observation in a stream of user feedback, and such observation is discarded after it is processed. Hence, incremental algorithms are capable of learning new concepts and have the ability to evolve over time.

Although incremental algorithms are able to process user feedback in real time, continuous incorporation of information leads to ever-growing models, which eventually causes the accumulation of obsolete and irrelevant information, and impacts accuracy and scalability. Considering obsolete information introduces noise in the learning procedure, resulting in distorted concepts, negatively affecting the recommendation process, and consequently degrading predictive capability. Also, by maintaining obsolete and irrelevant information on models while also accounting for continuously incoming observations leads to increasing time and memory requirements, eventually raising scalability issues.

In that sense, application of *forgetting mechanisms* in order to remove obsolete information from incremental algorithms can lead to improvements in accuracy and scalability (Vinagre and Jorge, 2012; Matuszyk et al., 2018), and acts as the second complementary way of adapting to changes in user and item dynamics.

3.4 FORGETTING

Forgetting mechanisms must *select* and *remove* obsolete information from the underlying recommendation model. These two processes are performed during model update, together with incremental procedure. Thus, selecting and removing information from online models incur in important design decisions, as they should avoid introduction of more complexity in the update process in order not to fall behind the data. Forgetting usually, but not necessarily, assume that recent data is more representative of users preferences and thus more relevant than older data. This assumption aligns to the benefits of considering short-term user interests.

Essentially, forgetting can be either *abrupt* or *gradual* (Gama et al., 2014; Matuszyk et al., 2018), and both can be adapted for both memory-based and matrix factorization approaches. The choice of forgetting mechanism depends on a trade-off between responsiveness to change and robustness to noise. We next define these two mechanisms.

Abrupt forgetting. Abrupt forgetting completely discards information based on their perceived relevance, measured by recency, and rely on fixed-size sliding windows. The idea is to repeatedly train the model solely with data that pertains to the window at each time step, hence being updated with new data. The main challenge is to define the appropriate window size: short windows may capture a changing distribution more quickly, but may degrade performance in periods of stability. Large windows on the other hand reacts slower to changes, but offers better predictive performance in periods of stability (Gama et al., 2014). Sliding windows can be classified into two basic types (Babcock et al., 2002): *sequence-based* windows, where the size of the window is defined according to a predefined number of observations, and *timestamp-based* windows, consisting of all observations whose timestamp are within a predetermined time interval of the current time.

Gradual forgetting. Instead of completely discarding information, gradual forgetting assigns weights to observations, or concepts, based on their relevance. By considering recent information to be more relevant than older ones, gradual forgetting is achieved by simply decreasing the importance of observations according to its age through some decay function (Vinagre and Jorge, 2012; Gama et al., 2014; Matuszyk et al., 2018). This process can be implemented by constantly decaying observations by a positive fading factor $\tau \in (0, 1)$, which controls the rate of forgetting, and ensures that older information continuously loses importance unless they are emphasized by newly generated observations.

The application of forgetting mechanisms in SBRS is shown to positively impact accuracy and scalability for both memory-based and factorization-based methods (Vinagre and Jorge, 2012; Matuszyk et al., 2018; Al-Ghossein et al., 2021). We discuss relevant forgetting contributions in the next chapter.

3.5 DISCUSSION

In this chapter, we posed the top-N recommendation problem as a data stream mining one, and consequently defined the field of stream-based recommender systems. This formulation follows the realistic assumption that in real-world systems user generated data possesses the characteristics of a data stream. Specifically, that user feedback incomes continuously at unpredictable rates and order, is potentially unbounded, new users and items enter the system at any point, and previously learned user-item dynamics change over time (Koychev and Schwab, 2000; Koychev, 2000).

The design of RS in data stream settings, however, faces several challenges, as outlined throughout the chapter. Mainly, SBRS must stay up-to-date with user generated data, ideally processing each observation faster than the rate of arrival of subsequent elements. Therefore, scalability to deal with increasing amounts of data is a fundamental requirement for data stream settings, and SBRS rely on incremental algorithms in order to address the challenges of these settings. Incremental algorithms are used to learn solely from incoming data that can then be discarded after processing, and an always-available model provides recommendations. This procedure overcomes the scalability and adaptability issues inherently related to batch-learning, as the model is always up-to-date and adapted to temporal dynamics.

In this thesis, we address recommendation in data stream settings, as it is a realistic and natural way of approaching it. Despite this, recommendation is not frequently addressed as a data stream. Thus, our contributions pertains to SBRS, more specifically on incremental memory-based models, as outlined in Chapter 5. In the next chapter, we discuss related work, some limitations of current approaches, and situate our proposals among them.

4 RELATED WORK

In this chapter we review and discuss a number of works related to the topic of Stream-Based Recommender Systems (SBRS). In Chapters 2 and 3, we mentioned categorizations of recommender systems that somehow exploit temporal effects and/or model the time dimension, such as Time-Aware Recommender Systems (Section 2.5.1), that exploits contextual information in the form of time at some stage of the prediction process, and Sequence-Aware Recommender Systems (Section 2.5.2), that learn sequential patterns from past user behavior with the goal of detecting short-term preferences and predicting the users' next interaction. Although these approaches share similarities to SBRS, mainly that they both rely on the order of interactions, they are not necessarily concerned with the following challenges, which are prerequisites of SBRS (Al-Ghossein et al., 2021):

- The recommendation model should be built on a single pass over user-generated data and updated after each received observation without access to past data;
- Observations should be processed faster than their rate of arrival and SBRS should be able to generate recommendations when required, hence both learning and recommending procedures must be done efficiently;
- Models must evolve over time, since previously learned concepts evolve and change in different ways.

Although both TARS and sequence-aware RS share some characteristics with SBRS, they assume that the whole dataset is available at any time for training with multiple passes allowed, which is prohibitive for streaming scenarios and are consequently not applicable on these scenarios. SBRS are specifically designed to operate on these restrict and specific streaming scenarios. Thus, in this chapter we review works that are design to overcome the challenges of dealing with streaming data, as discussed in Chapter 3. A detailed survey related to the exploitation of time in collaborative filtering can be found in Vinagre et al. (2015b). Surveys specific to the topics of TARS and sequence-aware RS can be found in Campos et al. (2014), Vinagre et al. (2015b) and Quadrana et al. (2018), and Wang et al. (2021), respectively.

A survey that is highly relevant and directly related to this thesis is a recent bibliography review in the context of SBRS done by Al-Ghossein et al. (2021). In their survey, the authors reviewed works that approach the recommendation problem under a data stream framework with the goal of overcoming the challenges of data stream mining discussed in Chapter 3. Based on a general schema for online adaptive learning proposed by Gama et al. (2014), Al-Ghossein et al. (2021) categorized existing work in SBRS in four modules:

- Memory module: selects data to be used for learning and also discards old data that has become irrelevant;
- Learning module: defines how models are learned and updated based on incoming data, which is mainly based on incremental learning;
- Change detection module: responsible for the active detection of drifts;
- Retrieval module: generates recommendations for users, usually by computing a relevance score for each item and selecting the N items with highest score.

The main contributions of this thesis, outlined in Chapter 5, pertains to the memory and learning modules. For that reason, we focus on these two when discussing related work. We also briefly discuss change detection contributions as they affect memory and learning modules. We do not discuss contributions from the retrieval module as such work falls outside the scope of this thesis. Besides the four aforementioned modules, Al-Ghossein et al. (2021) further categorizes SBRS work based on the used learning technique, the type of feedback and the evaluated domains. In the following sections we use the same categorization. Table 4.1, which is an adaptation and extension of their bibliography review, presents a summary of related work grouped by which module it approaches, followed by the used learning algorithm, the type of feedback used and the domain that it is explored. These contributions are discussed throughout this chapter. We split related work in five main groups, delimited by two alternating colors in Table 4.1 for ease of view:

1. Memory module (first group in gray): this group includes forgetting techniques and is discussed in Section 4.1 and further summarized in Table 4.2;

2. Learning module, specifically incremental neighborhood-based methods (first group in white): these contributions are further divided in nearest neighbors and graph-based approaches, and are discussed in Sections 4.2.1 and 4.2.1.1, respectively, and further summarized in Table 4.3;

3. Learning module, specifically incremental model-based methods (second group in gray): these contributions are divided based on the type of feedback (explicit or implicit) and strategy that they use (matrix factorization or others), and are discussed in Sections 4.2.2 and 4.2.3, and further summarized in Table 4.4;

4. Change detection module (second group in white): these contributions are discussed in Section 4.3;

5. Evaluation contributions (third group in gray): this group include contributions that raise relevant question towards evaluation of recommender systems on data stream settings, and also comparisons between incremental and batch-based algorithms. These works are discussed in Section 4.4.

We discussed in Section 2.1.1 the two types of feedback that are used by recommender systems, explicit and implicit, and how the choice of feedback directly affects the problem formulation, both learning and recommendation procedures, and consequently the entire design of a RS model. We highlight work that deal with implicit feedback as it is the type of feedback that we use in this thesis. In the following sections, we discuss subsets of related work and also present detailed summaries of such work.

4.1 MEMORY MODULE

The memory module selects the data used for learning while also discarding obsolete data that eventually becomes irrelevant. This module is fundamental given that in data streams observations are expected to be received at uncontrollable rates, and also that previously known concepts change over time and become obsolete (Domingos and Hulten, 2001; Gama et al., 2009), i.e., concept drift (Gama et al., 2014). Thus, adaptation to changes is an important aspect of online learning.

As discussed in the previous chapter, there are two complementary ways of adapting to changes in RS (Matuszyk et al., 2018): (1) incorporate new information into a model; (2)

Table 4.1: Categorization of related work. Adapted and extended from Al-Ghossein et al. (2021). **Focus**: MEM: memory module, LRN: learning module, CHG: change detection module, EVAL: evaluation contribution; **Technique**: CLST: clustering, ENS: ensemble methods, GRA: graph-based methods, KNN: neighborhood-based methods, MF: matrix factorization, NN: Neural networks, RL: reinforcement learning, SPM: stochastic process model, TREE: Context trees; **Feedback**: EXP: explicit, IMP: implicit; **Domains**: EC: e-commerce, MOV: movies, MUS: music, NEWS: news articles, POI: point of interest, VID: video, WWW: web navigation.

Paper (Year)	**Focus**	**Technique**	**Feedback**	**Domains**	**Section of discussion**
Symeonidis et al. (2020)	MEM	GRA	IMP	NEWS	
Vinagre et al. (2015a)	MEM	MF	IMP	MOV, MUS	
Nathanson et al. (2007)	MEM	CLST	EXP	WWW	
Nasraoui et al. (2007)	MEM	KNN	EXP	WWW	
Subbian et al. (2016)	MEM	KNN	EXP	EC	
Veloso et al. (2017)	MEM	MF	EXP, IMP	MOV	
Tabassum et al. (2020)	MEM	GRA	IMP	MUS	**Section 4.1**
Vinagre and Jorge (2012)	MEM	KNN	IMP	MOV, MUS	
Liu and Aberer (2014)	MEM	MF	IMP	EC, WWW	
Ding and Li (2005)	MEM	KNN	EXP	MOV	
Siddiqui et al. (2014)	MEM	KNN	EXP	MOV	
Liu et al. (2010)	MEM	KNN	EXP, IMP	MOV	
Matuszyk et al. (2018)	MEM	MF	EXP, IMP	EC, MOV, MUS	
George and Merugu (2005)	LRN	CLST, KNN	EXP	EC, MOV	
Chandramouli et al. (2011)	LRN	KNN	EXP	EC, NEWS	
Khoshneshin and Street (2010)	LRN	KNN	EXP	MOV	
Papagelis et al. (2005)	LRN	KNN	EXP	Others	
Miranda and Jorge (2009)	LRN	KNN	EXP, IMP	WWW	**Section 4.2.1**
Das et al. (2007)	LRN	KNN	IMP	NEWS	
Huang et al. (2015)	LRN	KNN	IMP	EC	
Yagci et al. (2017)	LRN	KNN	IMP	MOV, EC, NEWS	
Tofani et al. (2022)	LRN	KNN, SPM	IMP	MUS	
Jeunen et al. (2022)	LRN	KNN	IMP	EC, MOV, NEWS	

Continued on next page

Table 4.1 – continued from previous page

Paper (Year)	Focus	Technique	Feedback	Domains	Section of discussion
Backstrom and Leskovec (2011)	LRN	GRA	IMP	Others	
Cooper et al. (2014)	LRN	GRA	IMP	MOV	
Christoffel et al. (2015)	LRN	GRA	IMP	MOV	
Eksombatchai et al. (2018)	LRN	GRA	IMP	Others	
Gupta et al. (2013)	LRN	GRA	IMP	WWW	
Natarajan et al. (2013)	LRN	GRA	IMP	MUS, Others	**Section 4.2.1.1**
Paudel et al. (2017)	LRN	GRA	IMP	MOV	
Sarkar et al. (2008)	LRN	GRA	IMP	Others	
Symeonidis et al. (2020)	MEM	GRA	IMP	NEWS	
Trevisiol et al. (2014)	LRN	GRA	IMP	News	
Xiang et al. (2010)	LRN	GRA	IMP	Others	
Zhang et al. (2014a)	LRN	GRA	IMP	Others	
Agarwal et al. (2010)	LRN	MF	EXP	MOV	
Brand (2003)	LRN	MF	EXP	Others	
Devooght et al. (2015)	LRN	MF	EXP	EC, MOV	
Huang et al. (2017)	LRN	MF	EXP	MOV	
Matuszyk and Spiliopoulou (2017)	LRN	MF	EXP	EC, MOV	
Rendle and Schmidt-Thieme (2008)	LRN	MF	EXP	MOV	
Sarwar et al. (2002)	LRN	MF	EXP	MOV	
Song et al. (2015)	LRN	MF	EXP	MOV	**Sections 4.2.2, 4.2.2.1, 4.2.2.2 and 4.2.2.3**
Takács et al. (2008)	LRN	MF	EXP	MOV	
Takács et al. (2009)	LRN	MF	EXP	MOV	
Wang et al. (2013)	LRN	MF	EXP	MOV	
Wang et al. (2018b)	LRN	MF	EXP	MOV, WWW	
Yu et al. (2016)	LRN	MF	EXP	MOV, WWW	

Continued on next page

Table 4.1 – continued from previous page

Paper (Year)	Focus	Technique	Feedback	Domains	Section of discussion
Zhang et al. (2014b)	LRN	MF	EXP	Others	
Zhao et al. (2013)	LRN	MF	EXP	MOV	
Song et al. (2019)	LRN	NN	EXP	MOV	
Wang et al. (2018a)	LRN	NN	EXP	MOV	**Section 4.2.3**
Wang et al. (2014)	LRN	RL	EXP	MUS	
Chang et al. (2017)	LRN	SPM	EXP	MOV	
Anyosa et al. (2018)	LRN	MF	IMP	MUS	
Chen et al. (2013)	LRN	MF	IMP	WWW	
Diaz-Aviles et al. (2012a)	LRN	MF	IMP	WWW	
Diaz-Aviles et al. (2012b)	LRN	MF	IMP	WWW	
He et al. (2016)	LRN	MF	IMP	EC, POI	
Huang et al. (2016)	LRN	MF	IMP	VID	
Kitazawa (2016)	LRN	MF	IMP	MOV	**Sections 4.2.2.4, 4.2.2.5 and 4.2.2.6**
Pálovics et al. (2014)	LRN	MF	IMP	MUS	
Vinagre et al. (2014b)	LRN	MF	IMP	MOV, MUS	
Vinagre et al. (2015a)	LRN	MF	IMP	MOV, MUS	
Vinagre et al. (2018a)	LRN	MF	IMP	MOV, MUS	
Vinagre et al. (2018b)	LRN	MF	IMP	MOV, MUS	
Viniski et al. (2023)	CHG	MF	IMP	EC, MOV, Others	
Li et al. (2010)	LRN	RL	IMP	NEWS	
Li et al. (2016)	LRN	RL	IMP	NEWS	
Pereira et al. (2019)	LRN	RL	IMP	MUS	**Section 4.2.3**
Sanz-Cruzado et al. (2019)	LRN	RL	IMP	MOV, Others	
Çapan et al. (2022)	LRN	RL	IMP	MUS	
Garcin et al. (2013)	LRN	TREE	IMP	NEWS	
José et al. (2020)	CHG	MF	EXP, IMP	MOV, MUS	
Al-Ghossein et al. (2018a)	CHG	KNN	IMP	MOV	**Section 4.3**
Al-Ghossein et al. (2018b)	CHG	MF	IMP	MOV, NEWS	

Continued on next page

Table 4.1 – continued from previous page

Paper (Year)	Focus	Technique	Feedback	Domains	Section of discussion
Li et al. (2007)	CHG	TREE	EXP	MOV	
Viniski et al. (2021)	EVAL	MF, NN	EXP	EC	**Section 4.4**
Frigó et al. (2017)	EVAL	KNN, MF	IMP	EC, MOV, MUS, WWW	
Lommatzsch and Albayrak (2015)	EVAL	KNN, ENS	IMP	NEWS	
Vinagre et al. (2014a)	EVAL	KNN, MF	IMP	MOV, MUS	
Vinagre et al. (2021)	EVAL	MF	IMP	MOV, MUS	
Jugovac et al. (2018)	EVAL	KNN, MF, NN	IMP	NEWS	

forgetting obsolete information. The first is accomplished by incremental algorithms, resulting in models capable of evolving over time. The second way must include forgetting mechanisms in order to remove obsolete information.

Forgetting mechanisms aim to remove obsolete information from models to potentially improve accuracy and scalability (Vinagre and Jorge, 2012; Matuszyk et al., 2018), adapting to changes complementary to incremental learning. These mechanisms must select and remove obsolete information during model update while avoiding introduction of more complexity in the process in order to not fall behind the data. Forgetting usually, but not necessarily, assume that recent data is more representative of users preferences and thus more relevant than older data. Table 4.2 presents contributions related to forgetting mechanisms for neighborhood-based and matrix factorization methods.

Before we discuss these contributions, we mention two works that propose different ways to select data used for learning. Liu and Aberer (2014) propose a framework with an online component that is continuously updated, responsible for modeling short-term user preferences, and an offline component that is sporadically trained after the arrival of 10,000 new items to model long-term preferences. We note that this procedure must store the received observations in order to allow sporadic retraining, which is undesirable for streaming environments. Subbian et al. (2016) on the other hand presented a probabilistic neighborhood-based algorithm based on min-hash schemes to approximately compute the similarities between items. This way, instead of storing the full history of user interactions, only the hash functions necessary to compute the similarities are stored.

Table 4.2: Summary of related work with focus on the memory module.

Strategy	**Technique**	**Papers**	**Description**
Sliding window	KNN	(Nathanson et al., 2007; Nasraoui et al., 2007; Siddiqui et al., 2014; Vinagre and Jorge, 2012)	Considers only the most recent observations as defined by the size of the windows, which can be a fixed size or a time period.
Decay function	KNN	(Vinagre and Jorge, 2012; Tabassum et al., 2020)	Decreases similarities continuously over time as defined by a positive fading factor which are forgotten unless reinforced by new data.
Time function	KNN	(Ding and Li, 2005; Koychev, 2000; Liu et al., 2010; Symeonidis et al., 2020)	Time function are used to assign greater importance to more recent data. Old data is not removed from the model.
Rating-based	MF	(Matuszyk et al., 2018; Veloso et al., 2017)	Selects ratings to be forgotten based on the list of ratings for each user.
Latent factor-based	MF	(Vinagre et al., 2015a; Matuszyk et al., 2018)	Adjust the latent factors of users to reduce the impact of past observations.

Incremental neighborhood-based methods, discussed in Section 3.1, store similarities between users or items that are updated based on each received observation, where neighborhoods

and ranking of items are computed before each recommendation. Forgetting strategies have been devised to reduce the size of models used for neighborhood computation in order to improve recommendation times and to remove obsolete information. Such forgetting can be either *abrupt* or *gradual*.

Abrupt forgetting relies on sliding windows, defined by a number of interactions or time intervals, where only recent observations included in the windows are considered, and observations outside of the windows are forgotten. Sliding windows have been used both in user-based (Nasraoui et al., 2007; Siddiqui et al., 2014) and item-based (Nathanson et al., 2007; Vinagre and Jorge, 2012) neighborhood approaches. Regardless of the approach, performance depends largely on the size of the window. While its advantages are simplicity and straightforward application, a limitation of sliding windows is that past data is not necessarily obsolete, but such information would be abruptly forgotten based on recency as defined by the window.

An alternative approach is to gradually forget observations. *Gradual forgetting* relies on mechanisms to weight observations based on recency, by considering recent observations to be more important than older ones. Koychev (2000) proposes a technique to assign higher weights to more recent observations in a content-based method, thus gradually decreasing the importance of observations over time. Such a technique results in faster adaptations to new user interests.

Similar approaches were explored in Ding and Li (2005) and Liu et al. (2010). In Ding and Li (2005), a item-based CF algorithm is extended to predict new ratings based on time-weighted ratings. A time function is used to assign greater importance to recent data and less relevance to older data. Liu et al. (2010) proposed an online evolutionary CF framework based on an incremental item-based nearest neighbors method that also considers temporal relevance of ratings when generating recommendations, where a weighting function is used to increase the similarity of items that are rated around the same time. In Symeonidis et al. (2020) a Sigmoid-based function is used to assign weights to items according to their position in a session, such that recent items are given greater relevance. In general, generation of recommendations based on recent information results in accuracy improvements.

Vinagre and Jorge (2012) extends nearest neighbors methods using a decay function in the similarity calculation, causing older items to lose relevance. Similarity matrices are multiplied by a positive fading factor $\tau < 1$ before each update based on new information, where τ controls the forgetting rate. This causes similarities to decrease continuously over time unless reinforced by newly generated data. When the similarity values reach a low threshold value, it is assumed to be zero, which reduces the model size and consequently improves scalability as a result of lower computational requirements. The advantage of this method is its simplicity and application in a single scan, while a shortcoming is reduced effectiveness in the presence of subtle local changes, since forgetting is applied globally.

In the context of recurring link prediction in graph streams, Tabassum et al. (2020) proposes a forgetting function that weights the links in a stream exponentially based on frequency and recency. A exponential function is applied on every unique edge on the stream at predefined time intervals, where the weight of the edge at a previous timestamp is multiplied by $(1 - \tau)$. Parameter $\tau \in [0, 1]$ defines a bias rate, where higher values of τ tends to forget previous edge occurrences, biasing the weights according to their recency. The technique also includes a threshold parameter that prunes edges based on their weights at a given time interval, where higher values retains only strong and stable edges. This technique improved the performance of recurring link prediction in the context of recommendation.

Incremental matrix factorization methods, discussed in Section 3.2, maps users and items in a common latent feature space of low dimensionality, such that the affinity between users and items is given by the inner product of their embedded vectors. Hence, a MF method must

learn these representations. Koren (2009) showed that user behavior presents temporal patterns, and modeling temporal information in user latent vectors improves significantly the predictive capabilities of MF methods.

Forgetting for incremental MF was first studied in Matuszyk and Spiliopoulou (2014); Matuszyk et al. (2018). Matuszyk et al. (2018) proposes several forgetting strategies to select obsolete information and remove their effect from the model, giving more importance to more representative observations. These strategies are divided into two categories, *rating-based* and *latent factor-based* forgetting. Rating-based strategies operates directly on the list of ratings for each user, selecting and discarding ratings from these lists through sliding windows, using a fixed size or time frame, or sensitivity analysis, e.g., by removing ratings that causes dramatic changes in the user latent vector that are then not used to update the latent vectors.

Latent factor-based forgetting strategies adjust the latent factors of users to reduce the impact of past observations, and are deployed for each incoming rating in a stream. These strategies include user fading factors, that reduces the importance of past preferences based on volatility and frequency, and forgetting popular or unpopular items. Experiments suggest that latent factor-based forgetting, particularly user fading factor and forgetting unpopular items, are successful both in predictive power and computation time.

Veloso et al. (2017) proposes forgetting techniques based on individual, fixed size first in, first out (FIFO) queues, where for each user a queue containing their last n ratings is maintained. With the arrival of new user ratings, these ratings are inserted in the user queue, the queue is shifted and the user latent vector is faded according to the ratings on the queue. Fading is performed based on four forgetting strategies: two time-based functions, that fade the ratings according to the timestamps of both the faded rating and the current rating, and two positional-based functions, that take into account the position of the rating on the queue. These strategies improve accuracy when compared to the forget unpopular technique proposed in Matuszyk et al. (2018).

Finally, a FIFO queue is also used in Vinagre et al. (2015a). In their work, a global queue of all items seen in the stream, ordered based on the recency of item occurrences, is deployed. For every new observation in the stream, a few older items at the head of the queue are selected to be used as negative feedback to the current user, where only the user latent factor is adjusted, before updating both the user and item latent factors with the new observation. The items selected to be used as negative feedback are then reinserted at the tail of the queue. This procedure, which gives greater importance to more recent events, is shown to improve accuracy when compared to an incremental MF algorithm.

4.2 LEARNING MODULE

The learning module is responsible for defining how models are updated with incoming data, which usually is done through incremental learning, as described in Chapter 3. In this thesis we aim to design models capable of updating itself with each incoming observation, without storing observations and also without the need for retraining, which represents a very strict learning scenario, with tight time and processing requirements. In the following, we describe contributions that generate recommendations considering a streaming setting, i.e., models that recommend based on up-to-date information, and that are updated incrementally based on a previously trained model. First we discuss incremental neighborhood-based (or memory-based) models, followed by incremental matrix factorization ones.

4.2.1 Incremental neighborhood-based approaches

Table 4.3 presents contributions related to incremental neighborhood-based methods. These models (Section 3.1) were first proposed in Papagelis et al. (2005), where an incremental user-based CF algorithm for explicit feedback was presented. Similarities are stored and incrementally updated for each new observation, which allows for high scalability in comparison to previous CF approaches. The recommendation procedure consists of computing the nearest neighbors of the active user and then predicting ratings for all candidate items using Pearson correlation (Eq. 2.3).

Following the work of Papagelis et al. (2005), Miranda and Jorge (2009) proposed incremental user-based and item-based algorithms designed to learn from implicit feedback using cosine similarities instead of Pearson correlation. Such change affects the similarity formulation in a implicit feedback setting, which can be simplify to computing user and item co-occurrence counts, as defined in Eqs. (2.8) and (3.1). Two users co-occur when they both interact with the same item, while two items are said to co-occur when a given user interacts with both of them. Analogous to Papagelis et al. (2005), the rating matrix and the similarities are stored and incrementally updated for each new observation. With co-occurrence counts, an update is performed simply by incrementing these counters. Nearest neighbors computation and the prediction of ratings are also performed at the time of recommendation.

Recommendations based on co-occurrence counts were also explored in Das et al. (2007), Yagci et al. (2017), Tofani et al. (2022) and Jeunen et al. (2022). Das et al. (2007) proposed one of the first approaches to deal with online news recommendation in real time, that combines two clustering approaches with a covisitation counts algorithm. The covisitation algorithm is implemented as a graph, such that nodes represent items and edges represent covisitation of items weighted according to time. For a given candidate item s, its near neighbors are the set of items that have been covisited with it, weighted by the age discounted count of how often they were covisited.

Yagci et al. (2017) proposed an item-based algorithm similar to the works of Deshpande and Karypis (2004) and Miranda and Jorge (2009), that approximates frequent item co-occurrence counts in order to avoid storing both the rating matrix $\mathbf{R}$ and item similarity matrix $\mathbf{S}$, as done in Miranda and Jorge (2009). Their approach stores a list UL of size $|\mathcal{U}|$ containing the item interaction history of each user, and a list IL of size $|\mathcal{I}|$ that contains l counters for every item to hold its frequency co-occurrences with other items, where $l \ll |\mathcal{I}|$, which significantly improves space in comparison to storing a $|\mathcal{I}| \times |\mathcal{I}|$ matrix in memory. Lists UL and IL are implemented through the use of efficient data structures based on hash tables and linked lists. For every new observation in the stream, these lists are updated and a hash function is used to sample the item co-occurrences in the stream and update the counters if required. Candidate items are ranked based on cosine similarities calculated on these approximated co-occurrence counts.

Tofani et al. (2022) proposes three dynamic approaches for session-based music recommendation using information retrieval techniques, two of them based on TF-IDF weighting, one of which (IR-1NN) expands the nearest neighbors framework. A third method (IR-MC) extends the first-order Markov chain, a frequency count-based approach, in order to consider longer past sequences. Algorithm IR-1NN is a slight variation of session-based KNN (Ludewig and Jannach, 2018) that considers only items that belong to the top-1 nearest neighbor, and these items are ranked using a proximity ranking method that accounts for the overall proximity of candidate items in sessions to items in the query. Algorithm IR-MC extends first-order Markov chains by allowing it to consider the correspondence of past items in the context of previous sessions, by exponentially rewarding candidate items according to the size of the shared sequence of past items each session has in common with the current query.

Table 4.3: Summary of incremental neighborhood-based contributions.

Strategy	Technique	Papers	Description
User-based CF	KNN	(Papagelis et al., 2005; Miranda and Jorge, 2009)	Similarities are stored and incrementally updated for each new observation. Scores are computed with Pearson correlation and cosine similarity.
Item-based CF	KNN	(Miranda and Jorge, 2009; Chandramouli et al., 2011; Huang et al., 2015; Tofani et al., 2022)	Similarities are stored and incrementally updated for each new observation. Scores are computed with cosine similarity. For implicit feedback, these are simplified to co-occurrence counts between items.
	KNN with approximate similarity	(Yagci et al., 2017)	Frequency co-occurrence counters are approximated with hash functions and specific data structures to improve memory requirements. Scores are computed with cosine similarity.
	Co-occurrence counts	(Das et al., 2007)	Recommendations based on co-occurrence counts stored in item-based graphs that are updated for each new observation.
	Embarrassingly Shallow Auto-Encoder	(Jeunen et al., 2022)	Incremental updates on EASE^R, a linear item-based model that consists in computing the inverse of a given Gramian item-item matrix that holds the co-occurrence counts of items.
Graph-based CF	PPR	(Xiang et al., 2010; Backstrom and Leskovec, 2011; Gupta et al., 2013; Natarajan et al., 2013)	Models are built in an offline phase, and at recommendation time PPR is computed to rank items. These models include user-based (Backstrom and Leskovec, 2011; Gupta et al., 2013), item-based (Natarajan et al., 2013) and bipartite graphs (Xiang et al., 2010).

Continued on next page

Table 4.3 – continued from previous page

Strategy	Technique	Papers	Description
	Hitting time	(Sarkar et al., 2008)	Hitting time is estimated through short T-steps random walks, used to rank items on an heterogeneous graph.
	Co-occurrence counts	(Trevisiol et al., 2014)	Recommendations based on co-occurrence counts stored in item-based graphs that are updated for each new observation.
	Markov chains	(Zhang et al., 2014a)	N-th order Markov chains are used to recommend items computed on an item-based graph that is incrementally updated for new observations.
	KNN	(Cooper et al., 2014; Christoffel et al., 2015; Paudel et al., 2017)	Estimates the probability distribution matrix after T-steps (Eq. (2.11)) on bipartite graphs with random walk sampling.
	RWR	(Eksombatchai et al., 2018; Symeonidis et al., 2020)	Models are built in an offline phase, updated incrementally, and at recommendation time RWR is computed to rank items. These models include bipartite (Eksombatchai et al., 2018) and heterogeneous graphs (Symeonidis et al., 2020).
Other	Co-clustering	(George and Merugu, 2005; Khoshneshin and Street, 2010)	Users and items are clustered in an offline phase, these clusters are sporadically updated and are used for recommendation along with biases of users and items.

Jeunen et al. (2022) proposes Dynamic EASE^R, a method to incrementally update models that are based on the Embarrassingly Shallow Auto-Encoder (EASE^R) (Steck, 2019). EASE^R is a linear item-based model that consists in computing the inverse of a given Gramian item-item matrix that holds the co-occurrence counts of items. Despite its conceptual simplicity, EASE^R is capable of outperforming state-of-the-art approaches in several ranking tasks, while being computationally efficient in some settings as it depends only on the size of the item catalog. Considering dynamic scenarios, it becomes unfeasible to compute the inversion of the entire item-item matrix with every update. To avoid such computation, Dynamic EASE^R incrementally updates an existing EASE^R model when new data arrives, by exploiting the assumption of low-rank for the user-item interaction data. This allows the algorithm to target only parts of the resulting model that requires updating.

Efficient frameworks based on the aforementioned item-based CF approaches for data stream processing are described in Chandramouli et al. (2011) and Huang et al. (2015). Chandramouli et al. (2011) describes StreamRec, an implementation of an item-based CF approach for explicit feedback on a stream processing system that is deployed for the recommendation of news and movies in real-time. Huang et al. (2015) proposed TencentRec, a framework designed to deal with implicit feedback that implements an algorithm similar to the proposal of Miranda and Jorge (2009) on Storm[1]. An important feature of TencentRec is the inclusion of real-time pruning, done in order to reduce the computation costs of model update and recommendation generation, where dissimilar items that likely will not be part of the set of top candidates for recommendations are discarded.

We also note two neighborhood-based methods based on co-clustering. George and Merugu (2005) designed incremental and parallel versions of a co-clustering algorithm that simultaneously groups users and items. Predictions are made based on the average ratings of the co-clusters, which represents the neighborhoods of users and items, and also on biases of each user and item. Their approach requires an initial offline training that clusters users and items, that is then sporadically updated based on new data. To account for new users and items between updates, these are assigned to a global cluster before the next update. Several improvements to these methods are suggested in Khoshneshin and Street (2010), where an evolutionary co-clustering approach was proposed to overcome some accuracy limitations.

4.2.1.1 Graph-based approaches

In Section 2.3.1.1, we discussed the advantages of representing neighborhoods of users and items with graph-based models. Mainly, that they present reduced sensitivity to data sparsity, associated with structural transitivity, which allows similarities to be computed based on indirect connections. In these methods, nodes that are not directly connected are allowed to influence each other by propagating information through the edges of the graph, with their weights indicating the amount of information that is allowed to pass through, and the influence of nodes that are closer to a source node should be higher than the influence of nodes that are further away in the graph.

Personalized PageRank (PPR) (Page et al., 1999; Haveliwala, 2003) (Eqs. (2.14) and (2.15)) is a popular algorithm to rank nodes in a graph, and adaptations of it have been devised for recommendation tasks (Gori and Pucci, 2007; Baluja et al., 2008; Vahedian et al., 2017; Nikolakopoulos and Karypis, 2019). However, the application of PPR in data stream scenarios is limited given scalability issues raised with the impracticability of constantly recomputing the stationary distribution (Eq. (2.14)) on ever-changing graphs.

[1]https://storm.apache.org/

An alternative approach to overcome this issue is to estimate the stationary distribution with approximation algorithms at the expense of accuracy (Fogaras et al., 2005; Avrachenkov et al., 2007; Bahmani et al., 2010; Lofgren et al., 2014; Ohsaka et al., 2015), which has been shown to be effective in several contexts. In Backstrom and Leskovec (2011) and Gupta et al. (2013), efficient implementations of recommendations based on PPR and random walk with restarts computed on large bipartite graphs, representing relationships between friends in social media, are described. These approaches are shown to outperform other machine learning techniques.

In Sarkar et al. (2008), the notion of truncated hitting time is introduced, which is an approximation of the hitting time (Eq. (2.17)) with random walks limited to t steps, which reduces the sensitivity of the algorithm to long range paths. The algorithm is scalable and accurate, and it is shown to perform well in ranking tasks.

Cooper et al. (2014) proposed three ranking methods on a bipartite graph based on the t-step probability matrix (Eq.(2.11)): $\mathbf{P}^3$, $\mathbf{P}^5$ and $\mathbf{P}^3_\alpha$, where $\mathbf{P}^3_\alpha$ (Section 2.3.1.1) raises the transition probabilities to the power of a parameter α. The paper also provides approximations obtained with random walk sampling, and show that algorithm $\mathbf{P}^3_\alpha$ outperforms ItemRank and the methods proposed by Fouss et al. (2007) in accuracy and scalability.

Christoffel et al. (2015) proposes algorithm $\mathbf{RP}^3_\beta$ (Section 2.3.1.1), a re-ranking of P^3 that penalizes items with high degree in order to recommend long-tail items and increase diversity. The paper also describes a sampling procedure for algorithms $\mathbf{P}^3_\alpha$ and $\mathbf{RP}^3_\beta$, and shows that $\mathbf{RP}^3_\beta$ outperforms $\mathbf{P}^3_\alpha$, although it requires more samples to converge. An extended version of their algorithm is presented in Paudel et al. (2017), where a procedure to cache a variable number of walks from each node is proposed. The idea is to induce t-step random walks based on cached partial walks. Such procedure reduces the computational cost of sampling random walks, improving scalability, at the expense of memory.

The methods described thus far are effective in ranking vertices and modeling long-term user preference profiles. However, user preferences change over time and recommendations should consider short-term preferences in order to stay up-to-date. We next discuss methods that consider short-term information.

Xiang et al. (2010) incorporated short-term and long-term user preferences into a bipartite graph, where nodes represents users, items and sessions, and edges balance the influence of short-term and long-term preferences. User and item nodes are connected based on past user interactions, representing long-term preferences. Item and user-session nodes are connected based on user interactions in a time window, representing short-term interests. Recommendations are generated through random walks in the graph with PPR, and the impact of short-term and long-term preferences can be balanced parametrically.

In the context of next-app recommendation, Natarajan et al. (2013) proposed a method that predicts sequential actions of users based on behavioral graphs. In an offline step, users with similar sequential behavior, measured by first-order Markov chains between items, are clustered together with k-means algorithm, resulting in several Markov graphs. Transition probabilities are computed with the use of PPR on each of the resulting Markov graphs. To generate a recommendation to a given user, she is first mapped to the corresponding cluster, and recommendations are generated based on the probabilities computed by PPR.

Trevisiol et al. (2014) addresses cold-start issues in news recommendation domain by ranking articles based on two users' browsing graphs: BrowseGraph, which collects all user browsing behavior, and ReferrerGraph, which is a subgraph of the BrowseGraph induced by user sessions with the same referrer domain. To predict the next page to a newcoming user, the neighbors of both graphs are considered as candidates, and four simple strategies to select

the next page are used: random, content-based, most popular and edge-weight-based, with the edge-weight-based approach, which is similar to a co-occurrence count, obtaining the best results.

For location recommendations, Zhang et al. (2014a) proposed to incorporate sequential patterns from users' check-in behaviors in a location-location transition graph in incremental manner, where nodes represent locations, edges represent transitions between locations, and edge weights are based on transitions count. The proposed method, LORE, recommends new locations based on an additive n-th order Markov chain deduced through all visited locations of a given user, which reduces the impact of the data sparsity problem.

In Eksombatchai et al. (2018), sampling of short random walks in bipartite graphs are used for recommendation in a large-scale graph. The random walks are biased based on user features, such that walks traverse edges that are more relevant to the user. Recommendations are generated dynamically after each user interaction, and more relevance is given to the most recently interacted items. Their proposal also includes a strategy for early stopping, which terminates the random walk sampling when a given number of nodes have been visited by a predefined number of walks. The experimental results show that early stopping speeds up the similarity computation by a factor of two with only minor losses in accuracy.

Symeonidis et al. (2020) exploits random walks on a time-evolving heterogeneous information network for news recommendations. The information network is composed of five entities: users, sessions, articles, categories and article locations, and random walk with restarts are used to infer similarities between entities. The authors use a sliding time window in order to forget obsolete articles, and test five different article weighting strategies to generate recommendations, and show that the choice of strategy leads to different results in terms of accuracy and diversity, with the best accuracy usually obtained by considering mainly the most recently clicked item in the session.

4.2.2 IMF approaches

The success of model-based approaches in rating prediction tasks (Koren, 2009), specifically matrix factorization ones, has motivated several contributions to incrementally update MF models to allow its application in online scenarios. Table 4.4 presents a summary of such contributions, grouped by the manner in which updates are performed. The first incremental MF contributions were based on SVD, defined in Section 2.3.2.1 (Sarwar et al., 2002; Brand, 2003).

Sarwar et al. (2002) proposes incremental updates by computing latent vectors of new users and new items based on the current decomposition, and appending them to their corresponding matrices through the application of the fold-in method (Deerwester et al., 1990). Another contribution from Brand (2003) explores algebraic properties of SVD and defines sequential update rules that allow addition, update and removal of data embedded in the current decomposition. Despite the effectiveness of the aforementioned contributions, SVD has important accuracy and scalability shortcomings, mainly that it requires an initial dense matrix that relies on value imputation (Section 2.3.2.1).

4.2.2.1 Learning from explicit feedback

Analogously to the advances in MF for rating prediction, succeeding approaches adapted the paradigm of minimizing the regularized squared error for known ratings, as defined in Eq.(2.22), through SGD and ALS to the online setting. Rendle and Schmidt-Thieme (2008) proposed an online SGD algorithm that trains the latent factors of new users and items based on new ratings while it maintains the remaining latent factors fixed. Takács et al. (2008, 2009) proposed incremental updates on a SGD model with algorithm Biased Regularized Incremental

Table 4.4: Summary of incremental matrix factorization contributions.

Feedback	Strategy	Technique	Papers	Description
Explicit	Periodical incremental updates on one side of existing model when sufficient data is collected + periodical retraining	SGD	(Takács et al., 2008, 2009; Zhao et al., 2013; Wang et al., 2013; Matuszyk and Spiliopoulou, 2017)	Initial model is trained in batch. One side of the latent factors (users or items) is updated when new ratings are made available, while the other side is kept fixed and in some cases periodically retrained.
	Periodical incremental updates on one side of existing model + periodical retraining	ALS	(Yu et al., 2016)	Initial model is trained in batch. One side of the latent factors (users or items) is updated when new ratings are made available, while the other side is kept fixed. Requires periodical retraining from scratch.
	Periodical incremental updates on both sides of existing model + periodical retraining	SGD	(Rendle and Schmidt-Thieme, 2008; Agarwal et al., 2010; Wang et al., 2013; Song et al., 2015; Devooght et al., 2015)	Initial model is trained in batch. Both sides of the latent factors (users and items) are updated when new ratings are made available. Model can be periodically retrained.
		SVD	(Sarwar et al., 2002; Brand, 2003; Zhang et al., 2014b)	Updates and adds latent factors of users and items based on the current decomposition. Requires initial dense matrix.
		Continuous Markov processes	(Chang et al., 2017)	Continuous Markov processes model time-varying user and item factors. Relies on robust offline parameter estimation that builds an initial model from historical data. Online phase predicts unknown ratings based on a variational Bayesian method.

Continued on next page

Table 4.4 – continued from previous page

Feedback	Strategy	Technique	Papers	Description
		Multi-armed bandits	(Wang et al., 2014)	Represent each item on the dataset as an arm to be pulled. Goal is to learn a strategy that maximizes user satisfaction with each recommendation (exploitation) while also gaining knowledge about their tastes (exploration). Strategy is updated iteratively based on available feedback. Uses content-based information as features that are kept static.
		Neural networks	(Wang et al., 2018a; Song et al., 2019)	Leverages neural networks to capture long-term user preferences that varies slowly over time. Relies on parameter pre-training and requires expensive update procedures.
	Periodical incremental updates when sufficient data is collected based on reservoirs	SGD	(Diaz-Aviles et al., 2012a,b; Huang et al., 2017; Wang et al., 2018b)	Sample of dataset is maintained in a reservoir. Model updates are periodically performed based on the reservoir.
	Incremental updates for each incoming observation + reservoir	SGD	(Chen et al., 2013)	Sample of dataset is maintained in a reservoir. Model updates are performed incrementally, based on each new incoming observation and with positive and negative samples stored on the reservoir.
Implicit	Incremental updates for each incoming observation	SGD	(Vinagre et al., 2014b; Pálovics et al., 2014; Vinagre et al., 2015a; Huang et al., 2016; Anyosa et al., 2018; Viniski et al., 2023)	Model updates are made incrementally after the arrival of each incoming observation in a data stream. Latent factors of new users and new items are randomly initialized and appended into the model, dispensing the need for complete retraining.

Continued on next page

Table 4.4 – continued from previous page

Feedback	Strategy	Technique	Papers	Description
		ALS	(He et al., 2016)	Model updates are made incrementally after the arrival of each incoming observation in a data stream, by only updating the corresponding user and item latent factors until convergence. Requires optimizing each coordinate of the latent vector while keeping the remaining ones fixed.
		Ensemble methods	(Vinagre et al., 2018a,b)	Combines several models that are incrementally updated after the arrival of each incoming observation in a data stream. Generally improves accuracy at expense of scalability.
		Factorization machines	(Kitazawa, 2016)	Model updates are made incrementally after the arrival of each incoming observation in a data stream in a single step.
	Periodical incremental updates when sufficient data is collected	Context trees	(Garcin et al., 2013)	Context trees model items and sequences of items. Set of prediction models is associated with each context. Tree is dynamically updated based on active pool of items that changes periodically.
		Multi-armed bandits	(Li et al., 2010, 2016; Pereira et al., 2019; Sanz-Cruzado et al., 2019; Çapan et al., 2022)	Represent each user/item on the dataset as an arm to be pulled. Goal is to learn a strategy that maximizes user satisfaction with each recommendation (exploitation) while also gaining knowledge about their tastes (exploration). Strategy is updated iteratively based on available feedback.

Simultaneous MF (BRISMF). BRISMF is similar to a batch-based SGD with the addition of user and item biases, used in order to capture how the rating scale is used by users to evaluate items. After initially training BRISMF in batch, the algorithm retrains the user latent vectors every time new ratings are made available to each active user, while item latent vectors are kept fixed.

Zhao et al. (2013) extends the Probabilistic Matrix Factorization (PMF) model (Mnih and Salakhutdinov, 2007) to continuously recommend items to users based on up-to-date information, where interactive feedback is immediately inserted into the model so that the recommendations can be adjusted accordingly. In their proposal, PMF is used to capture the distributions of user and item latent factors, and in order to account for drifts in user interests and user cold-start, their proposal leverage exploitation-exploration algorithms, such as Thompson sampling (Chapelle and Li, 2011), to allow continuous learning and detection of drifts in user profile while also attempting to maximize their satisfaction. The proposed framework updates the corresponding user latent factor for every new observation, while keeping the item factors fixed, only retraining them periodically. Although incremental, the aforementioned approaches require computation of initial models in batch. Thus, retraining is necessary to include new latent factors into the model, or to update a side of the model that was kept fixed, which in turn requires storing the rating matrix.

As opposed to incrementally updating a single vector based on each incoming observation, Wang et al. (2013) proposed an framework that updates additional user vectors besides the one directly affected by the newly generated observation, and also the affected item latent factor. The framework exploits multi-task learning ideas (Phuong and Phuong, 2008), by incorporating a measure of similarity between users in the user interaction matrix. Then, in addition to incrementally adjusting the latent factor of the active user (affected by the current rating), the framework treats latent factors of users similar to the active one as tasks to also be optimized. Such extension comes at the cost of being less scalable, as the learning procedure has higher time complexity, but results in accuracy improvements in comparison to updating only the user and item latent factors directly related to the current observation.

Song et al. (2015) proposes an approach that re-learns the feature space, with auxiliary feature learning and matrix sketching strategies. The prediction error of new data is normalized and added to the existing feature matrices as auxiliary features. As such operation results in continuously increasing the number of dimensions of said matrices, sketches of lower dimension of these matrices are maintained. This approach is specifically designed to handle new users and new items, and their approach outperforms the one by Rendle and Schmidt-Thieme (2008) in these scenarios. Conversely, Agarwal et al. (2010) proposes to update user and item feature spaces through online bilinear regression. Such proposal relies on a predictive model built offline based on historical data, which is assumed to be comprehensive enough, that is then used to initialize online models and to learn linear projections with the goal of reducing dimensionality, which improves scalability.

Yu et al. (2016) proposes an incremental approach for MF models based on ALS, which is another popular algorithm for updating latent factors (Section 2.3.2.1). Their approach, termed one-sided least squares, consists in updating only one side of an existing model, i.e., either the user or item side is updated, such that only the latent factor that is affected by a new rating is updated. When a new user arrives, latent factors for items are fixed, and only the latent vector of given user is updated. Similarly, when a new item arrives, latent factors for users are fixed, and only the latent vector of given item is updated. Experimental results show that one-sided least squares obtains similar accuracy to ALS trained from scratch, assuming that there are enough ratings for incremental learning, but with reduced learning time. A limitation, however, is that it does not dispense the need for retraining. Instead, when the amount of incremental data reaches

a certain threshold, the model is retrained from scratch, which is still undesirable in online scenarios. The authors also describe a parallel implementation of one-sided ALS, which is able to significantly reduce training costs. Although highly parallel, the computational complexity of ALS is higher than SGD.

Matuszyk and Spiliopoulou (2017) proposed a semi-supervised framework that extends BRISMF (Takács et al., 2009) to alleviate sparsity problems in recommendation in streaming scenarios. Their approach consists in a model, composed of several co-trainers, initially built in batch and then updated incrementally for each observation generated in the stream. In the initial training in batch, several co-trainers, all based on BRISMF, are built on different parts of the training set, so that each co-trainer is specialized on a specific portion of the data and can therefore teach other trainers. After the initial co-trainers are built, they operate in streaming mode, where each co-trainer is updated continuously in supervised manner for each new rating from the stream. Sporadically, unsupervised learning takes place, where several unlabelled instances are selected, and predictions for these instances are made by the co-trainers. Based on a measure that assesses how reliable is the prediction of each trainer, the model selects reliable learners that provide labels for these instances that are then used to train unreliable ones in incremental manner. Obtained results show that co-training with three learners significantly outperformed supervised learning only, but that the number of co-trainers is strongly limited due to computational requirements.

4.2.2.2 Tensor factorization approaches

Tensor factorization is one of the most successful approaches for context-aware recommendation (Karatzoglou et al., 2010; Rendle and Schmidt-Thieme, 2010) (Section 2.5). Tensor factorization is similar to traditional MF. However, instead of factorizing a matrix, it factorizes a multidimensional tensor that includes extra dimensions, which represents contextual information, such as time features. Zhang et al. (2014b) proposed an incremental tensor factorization approach for web service recommendation. The approach uses three-dimensional tensors that represent users, services and time dimensions, in order to capture the triadic relation between them, such that tensors are updated incrementally following the incremental SVD procedure proposed by Sarwar et al. (2002). Their results suggest that a higher number of latent factors increases predictive capability, but at the expense of longer computation time and storage space, which is unsuitable for streaming environments.

4.2.2.3 Exploiting information from missing observations

One important consideration in rating prediction is how to handle missing observations, as the rating matrix is typically very sparse. Devooght et al. (2015) argues that the assumption that the distribution of observed ratings, used to built a prediction model, is representative of the distribution of unknown ones does not hold in most real-world scenarios. In turn, the authors suggest that it is likely for unknown items to be weakly rated, as users tend to rate a limited number of items that better reflect their interests. Thus, the authors extend several loss functions to account for explicit priors on unknown values, by adding an additional term related to missing ratings in the objective function. Online updates are performed in a single step, where only the affected (current) user and item latent factors are updated with a gradient step.

4.2.2.4 *Learning from implicit feedback*

IMF methods were also proposed for the more realistic online setting with implicit feedback, where the task becomes a top-N recommendation problem, i.e., a ranking task that consists in recommending N relevant items to the active user. Vinagre et al. (2014b) proposed an incremental version of the SGD method (ISGD) for positive-only feedback, which updates the model based solely on the current observation in a data stream. Latent factors of new users and new items are randomly initialized and appended into the model, that is updated for each incoming observation $\langle u, i, t \rangle$ by adjusting latent factors $\mathbf{p}_u$ and $\mathbf{q}_i$ on a single pass. This procedure dispenses the need for complete retraining as is done in previously mentioned approaches. ISGD is shown to be very scalable, with competitive accuracy in several datasets.

A limitation of ISGD is that it converges globally to the positive class, given the absence of negative examples, which eventually causes degradation in accuracy. To overcome this issue, a follow-up study (Vinagre et al., 2015a) proposed a recency-based scheme to perform negative preference imputation into ISGD (RAISGD), as discussed in Section 4.1. RAISGD selects infrequent items seen in the stream to be used as negative feedback to the active user, avoiding convergence to the positive class. Experiments suggest that RAISGD outperforms ISGD with minor overhead in scalability.

Another contribution that aims to address the challenge of absent negative feedback is presented in He et al. (2016). Instead of uniformly weighting missing data, as suggested by Devooght et al. (2015), which in turns assumes that missing entries are equally likely to be negative feedback, the authors propose to assign weights of missing data based on the popularity of items. Such a strategy assumes that popular items that are constantly recommended to a given user and are often ignored are more likely to be irrelevant, as opposed to unknown, by said user. The objective function is extended to account for this effect, by introducing a confidence factor that is parameterized based on item's popularity. The authors then propose a element-wise ALS procedure to reduce learning complexity and support online learning. This procedure optimizes parameters at element level, i.e., optimizes each coordinate of the latent vector while keeping the remaining ones fixed. An optimization procedure is proposed to avoid repeated computations when updating latent factors for different users by pre-computing and storing some commonly used terms. Incremental updates for new incoming observations can be performed by only updating the corresponding user and item latent factors until convergence.

Attempts to improve the accuracy performance of ISGD with ensemble methods were also explored. Ensemble methods attempt to obtain better predictive performance by combining multiple learning algorithms or models with different techniques (Dietterich, 2000). In this context, bagging (Vinagre et al., 2018a) and boosting (Vinagre et al., 2018b) techniques were studied. Bagging trains several sub-models, each model on a generated bootstrap sample of the original dataset. Boosting on the other hand trains several base models where each learner (weak learner) attempts to correct the deficiencies of the previous one. The prediction phase then aggregates the scores provided by the various sub-models.

To evaluate bagging in online settings, Vinagre et al. (2018a) trains a given number of ISGD models that are aggregated to generated recommendations. Considering a continuous flow of data, each incoming observation in the stream is used to train a subset of ISGD models, selecting models following a Poisson distribution, which allows the usage of bagging in a single pass over data and guarantees that each model is trained on a bootstrap sample of the dataset. Results show that bagging improves accuracy with manageable overheads, and that the optimal number of models depends on the desired trade-off between accuracy and computational cost.

A following study by Vinagre et al. (2018b) analyzes the impact of boosting in online recommendation with two proposed algorithms, also with ISGD as its base learner. Both

algorithms maintain M weak models, where the first model learns the value of the target and passes the residual (outcome of loss function) to the next model in order for it to learn the residual of the previous one, such that this process is repeated until the end of the iteration. Results suggest that accuracy can be improved with a small number of base models.

To account for changes in users preferences and items dynamics, which do not occur in the entire dataset and only pertains to a few users or items, Viniski et al. (2023) uses specialized parameters to adjust the learning rate for each user or item according to the observed performance of the recommender, and proposes several specialized extensions of adaptive gradient descent-based algorithms for IMF methods. Their proposal updates latent factors of users and items considering a specific optimizer term learned for the current user or item in the data stream. Results show that their proposed user-specialized variants outperforms ISGD and suggest that they are well suited for scenarios where fewer interactions per user are available, at the expense of an increased number of model parameters to store and update.

4.2.2.5 Usage of auxiliary data

Another extension to ISGD and RAISGD was explored in Anyosa et al. (2018), where an incremental matrix co-factorization algorithm, CORAISGD, was proposed. CORAISGD includes additional dimensions to be decomposed in the common latent factor space to consider auxiliary data from users and items, as opposed to traditional ISGD that relies solely on the user-item interactions. CORAISGD jointly factorizes users, items, user features and item features, where for each additional relation user/item-feature, a new matrix is added alongside the feedback matrix. CORAISGD is evaluated on music domain datasets, with one additional dimension related to items (songs), that represents artists of such items. Results show that CORAISGD significantly outperforms RAISGD, at the expense of higher update times.

Pálovics et al. (2014) investigate the impact of social influence between users in a music-based social network to improve recommendations. The key concept is to recommend to a given user artists that a similar friend has recently listened to. Their proposed matrix factorization model includes the influence of similar users bounded by a predefined time frame that is trained by a single SGD procedure. Results show that their proposed method outperforms static baseline recommenders.

Kitazawa (2016), explored additional user and item features through incremental factorization machines for online recommendation. Following the proposal of Vinagre et al. (2014b), the author extends factorization machines (Rendle, 2012), by updating its parameters for each received observation in a single step. Regularization parameters are also updated incrementally, using the most recent observation as a validation sample. The proposal is evaluated in a small movie-based domain dataset, and exploits features such as user demographics, movie genre, timestamp of provided rating and time elapsed between current and last observation by the user. Results show that the usage of additional information increases performance in comparison to ISGD and its batch-based counterparts, but with significant increase in recommendation time.

Huang et al. (2016) describes a recommender system used in production for a large-scale video recommendation service, Tencent Video, that has over 10 million active users and generates about one billion user interactions daily. Such system combines an incremental matrix factorization approach with additional features like video type and time factors to compute similarities between items (videos). The incremental matrix factorization method is similar to ISGD: an incoming observation indicates that the active user is interested in the interacted item, and the affected latent factors are adjusted to account for this assumption in a single step. However, in their system, implicit feedback is generated through many signals, such as clicks, views and comments. To account for this fact, individual learning rates are introduced for every observation,

where each source of feedback is assigned a different confidence level, based on the behavior of user that generates said feedback and features of the affected item. At recommendation time, considering that it is impractical to compute similarities between millions of items at every step, side information such as type similarity, time factor and demographic information are considered to reduce the size of candidate sets.

4.2.2.6 Reservoir-based approaches

The main advantage of ISGD-based methods are their scalability, as they perform incremental updates for every incoming observation in the stream. A potential shortcoming of performing single updates in such manner is reduced prediction quality in comparison to batch-trained models. However, memory usage is also of great concern for stream-based processing. To overcome such issue, Diaz-Aviles et al. (2012b) proposed Stream Ranking Matrix Factorization (RMFX), an online framework for topic recommendation in social streams that maintains a representative sample of the dataset in a *reservoir* (Vitter, 1985). Such reservoir incrementally maintains a random sample of fixed size $|R|$ of the incoming observations in the stream, where observations can occur more than once in order to reflect the distribution of observed data. The t^{th} observation in the stream is included in the reservoir with probability $|R|/t$, replacing uniformly at random a previously stored observation. The framework also includes a selective model update based on personalized small buffers, that keeps negative items for each user. These items are then used to update the MF model following a pairwise learning to rank approach, where items stored in the reservoir are considered as positive feedback. RMFX periodically performs model updates based on the reservoir, as opposed to ISGD that updates incrementally and continuously.

Diaz-Aviles et al. (2012a) extends RMFX by exploring additional sampling techniques. These strategies include a single pass approach that updates the model at every iteration, a user buffer that retains some of the most recent observations per user, and the aforementioned reservoir sampling approach. The task of the proposed framework, trained on a dataset of tweets, is to recommend hashtags to users. Results showed that reservoir sampling obtained the best results, followed by the user buffer one, where increasing the buffer size boosted performance.

Another reservoir-based approach for recommendation in social media was studied in Chen et al. (2013). Instead of periodically updating the model based on reservoir as done in Diaz-Aviles et al. (2012b), the authors propose TeRec, a MF approach that performs incremental updates after every observation, in order to always generate up-to-date lists. TeRec, which was designed to recommend appropriate hashtags for tweets, maintains a reservoir of samples that, instead of maintaining a sketch of the dataset, is updated based on recency, i.e., older observations in the reservoir are more likely to be replaced. The idea of this update procedure is to capture current user interests that better reflect their actions in real-time. Then, for every newly arrived data, the model is updated with the new observation, and with positive and negative samples stored in the reservoir. The authors report significant improvements in comparison to RMFX, which are related to the model always being up-to-date with recent data.

Wang et al. (2018b) proposes the use of a reservoir to capture long-term user interests, while also using new information incoming from streams as short-term ones. Their proposal, which extends PMF (Mnih and Salakhutdinov, 2007), updates the model based on incoming windows of data, by selecting samples from both the reservoir and the data stream. Samples are ranked and selected based on a proposed Gaussian classification model, that estimates their impact on the current model, where lower rank instances are assumed to be more informative as they are more likely to cause higher changes in the existing model, and thus have higher probability to be sampled. The authors report improvements when comparing their approach to

RMFX in movie-based and e-commerce datasets, at the expense of higher update time related to their proposed sampling procedure.

An approach to reduce memory usage when updating a incremental model is presented in Huang et al. (2017). Their proposal consists in a framework to perform incremental updates on MF models by designing a linear transformation of user and item latent factors over time. The idea is to have a model previously trained in batch, and update the latent factors with the arrival of new data, with the assumption that the latent feature matrix over a time period can be described as a linear function of its past value and by the factorization of the incremental matrix of new observations. Hence, models are updated incrementally with new ratings handled in a batch by batch basis, according to a time period defined by the application. Experiments showed that their approach obtained competitive accuracy results, compared to the approach of Rendle and Schmidt-Thieme (2008), with significant reduced memory usage.

4.2.3 Other model-based approaches

In this subsection we mention some of the efforts made outside of the scope of the MF setting for online recommendation, such as Markov models, deep learning and reinforcement learning. These models have also been explored in the somewhat related field of session-based recommendation, a categorization of sequence-aware recommendation (Quadrana et al., 2018). In this thesis we are interested in SBRS and incremental algorithms, specifically designed to the streaming model. A recent survey of session-based recommender systems that discuss these other models for recommendation can be found in Wang et al. (2021). As discussed by the authors in their paper, most existing studies in session-based recommender systems work on offline and static data, which is unrealistic given the naturally streaming nature of recommendation data. Thus, the discussion of these papers falls outside of the scope of SBRS and we limit to those that made specific contributions to our scope.

Garcin et al. (2013) proposed the use of context trees in the domain of news recommendation, which is characterized by frequent addition of new items with short lifespan and visits from anonymous users. Thus, both user and item cold-start are pervasive in this domain. Context trees partition the space of contexts organized in a hierarchy, where contexts are defined as sequences of articles or topics, or distribution of topics. Each node in the tree is a context and correspond to sequences within a partition, and contexts become increasingly specific deeper in the tree. A set of prediction models (experts) is associated with each context, and recommendations are made by combining the predictions of several experts. This combination is updated sequentially following Bayes' theorem. The tree is dynamically updated based on an active article pool that changes periodically, where new branches are created with the addition of new articles, and nodes of old articles are removed when these leave the current pool.

Chang et al. (2017) proposed a framework that deals with streams through a continuous-time random process, that captures three types of events: user feedback activities, new users and new items. Continuous Markov processes are used to model time-varying user and item factors, with the goal of capturing time-related dynamics that naturally occur in streaming settings. The framework is composed of two modules: an offline parameter estimation that builds an initial model from historical data, and an online phase that predicts unknown ratings based on a variational Bayesian method that performs efficient online inference. Experiments show that their proposal outperforms batch algorithms in movie-based datasets with explicit feedback.

Song et al. (2019) proposed Coupled Variational Recurrent CF, a framework that combines the previously mentioned Bayesian framework with deep learning methods. Latent factors are modeled as the combination of a stationary term that captures long-term preferences that varies slowly over time, and a dynamic one that captures short-term changes following a

Markov process. The variational inference algorithm then leverages two variational recurrent neural networks, one for users and another one for items. Experiments show that the framework models complex drifting patterns and slightly outperforms the one proposed in Chang et al. (2017), albeit with higher complexity and sensitivity to updating intervals.

Wang et al. (2018a) argue that a limitation of sequential recommender models based on recurrent neural networks is that they are limited in their capability of capturing long-term information, as they are trained on recent data. To capture both dynamic (short-term) and stable (long-term) interests in streaming scenarios, the authors proposed Neural Memory Recommender Networks, a model that relies on key-value memory networks (Miller et al., 2016). As the limitation of deploying deep learning based methods in streaming environments is its costly training procedure, the authors also proposed a sampling procedure based on Generative Adversarial Network (GAN) (Goodfellow et al., 2020) that generates informative negative samples to the active user based on the current values of the model parameters. The authors report improvements in movie-based datasets, but the model still relies on parameter pre-training and expensive update procedures.

Finally, we mention some contributions based on reinforcement learning, in particular the ones related to multi-armed bandits. Multi-armed bandit approaches (Sutton and Barto, 2018) generally represent each item in the dataset as an arm to be pulled, and the goal of the model is to learn a strategy that maximizes user satisfaction with each recommendation (exploitation) while also gaining knowledge about their tastes to further maximize satisfaction over time (exploration). The strategy is then updated iteratively based on the available feedback on the recommended items: selecting an arm is equivalent to recommending an item, and the reward is the user response to the recommendation. As noted in Al-Ghossein et al. (2021), the sequential update procedure somewhat aligns to the scope of SBRS. A major consideration, however, is how to efficiently update these models for new incoming observations to allow its deployment in streaming settings.

Li et al. (2010) modeled personalized news recommendation as a contextual bandit problem. To solve it, they introduced an algorithm, denoted LinUCB, that views articles as arms, and sequentially selects articles to be recommended based on contextual information of both users and articles (with disjointed and hybrid features), and leverages user clicks to optimize such selection. The approach performs well in their evaluated setting, although exploration can be hindered with a large article pool, and performance depends on feature construction.

Also in the context of news recommendation, Li et al. (2016) proposed the use of collaborative information in multi-armed bandits through adaptive clustering, that dynamically groups users according to the considered items, and at the same time groups items based on the similarity of clusters generated over the users, which is similar to co-clustering (George and Merugu, 2005). Their proposal is well-suited for user cold-start scenarios, as recommendation for new users can be made based on information from old ones, although it has higher computational cost compared to other multi-armed bandit methods such as LinUCB.

For music recommendation, Wang et al. (2014) proposed a multi-armed bandit approach that aims to provide accurate and novel recommendations in interactive manner. Their work investigates how to recommend repeated songs that are of interest to the current user (exploitation), while also providing novel content (exploration), i.e., songs that might be of interest but are unknown to the current user. The approach treats songs as arms, leverages explicit feedback signals as payoffs, and uses content-based information as feature vectors. Parameters are updated following a Bayesian regression model. While their approach is shown to perform well in cold-start scenarios, since feature vectors represent content-based information, it leverages

explicit feedback, which is onerous to collect (specially in music-based domains), and it keeps user's preferences static, which is unrealistic in streaming settings.

Another Bayesian model for music recommendation is presented in Çapan et al. (2022). Their work follows the assumption that, while the goal of a recommendation algorithm is to learn from previous interactions with users, the algorithm actually influences the interaction data that is further used for training. In other words, feedback is generated in interactive manner: users typically choose one item among a limited subset presented by the recommendation algorithm, where the subset was learned based on previous interactions. Their proposal, the Dirichlet-Luce choice model, accounts for such interactive characteristic, leading to an online bandit algorithm based on Thompson sampling, that deals with biases inherently related to online recommender systems, e.g., overestimation of user preferences and penalization of underexposed or underrepresented items.

Pereira et al. (2019) proposed a framework called Counterfactual Dueling Bandits, designed for sequential music recommendation with continuous feedback. Their approach encodes objects in a low-dimensional feature space, composed of collaborative and content-based signals, i.e., textual, social and audio representations. On this space, multiple directions are explored through duels of candidate recommendation models, defined as arms. To select the winning model at some time point, the approach leverages user feedback through implicit signals, song plays and skips, that are used as positive and negative feedback, respectively, and estimates the effectiveness of each model based on their ranking of the next song listened by the user at that time. Results show that it outperforms two variants of LinUCB (Li et al., 2010).

Sanz-Cruzado et al. (2019) proposed a simple multi-armed nearest-neighbors bandit framework for interactive recommendation. The approach is similar to traditional nearest-neighbors recommendation, with the addition of a stochastic exploration capability of neighborhoods through the application of Thompson sampling. As opposed to general multi-armed bandits, their approach model the arms as users (neighbors), and items to be recommended are selected on the advise of chosen neighbors. Such procedure introduces a stochastic factor in the neighborhood selection. Results show that their approach outperforms user-based KNN and implicit MF in accuracy in small datasets, with updates performed after every 500 data points.

4.3 CHANGE DETECTION MODULE

The change detection module defines mechanisms to actively detect drifts in the learned concepts, as opposed to memory and learning modules that passively adapt to changes (Gama et al., 2014; Al-Ghossein et al., 2021). In other words, this module is specifically designed to detect changes and is responsible for triggering procedures to adapt models to such changes. Contributions related to this module for SBRS are still scarce in the literature (Al-Ghossein et al., 2021). We next discuss some contributions towards this module.

Li et al. (2007) proposed an approach to detect drifting preferences of users with the use of decision trees through CVFDT algorithm (Hulten et al., 2001). The key idea is to built two decision trees for each item in the dataset based on ratings of correlated items: one built based on current observations and another based on older observations. The old trees are then replaced by new trees once their accuracy is surpassed by the new ones.

Al-Ghossein et al. (2018a) proposes dynamic local models (Christakopoulou and Karypis, 2016) to adapt concepts based on drifts in user interests. Their proposal, DOLORES, maintain several local item-based models and a global one, similar to the proposal of Miranda and Jorge (2009), where each user is clustered to a single local model. Drifting interests for a given user u are detected when a different local model starts to perform better than the one u was

previously assigned to. When u is reassigned to a different local model, a forgetting technique proposed by Matuszyk and Spiliopoulou (2014) is used to remove old interactions of u, in order to better reflect her current interests. Both the global model and the local model that the active user is assigned to are updated for each incoming observation.

Al-Ghossein et al. (2018b) proposes a hybrid system that combines textual information to reduce the impact of item cold-start in data streams to an incremental matrix factorization method (Vinagre et al., 2014b) to keep users' preferences up-to-date. The drift detection is made on item textual descriptions with topic modeling of items and with the adaptive sliding window algorithm (Bifet and Gavalda, 2007), such that the topic model is maintained up-to-date and retrained in batch when a drift is detected using documents automatically selected by the adaptive sliding window algorithm.

José et al. (2020) propose to extend incremental matrix factorization methods with personalized learning rates on individual user and item basis. Their algorithm, ADADRIFT, uses two moving window parameters to detect short-term and long-term changes, which are used to increase the learning rate to quickly adapt to drifts, or decrease it in cases where the concepts are stable. The reported experiments, which extended BRISMF (Takács et al., 2009) and ISGD (Vinagre et al., 2014b) with ADADRIFT, suggest a high correlation between the impact of ADADRIFT and the average size of interaction history, i.e., it requires long lists of interactions for drifts to become noticeable and for accuracy to increase with its application, while a decrease in accuracy was obtained for scenarios with short lists.

4.4 EVALUATION CONTRIBUTIONS

Finally, we mention important contributions that raise relevant questions towards the evaluation of recommender systems on data streams settings. Recommender systems can be evaluated through online and offline methods (Gunawardana et al., 2022). Although online methods provide stronger evidence towards the efficiency of a given RS, as it actually evaluates its impact on user behavior, RS are usually evaluated through offline methods. The main advantage of such methods is that user behavior can be simulated through finite datasets, and several algorithms can then be evaluated and compared following the same protocols.

In an offline setting, hold-out methods are used. These methods split a dataset into two non overlapping subsets - a training set and a testing set. The training set is used to build a predictive model, and the test set is used to evaluate the performance of the predictive model built from the training set, by comparing the predictions and the actual observations from the test set.

Vinagre et al. (2014a) discusses several limitations of evaluating stream-based RS with hold-out, offline methods. First, if the dataset is randomly split, the time dimension is disregarded, as it is naturally embedded in the sequential information that is shuffled. Besides losing dynamic information regarding users and items that are related to time, some inconsistencies may be introduced in the evaluation, such as predicting a past interaction based on future ones. Second, a static hold-out method would limit the evaluation of incremental algorithms, since these are updated based on newly available observations. As such, the model is dynamic and the datasets splits should not be static. Finally, in online systems, users' decisions are actually influenced by the recommendations, i.e., the RS may direct the user towards the recommended items. This impact is hardly trivial to simulate on offline settings.

Given the aforementioned problems with hold-out method, Vinagre et al. (2014a) then proposes a prequential evaluation protocol, particularly suited for SBRS evaluation. The protocol consists in a test-then-learn procedure that is executed for each new observation in the stream (or finite dataset). In its basic form, for each new observation, a recommendation is generated for the

active user with the current model. The prediction is then tested against the actual observation, and finally the model is updated with this observation. The main advantage of the protocol is that it allows continuous monitoring of the RS' performance over time through several metrics.

Another contribution from Vinagre et al. (2021) discusses the challenges of conducting efficient statistical significance validation tests in stream-based settings, and proposes the use of a k-fold validation framework with McNemar and Wilcoxon tests over adaptive-size sliding windows (Bifet et al., 2015). Our experiments are evaluated through the test-then-learn protocol proposed in Vinagre et al. (2014a), and statistical significance of the results are measured with McNemar's test. We discuss the evaluation protocol in more depth in Chapter 6.

Lommatzsch and Albayrak (2015) evaluated several features of user-item interaction streams in the context of news recommendation. In these scenarios, new items are frequently added and are typically only relevant for a few days. The work analyzes and describes different properties of several news portals that were available in the ACM RecSys News Challenge 2013[2], such as time dependent user behavior, e.g., number of interactions, content, time and day of the week, and used device, and item properties, e.g., popularity and lifecycle, and show that these features vary and are highly dependent on the domain.

An evaluation of different algorithms is also presented, both online and offline. In the online evaluation, an algorithm must provide a few suggestions to each request within 100 ms including the communication time, and the performance is measure with *Click-Through-Rate* (CTR). The offline evaluation analyzes the effectiveness of algorithms in predicting the next clicks of a user. Hence, in this context, algorithms must operate under strict time constraints and account for recent information that is continuously generated at fast rates. The evaluated algorithms include recommending the most popular articles, most popular sequence, recently requested articles, user-based and item based CF, content-based and ensemble methods. The results show that ensemble methods perform well in online scenarios, as there is no optimal algorithm for all contexts, and incoming requests can be delegated to the current best-suited one. The work also suggests that neighborhood-based methods are well suited for streams as they are able to adapt to its changing properties without significant overheads during model update.

Also in the context of news recommendation, Jugovac et al. (2018) proposed StreamingRec, an open-source framework for evaluating news SBRS in reproducible manner. StreamingRec is based on a replay evaluation protocol that allows algorithms to update its underlying models in real time when new observations are recorded and new items are available for recommendation. The framework implements several baseline algorithms, including algorithms that can immediately incorporate the incoming events into their predictions, with a focus on session-based recommendation (Quadrana et al., 2018).

Their work evaluates the performance of several incremental algorithms on two datasets, and compare their performance with two complex state-of-the-art models that are periodically updated, BPR (Rendle et al., 2009) (Section 2.3.2.2) and GRU4REC (Hidasi and Karatzoglou, 2018), a state-of-the-art algorithm for session-based recommendation (Section 2.5.2). These include recommending the most popular, recently popular, recently published and recently clicked items, item-based CF (Deshpande and Karypis, 2004), session-based nearest neighbors (Jannach and Ludewig, 2017), sequence-aware nearest neighbors (Ludewig and Jannach, 2018), pairwise item co-occurrences in a session, sequential pattern mining and content-based methods. The results show that the worst performing algorithms are the ones that do not consider item recency, and that the best results are obtained by the neighborhood-based methods that do take into account item recency, outperforming the complex models that are only periodically updated.

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

Frigó et al. (2017) note that while in real-world scenarios recommender systems processes data and provide suggestions following an online temporal sequence, the design and evaluation of online learning methods in highly non-stationary environments is scarce in the literature. To motivate the design of such algorithms, their work evaluates and compares batch and online-based methods in non-stationary datasets. The evaluated algorithms include popularity-based recommender, time-based nearest neighbors (Deshpande and Karypis, 2004; Ding and Li, 2005), item-to-item transition model and batch, online and assymmetric MF. Their work also proposes a sampling technique to generate positive and negative samples to update the model in a single iteration in online manner, thus somewhat resembling a batch procedure.

The obtained results showed that although batch-based MF performs well in rating prediction, they are outperformed by item-to-item nearest neighbor methods in ranking tasks, and their performance drops considerably when data is non-stationary. Batch-based MF is also significantly outperformed by its online counterparts. The transition model, despite its simplicity, presented high competitiveness in non-stationary settings. Overall, their work showed that simpler algorithms that can be efficiently updated online with the most recent data outperforms more complex algorithms that require periodical updates.

Finally, Viniski et al. (2021) presents a case study comparing the performance of batch and incremental algorithms under concept drifts and cold-start on a dataset that presents both issues. The work compares batch versions of SVD (Sarwar et al., 2000), BPR-MF (Rendle et al., 2009), Generalized Matrix Factorization, Multi-layer Perceptron and Neural Matrix Factorization (He et al., 2017) to two incremental algorithms, ISGD (Vinagre et al., 2014b) and IBPR-MF (Rendle et al., 2009). Results show that incremental algorithms, specially ISGD, outperformed batch-based models, including neural ones, highlighting the significance of constantly updating models as new data becomes available. As a consequence of such updates, incremental models are also able to recover swiftly from cold-start issues and better adapt to concept drifts in comparison to periodical training.

4.5 DISCUSSION

Throughout this chapter, we have outlined related work, particularly those that somehow acknowledge that user generated data follows the properties of a data stream (Domingos and Hulten, 2001). Specifically, that such data is generated in unpredictable order and is potentially unbounded, new users and items continuously join the system, and models must evolve to incorporate newly generated data and provide recommendations based on up-to-date information in scalable manner.

We note that the vast majority of related work are generally not suitable for the online setting, as they either require an initial robust model or only periodically update its underlying models. Both approaches require storing historical data in some capacity, which in turn raises the following issues: (1) increasing update time related to the retraining/updating of the current model; (2) disregard of incoming observations and information from new users/items that are only considered at the next update period, which in turn fails to track fast-changing trends, such as user preferences and changes in item popularity; (3) increasing memory requirements to store historical data.

Another consideration is the type of feedback available for training. While explicit feedback is considerably more informative, it is onerous to collect as it requires direct labeling from users. Implicit feedback on the other hand, can be easily collected from various signals and it is continuously generated similarly to a data stream. We thus focus on learning from implicit feedback in the following chapters.

A minor subset of work, that are more closely related to our present work, propose to incorporate each incoming observation based on implicit feedback in the previously learned model incrementally (Vinagre et al., 2014b), potentially at the expense of some accuracy, with the added benefit of allowing models to stay up-to-date along the stream. These assumptions form a more realistic setting and allow the application of these models in online scenarios. Approaches adapted to this setting include neighborhood-based methods, based on the KNN framework (Strategy *item-based CF* in Table 4.3) (Papagelis et al., 2005; Miranda and Jorge, 2009; Chandramouli et al., 2011; Huang et al., 2015; Yagci et al., 2017), and model-based methods, particularly matrix factorization (Strategy *incremental updates for each incoming observation* in Table 4.4) (Vinagre et al., 2014b, 2015a; Anyosa et al., 2018; Vinagre et al., 2018a,b).

Despite the premise of obtaining scalable updates at the expense of accuracy, such approaches are shown to outperform in accuracy others that are based on periodical retraining (Section 4.4), including recent complex models based on deep learning. In fact, a recent array of comparative work, as discussed in Section 4.4, has suggested that conceptually simpler algorithms (e.g. item KNN) with appropriate configuration and parameter adjustment outperform more complex neural methods both in accuracy and scalability (Dacrema et al., 2021; Ludewig et al., 2021; Latifi and Jannach, 2022). How to efficiently update complex neural methods in streaming settings is still an open challenge (Zhang et al., 2019) and falls outside the scope of this thesis. On the other hand, recent surveys on RS calls for contributions on incremental learning on non-stationary and streaming settings (Zhang et al., 2019; Wang et al., 2021).

Hence, in this thesis we focus on the top-N recommendation problem following the more realistic setting of learning from data streams, defined in this context as a stream of user interactions, and investigate ways of addressing the challenges related to stream-based learning discussed so far. In such a setting, a model must be updated based on each incoming observation that is subsequently discarded, while being able to provide recommendations based on up-to-date information when required.

In particular, we focus on neighborhood-based models as they present competitive accuracy in several recommendation scenarios (Vinagre et al., 2014b; Lommatzsch and Albayrak, 2015; Jugovac et al., 2018; Frigó et al., 2017). Two major considerations of these models, besides the ones inherently related to the streaming setting, are scalability, since naively updating similarity matrices results in impractical update times, and sparsity, since little information is available for each user/item. On this note, graph-based approaches (Section 4.2.1.1) are well suited, but seldom explored, as similarities can be induced based on indirect connections, i.e., nodes that are not directly connected can still influence other nodes by propagating information through the edges of the graph. Such impact can be measured through its weights and also by the algorithm in use.

In the following chapter we define our contributions, centered on a proposed graph-based model, $\text{IGSI}_{\hat{\pi}^t}$, that aims to overcome these limitations. $\text{IGSI}_{\hat{\pi}^t}$ is capable of continuously incorporating user feedback, provides scalable recommendations based on simulations of random walks that balances short-term and long-term user preferences. We also propose and incorporate in our model forgetting techniques, which can be generalized to other neighborhood-based methods. These remove obsolete information from models, which in turn improves scalability and predictive performance.

5 PROPOSED APPROACH

In this chapter we present our proposal and define our contributions. As described throughout the previous chapters, we are interested in exploring and developing incremental algorithms for data stream recommendation, or Stream-based Recommender Systems (SBRS) (Al-Ghossein et al., 2021). The motivation behind this is three-fold: First, user generated data shares several similarities to a data stream, i.e., user feedback is generated continuously at unpredictable rate and order, is potentially unbounded, new users and items enter into the system and previously learned concepts change over time, and is thus a natural way of designing the recommendation problem. Second, incremental algorithms consistently outperform their batch-based counterparts in online scenarios as shown in numerous works (Lommatzsch and Albayrak, 2015; Frigó et al., 2017; Jugovac et al., 2018; Viniski et al., 2021). Third, recommendation is not yet frequently addressed as a data stream problem, hence many open challenges exist in this research field (Al-Ghossein et al., 2021).

Approaching the recommendation problem as a data stream requires designing algorithms that learn from data in incremental manner, such that these incremental models must update their concepts in a single-pass over the incoming observations, preferably *without storing them*, include new users and items *without retraining* and generate recommendations *with up-to-date information* with *restricted processing time and available memory* in order to avoid falling behind the data. These constrains have direct impact in scalability and accuracy, since algorithms have limited time and resources to update their underlying models, which are ever-growing, and recommend relevant items on-the-fly with up-to-date information.

In Chapters 3 and 4 we detailed some of the most relevant approaches for recommendation on data streams. We are particularly interested in those that continuously update their models without storing observations. This way, the underlying model is always up-to-date without requiring retraining. Approaches are categorized in neighborhood-based and model-based approaches. We are interested in neighborhood-based methods as they present competitive accuracy in several recommendation scenarios (Lommatzsch and Albayrak, 2015; Dacrema et al., 2021; Ludewig et al., 2021; Latifi and Jannach, 2022). However, numerous challenges are still present when designing models in data stream settings.

First, these approaches are prone to scalability issues. Although the similarities between users or items can be updated incrementally, the inclusion of recent information in the recommendations requires computing the nearest neighbors after these updates, which increases update and recommendation times, specially for ever-growing models.

Second, recommender systems in general are prone to sparsity issues. These issues are related to the amount of information available to learn user preferences. As users interact with only a small subset of items in the catalog, little information is available from users and items.

Third, in many online systems the majority of interactions are centered in the most popular items available, which form only a small fraction of the item catalog. This means that there is even less information regarding the remaining set of less popular items, which form the *long-tail* of the interaction distribution (Celma and Cano, 2008; Cremonesi et al., 2010). Including items from the long-tail in the recommendation lists increases its novelty and diversity, which serve as other relevant metrics of user engagement (Gunawardana et al., 2022).

Graph-based approaches, which form a subset of neighborhood-based ones, are well suited for sparsity issues, as they easily allow incremental inclusion of data and are able to infer relations between nodes that are not directly connected, resulting in accuracy improvements. The

main consideration is how to infer these relations, i.e, how to rank users and items based on their connections. We described some strategies in Section 2.3.1.1, specifically random walk-based ones that present competitive results. However, they are not necessarily designed for streaming data.

We note here the contributions of Cooper et al. (2014) and Christoffel et al. (2015) that represent user and item relations in user-item bipartite graphs. Although not originally designed for data stream recommendation, their contributions raises concerns related to scalability and limited resources, i.e., running time and memory usage. To avoid storing entire rating matrices in main memory and the high computational time required in computing stationary distributions, their proposals are centered on random walk sampling, or simulations of random walks, where such distributions are approximated based on a predefined number of samples. This procedure introduces a trade-off between accuracy and scalability, such that higher number of samples better estimates the distribution at the expense of higher running times. Based on their findings, we argue that it is beneficial to use graph-based models in order to overcome sparsity and scalability issues related to recommendation and data stream mining.

One of the drawbacks of these methods is that although they can be adapted to learn incrementally and approach data sequentially, they disregard time-related information. User feedback is naturally a chronological sequence, and an intrinsic relation between data and time exists, such that user preferences and trends adjust over time (Moore and Chen, 2013). The sequential user logs can be informative of their long-term preferences, while recent events are more relevant to predict short-term preferences (Quadrana et al., 2018), since the next action of a user can be directly dependent on her recent actions (Shani et al., 2005). Thus, these methods are effective in modeling long-term interests since they are constantly updated with new user feedback, but fail to account for short-term interests that are represented by the most recent actions.

Furthermore, the concepts extracted from data streams are not static, i.e., users change their opinion over time (Koychev and Schwab, 2000; Koychev, 2000), meaning that while the sudden change in short-term preferences may be temporary, it may also indicate a change in interests, a phenomena known as *concept drift* (Gama et al., 2014). Incremental models are constantly updated with new user feedback, but still retain concepts inferred from past data that may eventually become outdated. Relying solely on incremental learning may lead to models that are sensitive to noisy data. An alternative to overcome this issue is the deployment of forgetting mechanisms to *forget* data that is no longer representative of the current concepts being learned, i.e., to remove outdated information from models (Matuszyk et al., 2018).

Thus, the main objective of this research is the proposal of a stream-based recommender system designed to work in online scenarios, tackle the issues inherently related to recommender systems and data stream mining, such as sparsity, accuracy, scalability and concept drift, and provide alternatives to the shortcomings of previous algorithms. To that end, our contributions are centered on a stream-based model, that consists in an evolving graph of sequential interactions with forgetting for data stream recommendation with implicit feedback. We use the term *evolving* in the sense that the model *adapts* and learns, continuously and incrementally, based on information generated along the data stream, but also removes information that is deemed as obsolete. Figure 5.1 presents an overview of our proposal.

We focus on graph-based methods due to their robustness to sparsity, natural incremental capability and flexibility. We split our proposal into two parts. First, we propose to incorporate implicit user feedback into an item-graph in incremental manner with the assumption that user behavior can be extracted from the sequence of user interactions as time progresses. With this assumption, potentially relevant item recommendations can be made to users based on past user

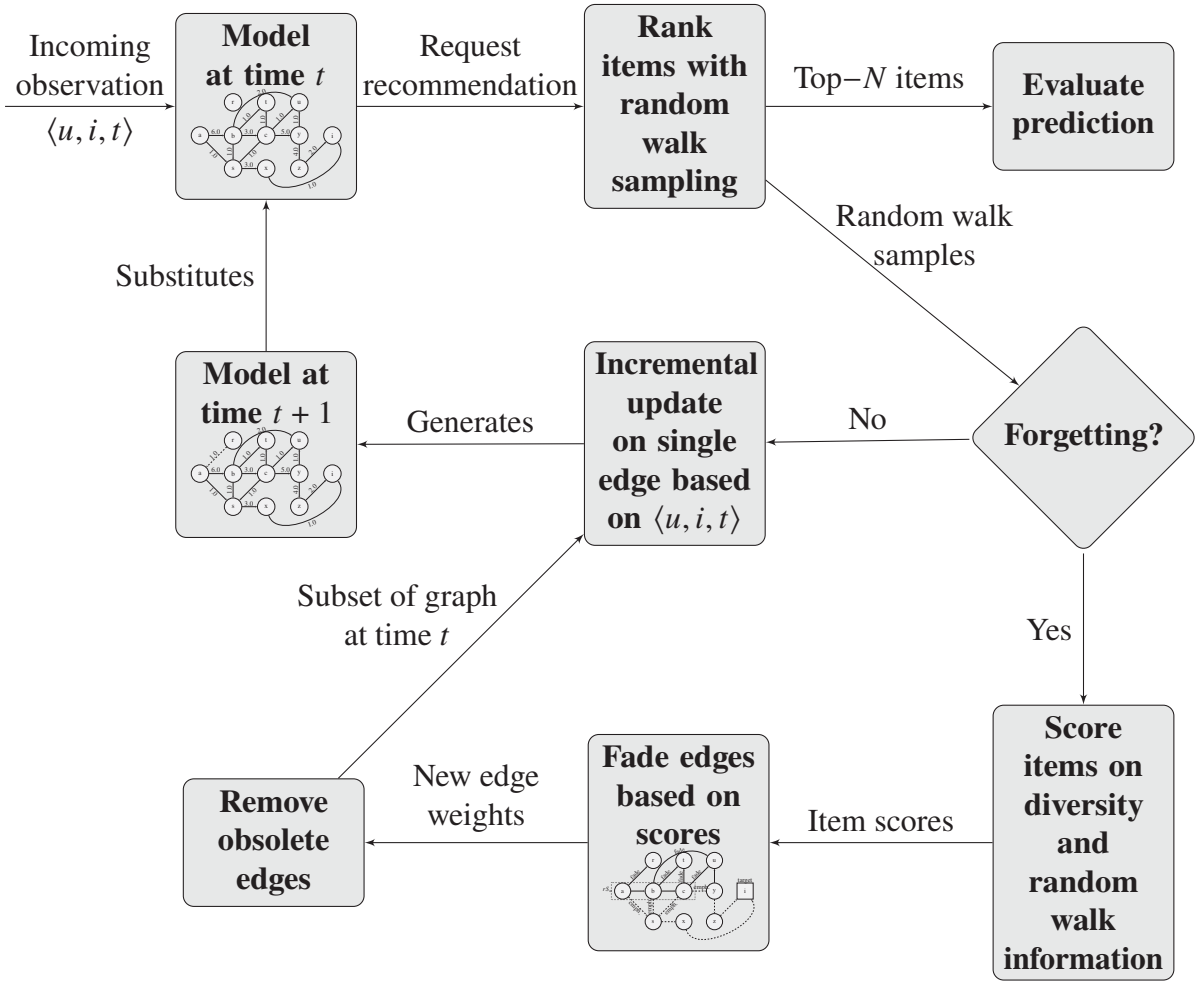


Figure 5.1: Overview of our proposed stream-based recommender system. This process is repeated for every observation generated along the stream.

behavior. As an example, the release of a film can lead a user to watch the director's past work before watching the film. Another example is the birth of a child. As time progresses, the family can direct their purchases towards products intended for the children as they grow.

In that sense, we assume that the sequence of interactions can be useful in modeling short-term and long-term user interest. To that end, edges are continuously included in the graph and their weights are updated according to the sequential user interactions, such that for each incoming user feedback in a data stream, a directed edge connects the last item interacted by the user to the current interaction, and the relevance of each sequential interaction is reinforced in the weight of the edge. We explore the use of *random walk sampling* (or simulation of random walks) on the item-graph in order to generate recommendations to users in real-time based on up-to-date information.

The item-graph offers two main differences to a user-item bipartite graph: it extracts information from sequential data, and allows the recommendation of items based on short-term and long-term interests. We refer to the proposed item-graph as IGSI (Incremental Graph of Sequential Interactions). These contributions are described in Section 5.1.

Second, we propose a forgetting mechanism designed specifically to exploit the advantages of IGSI, but that could be generalized to other graph-based and neighborhood-based approaches. We propose to use random walk sampling not only to rank items and generate recommendations to users but also to capture structural information from the graph and infer the

relevance of items. This way, items that are perceived as irrelevant can be forgotten. We discuss these contributions and the challenges of deploying forgetting mechanisms in Section 5.2.

5.1 SCALABLE STREAM-BASED RECOMMENDATIONS WITH RANDOM WALKS ON INCREMENTAL GRAPH OF SEQUENTIAL INTERACTIONS

In order to continuously capture sequential interactions between users and items, we create a directed item-graph, where nodes represent items and edges represent user interactions, such that the edge direction indicates the order in which items where visited, i.e., sequential interaction. Thus, an edge from item i to item j exists if a user interacted with item i, and her next interaction was with item j. Therefore, for each new user interaction, the feedback is included into the graph by an edge that connects the last interacted item to the item of the new interaction.

To distinguish the relevance of edges, each edge has an associated weight. Edge weights are updated every time a transition is made by a user, and the weight to be reinforced is associated with the time elapsed between interactions. In other words, interactions that occur within a time window (e.g. user session) are more relevant than interactions that occur further apart, and this relevance is considered when updating the weights.

Figure 5.2 shows an example of IGSI online maintenance, illustrating two possible scenarios based on an incoming event. Consider the graph presented in Figure 5.2(a) and a user u, whose last interaction was with item r. At some time t, u interacts with item a. Since there is no edge connecting item r to item a, the edge from item r to item a is inserted, as illustrated by the dashed edge in Figure 5.2(b). Now consider that at time $t+1$, u interacts with item s. Since an edge from item a to item s exists, its weight must be updated, as denoted in Figure 5.2(c).

Denote by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ a weighted directed graph, where $\mathcal{V} = \{v_1, v_2, ..., v_n\} \subseteq \mathcal{I}$ denotes the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges. Each edge e has an associated weight $w(e) \in \mathbb{R}_+$. We define $S_u = \{(v_1, t_1), (v_2, t_2), ..., (v_n, t_n)\}$ as a list of items interacted by user $u \in \mathcal{U}$ ordered according to time t. The last interacted item by u is defined as $li_u \in \mathcal{I}$ and the time of the last interaction by u is defined as lt_u, i.e., (li_u, lt_u) is the last element of S_u. The graph is updated incrementally considering li_u, lt_u and the current observed interaction.

User feedback is modeled as a data stream, where each observation is defined as $\langle u, i, t \rangle$, indicating that user u interacted with item i at time t. When updating the graph considering the current observation, the model must include feedback from new incoming users and items, while also updating older concepts. To that end, edge weights are updated as:

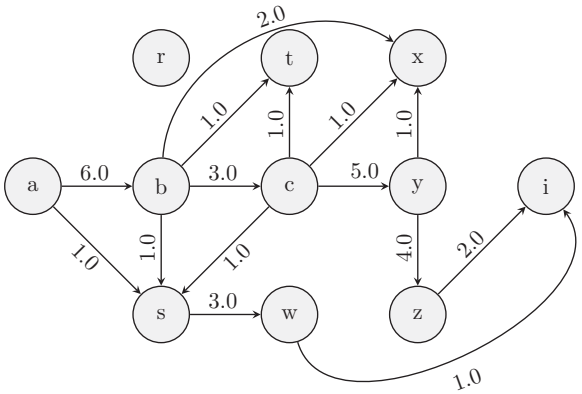
$$w((li_u, i)) = \begin{cases} w((li_u, i)) + 1 & \text{if } t - lt_u < \rho \\ w((li_u, i)) + \frac{\rho}{t - lt_u} & \text{otherwise} \end{cases} \tag{5.1}$$

where ρ is a time window hyperparameter. Eq.(5.1) gives more weight to the edge if the interactions occurred within a time window (e.g. user session) and decreases the weight to be added to the edge based on user's inactivity.

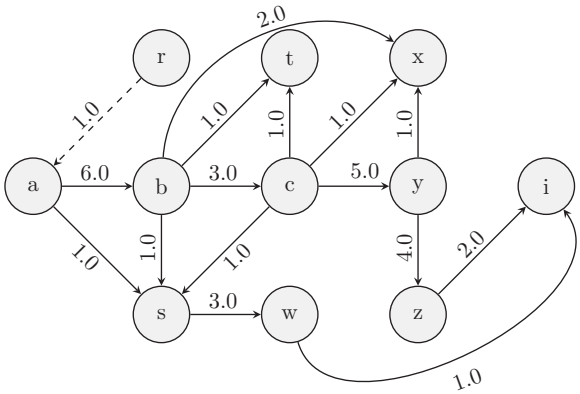
For each incoming observation $\langle u, i, t \rangle$, there are four possible scenarios to update the model, as outlined in Algorithm 8:

1. User and item are unknown by the system ($u \notin \mathcal{U}$ and $i \notin \mathcal{I}$). In this case, user and item are included in the system, a node for i is added to $\mathcal{V}$ and (i, t) is included in S_u;

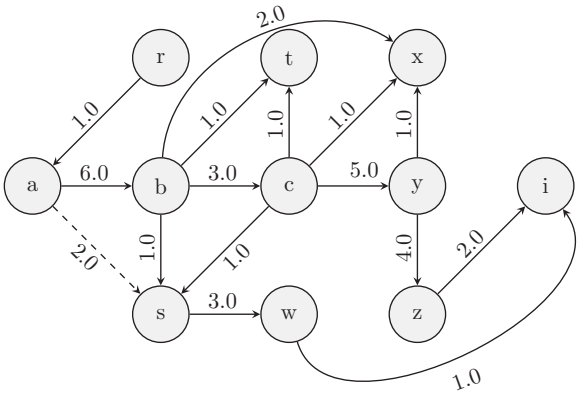
2. User is unknown and item is known by the system ($u \notin \mathcal{U}$ and $i \in \mathcal{I}$). In this case, user is included in the system and (i, t) is included in S_u. Given that i is the first interaction of u and i is known, there is no change in the graph;



(a) Example of a graph before updates based on user interactions.



(b) Insertion of edge from item r to item a based on user interaction.



(c) Weight update of edge from item a to item s based on user interaction.

Figure 5.2: An example of graph update based on a few user interactions. Considering the graph in (a) and a user u whose last interaction was with item r, (b) presents a scenario where u interacts with item a by inserting the edge connecting r to a. In (c), u interacts with item s after interacting with a, and the weight of the edge connecting a to s is updated.

3. User is known and item is unknown by the system ($u \in \mathcal{U}$ and $i \notin \mathcal{I}$). In this case, item is included in the system and a node for i is added into $\mathcal{V}$. In this scenario, u has already interacted with at least one item before. Since this is the first interaction by any user with i, an edge (li_u, i) is included in $\mathcal{E}$, where $w((li_u, i))$ is defined by Eq.(5.1). Lastly, (i, t) is included in S_u;

4. User and item are known by the system ($u \in \mathcal{U}$ and $i \in \mathcal{I}$). In this case, the current sequential interaction is li_u to i. If such interaction has happened before by any user, i.e., $(li_u, i) \in \mathcal{E}$, then $w((li_u, i))$ is updated according to Eq.(5.1). If this is the first time such interaction happens, an edge (li_u, i) is included in $\mathcal{E}$, and $w((li_u, i))$ is defined by Eq.(5.1).

5.1.1 Item ranking methods

Based on the incremental item-graph, relations between items can be inferred. Considering that IGSI is updated in a way that relevant items are directly connected by an edge, and the weight of such edge measures the relevance of the relation between items, we define that relevant nodes to a source item $v \in \mathcal{V}$ are nodes that are close to v. To capture relations between items, random-walk based methods are naturally well-suited, since they can propagate information

Algorithm 8 Online incremental graph update

Require:

$\mathcal{D} = \{(< u, i, t >)_1, (< u, i, t >)_2, ...\}$: data stream
$\mathcal{U}$: set of users
$\mathcal{I}$: set of items
S_u: list of interactions by user u
li_u: last item interacted by user u
lt_u: time of last interaction by user u
$(\mathcal{V}, \mathcal{E}, w)$: weighted directed graph
ρ: time window parameter

procedure IncrementalUpdate($\mathcal{D}$, $\mathcal{U}$, $\mathcal{I}$, S_u, li_u, lt_u, $(\mathcal{V}, \mathcal{E}, w)$, ρ)
  for $< u, i, t > \in \mathcal{D}$ **do**
    if $u \in \mathcal{U}$ **then**
      if $i \in \mathcal{I}$ **then**
        if $(li_u, i) \in \mathcal{E}$ **then**
          if $t - tl_u < \rho$ **then** // Eq.(5.1)
            $w((li_u, i)) \leftarrow w((li_u, i)) + 1$
          else
            $w((li_u, i)) \leftarrow w((li_u, i)) + \frac{\rho}{t - tl_u}$
        else
          $\mathcal{E} \leftarrow \mathcal{E} \cup \{(li_u, i)\}$
          if $t - tl_u < \rho$ **then** // Eq.(5.1)
            $w((li_u, i)) \leftarrow w((li_u, i)) + 1$
          else
            $w((li_u, i)) \leftarrow w((li_u, i)) + \frac{\rho}{t - tl_u}$
      else
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
        $\mathcal{V} \leftarrow \mathcal{I} \cup \{i\}$
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{(li_u, i)\}$
        if $t - tl_u < \rho$ **then** // Eq.(5.1)
          $w((li_u, i)) \leftarrow w((li_u, i)) + 1$
        else
          $w((li_u, i)) \leftarrow w((li_u, i)) + \frac{\rho}{t - tl_u}$
    else
      $\mathcal{U} \leftarrow \mathcal{U} \cup \{u\}$
      if $i \notin \mathcal{I}$ **then**
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
        $\mathcal{V} \leftarrow \mathcal{I} \cup \{i\}$
    $S_u \leftarrow S_u \cup \{(i, t)\}$
    $li_u \leftarrow i$
    $lt_u \leftarrow t$
  return $\mathcal{U}, \mathcal{I}, S_u, li_u, lt_u (\mathcal{V}, \mathcal{E}, w)$

along the edges and thus are robust to sparsity, while providing such information in scalable manner for short-step walks.

In Section 2.3.1.1, we described a few ranking procedures based on random walks. Two of them possess the aforementioned desirable characteristics. The first ranking procedure is based on the t-step random walk transition probability matrix $\mathbf{P}^t$, defined in Eq.(2.11). In this procedure, nodes are ranked for a source node v based on the probability of being reached with a fixed t-step walk starting in v. In the case of bipartite graphs, t must be an odd number starting from a user node so that the walk ends in an item node. For IGSI, any t is acceptable as the walks start and finish in item nodes. The limitation of this procedure is that ranking is limited to a fixed t-step walk, and the tendency is that larger steps deteriorate performance (Cooper et al., 2014; Paudel et al., 2017). We include in our experiments this ranking procedure with $t = 1$, which results in an algorithm analogously to a frequent mining procedure that is easy to compute, as it depends only on the weights of adjacent nodes. Despite the simplicity, the approach performs well under certain scenarios and is able to outperform other stream-based approaches. We refer to this ranking procedure on the item-graph as $\text{IGSI}_{\mathbf{P}^t}$. The limitation of it is that ranking is restricted to a fixed t-step walk, and the tendency is that larger steps deteriorate performance (Cooper et al., 2014; Paudel et al., 2017).

Alternatively, we can rank items based on a given node s through a t-step random walk with restart starting in s (Eq.(2.15)), which is a weighted sum of all landing probabilities. Thus, we aim to produce recommendations based on π^t.

However, it is impractical to constantly compute π^t in an evolving graph. To allow the ranking of items based on π^t in a data stream framework, we approximate π^t through simulation of random walks. More specifically, we can estimate π^t_s using Monte Carlo algorithms. Similar to Avrachenkov et al. (2007, 2011), we approximate π^t_s by simulating M independent t-step random walks starting from s, such that for each node $v \in \mathcal{V}$, $\pi^t_{s,v}$ is measured by the number of times v is visited by the M random walks multiplied by $\frac{(1-\gamma)}{M}$. Hence, the estimator $\hat{\pi}^t_s$ for a node v can be expressed by:

$$\hat{\pi}^t_{s,v} = \frac{1-\gamma}{M} \sum_{k=1}^{M} RW^k_{s,t}(v) \tag{5.2}$$

where $RW^k_{s,t}(v)$ is the number of visits to node v in the k-th run of a t-step random walk initiated at s. Algorithm 9 describes the implementation of the approximation procedure defined in Eq.(5.2). The algorithm samples M t-step random walks starting from a given vertex v_i and at each step selects a neighbor v_c at random, where the transition probability is proportional to the weight of the edges (Eq.(2.10)), and then in line 9 updates the visit count $\text{h}[v_c]$ by incrementing $h[v_c]$ by one. The sampling returns the visit count h multiplied by $\frac{(1-\gamma)}{M}$, where items with more visits are considered to be more relevant to the source node v_i. We refer to this ranking procedure as $\text{IGSI}_{\hat{\pi}^t}$.

5.1.2 Convergence of sampling

Similar to Fogaras et al. (2005); Avrachenkov et al. (2007); Sarkar et al. (2008), we can show that the rate of convergence of Eq.(5.2) is exponential. Recall that we run M independent t-step random walks starting from s. Denote R as the set of i.i.d. variables $R = \{RW^1_{s,t}, RW^2_{s,t}, ..., RW^M_{s,t}\}$. For a given node v, the maximum number of visits in a single walk $RW^i_{s,t}(v)$ is bounded by $[0, (1-\gamma)t]$, where t is the length of the walk. Given the law of large numbers, if the number of samples $M \to \infty$, then $\hat{\pi}^t_{s,v}$ converges to $\pi^t_{s,v}$. We can show that the rate of convergence is

Algorithm 9 Random walk sampling

Require:
v_i: starting vertex
$(\mathcal{V}, \mathcal{E}, w)$: weighted directed graph
t: length of walks
M: number of walks
γ: probability of continuing the random walk

```
1:  procedure RandomWalk(v_i, (V, E, w), t, M, γ)
2:      totalWalks = 0
3:      h[v] = 0, ∀v ∈ V
4:      while totalWalks < M do
5:          v_c = v_i    // v_c is the current node
6:          for i ∈ [1, 2..., t] do
7:              if random() < γ then
8:                  v_c = weightedRandomNeighbor(E[v_c])    // Eq. (2.10)
9:                  h[v_c]++
10:             else
11:                 break
12:         totalWalks + +
13:     return (1−γ)h / M
```

exponential using Hoeffding's inequality. We set the upper bound to be less than a small value δ. This gives:

$$P(|\hat{\pi}^t_{s,v} - \pi^t_{s,v}| \geq \varepsilon) \leq 2\exp\left(-\frac{2M\varepsilon^2}{(1-\gamma)^2 t^2}\right) \leq \delta$$

which provides a lower bound for M as $\frac{(1-\gamma)^2 t^2}{2\varepsilon^2} \log\left(\frac{2}{\delta}\right)$.

5.1.3 Recommendation methods

Scalable recommendations for users are generated by ranking items with algorithm $\text{IGSI}_{\hat{\pi}^t}$. This algorithm computes a relevance vector based on a source item. Recall that we store the interactions made by each user through time in S_u. Thus, for a user u we compute relevance vectors based on items that the user has interacted before through algorithm $\text{IGSI}_{\hat{\pi}^t}$, and average these vectors to recommend the k items with highest average relevance score. This procedure is outlined in Algorithm 10.

The choice of relevant items in S_u to infer user interest directly influences the scalability and accuracy of the underlying recommendations. Computing relevance scores for several items can easily lead to unfeasible recommendation time in data streams scenarios. Another relevant aspect is that user preferences change over time, therefore including older data may deteriorate accuracy. Hence, it might be practical to model users based only on a few recent events. To address these issues, we test three approaches to recommend items to users in order to verify the balance between accuracy and scalability.

To assess the influence of the most recent interaction on user interest, recommendations to a user u are generated based only on the last item interacted by u, i.e., li_u. We refer to the ranking algorithm based in li_u as $\text{IGSI}^{li_u}_{\hat{\pi}^t}$. Note that this approach results in non-personalized

Algorithm 10 Recommend k items

Require:
u: active user
S_u: ordered list of interactions by u
r: number of most recent items to consider
k: number of items to recommend
$(\mathcal{V}, \mathcal{E}, w)$: weighted directed graph
t: length of walks
M: number of walks
γ: probability of continuing the random walk

1: **procedure** RecommendItems($u, S_u, r, k, (\mathcal{V}, \mathcal{E}, w), t, M, \gamma$)
2: rS_u = last r elements in S_u
3: **for** $i \in rS_u$ **do**
4: h_i = RandomWalk($v_i, (\mathcal{V}, \mathcal{E}, w), t, M, \gamma$) // Algorithm 9
5: **for** v $\in \mathcal{V}$ **do**
6: **if** v $\notin \mathcal{S}_u$ **then**
7: c[v] = $\sum_{i \in rS_u} h_i[v] * \frac{1}{|rS_u|}$
8: **return** top k items in c

recommendations, and simply recommends the higher ranked items as inferred by random walks started in li_u.

To model long-term interest, recommendations to a user u are generated based on the entire list of interactions S_u. We refer to this ranking algorithm based in S_u as $\mathrm{IGSI}^{S_u}_{\hat{\pi}^t}$.

To model short-term interest, assuming that user preferences change over time, we rank items based on the r most recent user interactions. We refer to this ranking algorithm as $\mathrm{IGSI}^{r}_{\hat{\pi}^t}$. The impact of hyperparameter r is evaluated through experiments reported in Section 6.2.4.

We note that for experimentation purposes, we store S_u in its entirety for each user in the dataset, to evaluate the impact of short-term and long-term interests in the recommendations. However, memory usage can be optimized by storing only the necessary interactions per user, e.g., its last r interactions.

Regarding space and time complexity of $\mathrm{IGSI}_{\hat{\pi}^t}$, the graph is implemented with an adjacency list, and its space complexity is $O(|\mathcal{V}| + |\mathcal{E}|)$. As far as time complexity, Algorithm 9 depends on the number of random walk samples M and the function in line 8, and its complexity is $O(M \cdot \log |a|)$, where a is the set of adjacent nodes, with $|a| \ll |\mathcal{V}|$. Algorithm 10 generates the recommendations based on Algorithm 9 and depends on hyperparameter r. Hence, the time complexity for $\mathrm{IGSI}_{\hat{\pi}^t}$ to rank items and compute the recommendations is $O(r \cdot M \cdot \log |a|)$, where recommendation times can be controlled through the number of samples M.

Algorithm 11 presents the online functioning of $\mathrm{IGSI}_{\hat{\pi}^t}$, our proposed SBRS. $\mathrm{IGSI}_{\hat{\pi}^t}$ processes each observation generated by the data stream sequentially, and performs two main operations: recommendation of items to the active user u, as defined in Algorithms 9 and 10, and an incremental update based on the previous interactions of u and the new interaction of u with item i, as defined in Algorithm 8. Such a scope allows for continuous evaluation of the model, following a test-then-learn scheme (Vinagre et al., 2021), i.e., first a recommendation is generated and evaluated, and then the observation is used to update the model. We describe the evaluation procedure, (line 4), in Chapter 6. We also present in Table 5.1 a summary of the hyperparameters of our model. Note that in addition to the incremental update in line 6, a forgetting mechanism

(line 5) can also be deployed. We next discuss the need for such a mechanism and our proposed technique.

Algorithm 11 Online $\text{IGSI}_{\hat{\pi}^t}$

Require:
$\mathcal{D} = \{(< u, i, t >)_1, ...\}$: data stream
$\mathcal{U}$: set of users
$\mathcal{I}$: set of items
S_u: list of interactions by u
li_u: last item interacted by u
$(\mathcal{V}, \mathcal{E}, w)$: weighted directed graph
r: recency parameter
k: recommendation list size
t: length of walks
M: number of walks
γ: probability of continuing the random walk

1: **procedure** $\text{IGSI}_{\hat{\pi}^t}$
2: **for** $< u, i, t > \in \mathcal{D}$ **do**
3: top_n_items = RECOMMENDITEMS(S_u, r, k, $(\mathcal{V}, \mathcal{E}, w)$, t, M, γ) // Algorithm 10
4: EVALUATE(top_n_items) // Chapter 6
5: $(\mathcal{V}, \mathcal{E}, w)$ = FORGETTINGMECHANISM // Optional
6: $\mathcal{U}$, $\mathcal{I}$, S_u, li_u, lt_u, $(\mathcal{V}, \mathcal{E}, w)$ = INCREMENTALGRAPHUPDATE // Algorithm 8

5.2 NOVEL FORGETTING TECHNIQUE WITH RANDOM WALK SAMPLING

The aforementioned $\text{IGSI}_{\hat{\pi}^t}$ model performs two main operations: updating its underlying graph, and generating recommendations based on up-to-date information. The update procedure is performed on a single edge in the graph, depending on the previously collected sample. If the relation between two items as inferred by the most recent sample is new, an edge is created between these two items; otherwise, edge connecting these two items already exists, and its weight is increased to reinforce its importance. The recommendation procedure then ranks candidate items through random walk sampling, such that nodes that are closer to the sources are assumed to be more relevant. While continuous inclusion of new information allows the model to stay up-to-date and immediately consider recent user preferences, which are likely to better reflect their present ones, such preferences are dynamic and change over time, hence subject to concept drift (Gama et al., 2014).

As such, information that is included in incremental manner may become obsolete, which results in negative impact both on scalability and accuracy. Whereas scalability is impacted by the continuous growth of the underlying graph, predictive performance is affected by the inclusion of obsolete concepts in the recommendation procedure. The potential inclusion of new edges based on every incoming observation leads to a graph that grows linearly to the number of generated samples, which in turns raises time required for generation of recommendations. While degrading accuracy is obviously undesirable, scalability issues may actually hinder the applicability of models in data stream settings.

While one of the proposed recommendation procedures is based on the most recent user interactions, as defined by a hyperparameter r, somewhat accounts for the short-term interests of users and adjusting the recommendations accordingly, the lack of mechanisms to remove obsolete

information means that $\text{IGSI}_{\hat{\pi}^t}$ is susceptible to the aforementioned concerns, as the incremental updates results in an ever-growing graph.

To avoid these issues, we propose a novel forgetting mechanism, designed to exploit the advantages of $\text{IGSI}_{\hat{\pi}^t}$, and that could be generalized and extended to other graph-based approaches (Cooper et al., 2014; Christoffel et al., 2015) or neighborhood-based ones (Miranda and Jorge, 2009).

$\text{IGSI}_{\hat{\pi}^t}$ updates the edge connecting li_u, the last interacted item by user u, to i, item of new interaction. The most straightforward way to apply forgetting would be to simply fade all neighbors of li_u before updating edge (li_u, i), as defined by Eq.(5.3):

$$w((li_u, s))_t = \alpha \cdot w((li_u, s))_{t-1} \tag{5.3}$$

where $\alpha \in [0, 1]$ is a fading factor that controls the rate of forgetting and s is a neighbor node of li_u. This process, which is performed locally, ensures that recent data is emphasized over older information, and avoids aggravation of sparsity issues, as forgetting only occurs for items that are guaranteed to be updated with new information, i.e., concepts are replaced only in the presence of newer ones.

Table 5.1: Summary of the hyperparameters of our model. Hyperparameters ρ, t, M and r relate to our SBRS described in Section 5.1. Hyperparameters α, β, τ and x relate to our forgetting technique described in Section 5.2.

Hyperparameter	Description	Values
ρ	Time window hyperparameter used in the incremental update procedure. Increases weight to be added to an edge if interactions occurred within a time window, and decreases based on user's inactivity. Time is measured in hours.	$\rho \in \mathbb{Z}^+$
t	Length of random walk. Used in both recommendation and forgetting processes.	$t \in \mathbb{Z}^+$
M	Number of independent t-step random walks samples. Higher values for M increases accuracy at the expense of recommendation time.	$M \in \mathbb{Z}^+$
r	Hyperparameter that filters user interactions. The r most recent interactions of users are sources of random walks used for both recommendation and forgetting processes.	$r \in \mathbb{Z}^+$
α	Fading factor that controls the rate of gradual forgetting.	$\alpha \in [0, 1]$
β	Diversity parameter that balances diversity and exploration. Used to score and emphasize items from the long-tail of item catalog.	$\beta \in [0, 1]$
τ	Acceptance parameter. Used to score and penalize items perceived as negative based on their acceptance ratio.	$\tau \in [0, 1]$
x	Controls the weight threshold $\phi = \alpha^x$. Can be seen as number of forgetting interactions in the neighborhood without reinforcement.	$x \in \mathbb{Z}^+$

However, this forgetting mechanism does not further distinguish items and simply fades those that are not reinforced by the current observation. Hence, we propose an improved forgetting function that, besides recency, scores items based on popularity and structural information, fading them proportionally to a predefined relevance score, as defined by Eq.(5.4):

$$w((p,s))_t = \alpha^{(1-score)} \cdot w((p,s))_{t-1} \tag{5.4}$$

where p and s are adjacent nodes in the graph and $score$ is a relevance score assigned to s from p, such that items with low score are subject to faster forgetting.

We consider popularity information in order to increase the diversity of recommendation lists and boost the inclusion of items from the *long-tail*, a known issue in recommender systems (Celma and Cano, 2008). In recommender systems, the majority of interactions are related to a small fraction of most popular items, while the remaining large portion of the catalog, the *long-tail* of the distribution, only account for a small fraction of interactions (Cremonesi et al., 2010).

We measure item popularity through node degree information, and are interested in items with both low indegree and high outdegree. Items with low indegree are likely to be either from the long-tail or new ones, while items with high outdegree encourage exploration of random walks and penalize sink nodes, i.e., nodes without outgoing connections.

We also score items based on an acceptance factor, measured by the ratio between the number of times an item is accepted (clicked) by users when recommended and the number of times it is recommended. The premise behind it is that popular items which are constantly ignored are assumed as irrelevant as opposed to unknown to users (He et al., 2016). Hence we define the score for a given item in Eq.(5.5):

$$score = (\beta x_{s,p} + (1-\beta) y_{s,p}) \cdot (z^{\tau}_{s,p}) \tag{5.5}$$

where $\beta \in [0,1]$ is a diversity hyperparameter and a convex combination that balances diversity and exploration, with emphasis on items from the long-tail that increase the exploration of random walks and $\tau \in [0,1]$ is the acceptance hyperparameter. $x_{s,p}$ and $y_{s,p}$ are the indegree factor and outdegree factor of node s normalized according to the degree distribution of neighborhood of node p, respectively, and $z_{s,p}$ is the acceptance factor of item s normalized according to the acceptance factors of neighbors of node p. Indegree factor $x_{s,p}$ and outdegree factor $y_{s,p}$ are defined by Eq.(5.6) and Eq.(5.7), respectively:

$$x_{s,p} = \begin{cases} 0 & \text{if } \Delta^-(N^+(p)) = \delta^-(N^+(p)) \\ \frac{\Delta^-(N^+(p)) - deg^-(s)}{\Delta^-(N^+(p)) - \delta^-(N^+(p))} & \text{otherwise} \end{cases} \tag{5.6}$$

$$y_{s,p} = \begin{cases} 0 & \text{if } \Delta^+(N^+(p)) = \delta^+(N^+(p)) \\ \frac{deg^+(s) - \delta^+(N^+(p)}{\Delta^+(N^+(p)) - \delta^+(N^+(p))} & \text{otherwise} \end{cases} \tag{5.7}$$

where $deg^-(s)$ and $deg^+(s)$ are the indegree and outdegree of node s, respectively, $N^+(p)$ is the set of neighbors of node p, $\Delta^-(N^+(p))$ and $\delta^-(N^+(p))$ are the maximum and minimum indegree among neighbors of p, respectively, and $\Delta^+(N^+(p))$ and $\delta^+(N^+(p))$ are the maximum and minimum outdegree among neighbors of p, respectively.

Finally, we use random walk sampling *not only to provide recommendations but also to infer the structural information of the graph*. To that end, while performing random walk sampling in order to recommend relevant items, we store these samples to update the model *after* receiving the actual observation $\langle u, i, t \rangle$.

Denote by $RW_{p,t} = \{RW^1_{p,t}, RW^2_{p,t}, ..., RW^M_{p,t}\}$ as the set of M t-step random walk samples starting from node p, where $RW^k_{p,t} = \langle v_0, v_1, ..., v_t \rangle$ is the random sequence of nodes generated by the k-th t-step random walk started from p. In the end, we are interested in nodes that are on paths from the M t-step random walks started from p that reached the desired item i, defined as $RW'_{p,t} = \{v \in RW^k_{p,t} : k = \{1, ..., M\}, i \in RW^k_{p,t}\}$. Items s not in any path that reached i, i.e., $s \notin RW'_{p,t}$ are deemed irrelevant and are faded based on Eq.(5.4), while items in paths that reached i, i.e., $s \in RW'_{p,t}$ are considered relevant, and emphasized based on Eq.(5.8):

$$w((p,s))_t = \alpha^{-score} \cdot w((p,s))_{t-1} \tag{5.8}$$

In this way, we avoid aggravation of sparsity issues and ensure that relevant and recent information is kept in the graph, as perceived important connections are increased and irrelevant ones are faded as induced by the walks, by popularity factors and by recent observations from the data stream. The application of our proposed forgetting mechanism on $\text{IGSI}_{\hat{\pi}^t}$ is illustrated in Figure 5.3.

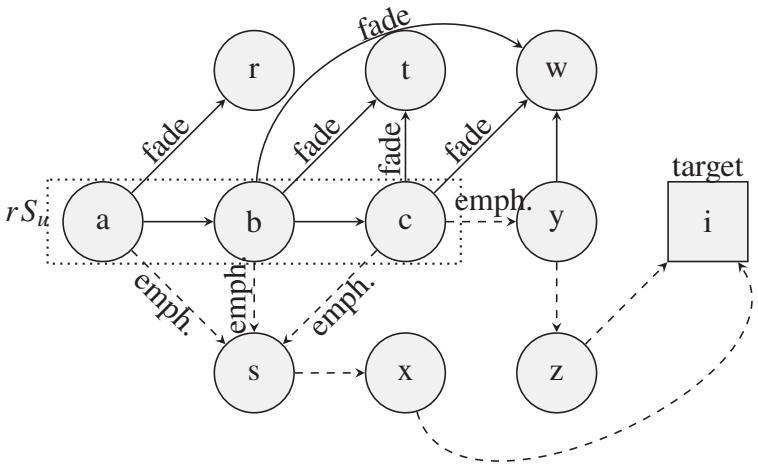


Figure 5.3: Application of our proposed forgetting mechanism on $\text{IGSI}_{\hat{\pi}^t}$. We omit edge weights for ease of visualization and instead notate where to emphasize and where to fade. The example considers a given user u whose most recent interactions are with items a, b and c, i.e., $rS_u =< a, b, c >$, as highlighted by a dotted black square. Thus, a recommendation would consist in performing $t-$step random walks from these source nodes. Assuming that the next interaction from u is with item i, we wish to emphasize neighbors of nodes in rS_u that reached i as inferred by the samples with Eq.(5.8) and fade neighbors that did not reach i with Eq.(5.4). Assuming $t = 3$, paths to i from nodes in rS_u are highlighted with dashed edges. In this example i is reachable from a through s, thus edge (a, s) is emphasized and its remaining neighbor r is faded; as i is also reachable from b through s, edge (b, s) is emphasized and all its remaining neighbors t and w are faded; and as i is reachable from c through nodes s and y, edges (c, s) and (c, y) are emphasized, while the remaining neighbors t and w are faded.

As this process ensures that irrelevant information is continuously faded unless reinforced by new data, hence ensuring that relevant information is retained in the neighborhood of nodes, we introduce a weight threshold $\phi = \alpha^x$ that removes obsolete edges proportionally to the fading factor α to account for the increasing scalability concerns related to ever-growing models, where x is a hyperparameter that controls the weight threshold based on α, and can be seen as the number of forgetting interactions in the neighborhood without reinforcement. This way, the growth of the graph can be controlled parametrically.

Thus, our forgetting technique requires four hyperparameters: fading factor α, diversity hyperparameter β, acceptance hyperparameter τ and x that controls weight threshold ϕ. We refer

to this technique as *local neighborhood decay* (**LND**). The online procedure of $\text{IGSI}_{\hat{\pi}^t}$ with our proposed forgetting mechanism LND is presented in Algorithm 12.

When a new observation $\langle u, i, t \rangle$ arrives from the data stream, we first produce a recommendation to u for evaluation purposes, and also store the random walk samples generated in this process in $RW_{rS_u,t}$. Then, *after* evaluation, we apply forgetting over rS_u, the last r interacted items by user u that are used as sources in the random walks. Neighbors of nodes in rS_u that are in any path that reached i as inferred by the samples, $RW'_{p,t}$, are emphasized with Eq.(5.8), while nodes that are not in $RW'_{p,t}$ are faded based on Eq.(5.4). Then, obsolete edges are removed if necessary as defined by the weight threshold. Finally, we update the graph with the new observation, as described in Section 5.1.

Algorithm 12 Online local neighborhood decay forgetting

Require:
$\mathcal{D} = \{(< u, i, t >)_1, ...\}$: data stream
$\mathcal{U}$: set of users
$\mathcal{I}$: set of items
S_u: list of interactions by u
li_u: last item interacted by u
$(\mathcal{V}, \mathcal{E}, w)$: weighted directed graph
r: recency parameter
k: recommendation list size
t: length of walks
M: number of walks
α: decay parameter
β: diversity parameter
τ: acceptance parameter
x: parameter for edge removal

1: **procedure** IGSI ONLINE UPDATE
2: **for** $< u, i, t > \in \mathcal{D}$ **do**
3: top_n_items, $RW_{rS_u,t}$ = RECOMMENDITEMS(S_u, r, k, $\mathcal{V}$, $\mathcal{E}$, t, M) // Algorithm 10
4: EVALUATE(top_n_items)
5: **for** $p \in rS_u$ **do** // Forgetting, for last r items interacted by u
6: **for** $s \in neighbors(p)$ **do**
7: **if** $s \notin RW'_{p,t}$ **then**
8: $w((p,s))_t = \alpha^{(1-score)} \cdot w((p,s))_{t-1}$ // Eq.(5.4)
9: **if** $w((p,s)) < \alpha^x$ **then**
10: $E \leftarrow E \setminus (p,s)$
11: **else**
12: $w((p,s))_t = \alpha^{-score} \cdot w((p,s))_{t-1}$ // Eq.(5.8)
13: $\mathcal{U}, \mathcal{I}, S_u, li_u, lt_u, (\mathcal{V}, \mathcal{E}, w)$ = INCREMENTALGRAPHUPDATE // Algorithm 8

5.3 DISCUSSION

In this chapter we presented our proposal, which aims to overcome the issues related to recommendation and data stream mining, such as sparsity, accuracy, scalability and adaptability. Our contribution is an SBRS based on an evolving item-graph that extracts information from sequential data, i.e., sequential user interactions, and allows the recommendation of items based

on short-term and long-term interests in scalable manner through the use of simulation of random walks. Our proposal also includes a forgetting mechanism that reuses the random walk samples generated during the recommendation process. By reusing these samples, we reduce computational costs, while being able to infer structural information from the graph in order to score items based on their relevance, with minor additional processing. We also score items based on its diversity and acceptance from users, fade connections between items based on these scores and remove these connections when they become obsolete. In the next chapter we evaluate our proposed SBRS method and the impact of our proposed forgetting method.

6 EXPERIMENTS

In this chapter we report the experiments performed to evaluate the proposed approach. First, we describe the evaluation protocol, the datasets and evaluation metrics. Then, we split the evaluation in two parts. The first part is concerned with evaluating our proposed stream-based recommender system, $\text{IGSI}_{\hat{\pi}^t}$, by comparing it with other competing algorithms under several metrics. The second aims to evaluate the impact of our proposed forgetting mechanism, LND. We extend $\text{IGSI}_{\hat{\pi}^t}$ with LND and assess its impact on scalability, accuracy and diversity, and also perform comparisons with other forgetting mechanisms. Besides the evaluation of $\text{IGSI}_{\hat{\pi}^t}$ and LND under the constrains of data stream settings, the experimental analysis provided in this chapter is designed to address the research questions posed in Chapter 1:

1. *Can graph-based algorithms for collaborative filtering be used effectively in data stream scenarios? How do they perform in scalability and accuracy compared to other incremental models?*

2. *How can time-related information be incorporated into graph-based models? Does such information improve the quality of the recommendations?*

3. *Are current incremental algorithms sufficient in providing accurate, diverse and scalable recommendations? Can graph-based approaches effectively provide recommendations that are relevant in practice?*

4. *Considering that learning incrementally based on continuous incoming data leads to ever-growing models, what techniques can be used to remove obsolete information from these models? Are evolving methods enough to deal with issues inherently related to recommender systems and data stream mining, such as concept drift*?

6.1 EVALUATION SETTING

In order to evaluate the proposed approach on a data stream, a suitable evaluation methodology is required. We employ the prequential evaluation protocol proposed by Vinagre et al. (2014a, 2021), which is a test-then-learn procedure that performs evaluation at every observation before it is used to update the model, thus allowing continuous online monitoring of the predictive ability. In other words, this protocol evaluates how well the algorithm predicts the next observation. For each incoming event $\langle u, i, t \rangle$, the model is first tested and then updated based on the following steps:

1. If u is a known user, use the current model to recommend N items to u, otherwise go to step 3;

2. Score the recommendation list given the observed item i;

3. Update the model with the observed event;

4. Proceed to the next observation.

The aforementioned protocol offers several advantages, mainly, the continuous monitoring of a model's performance, evaluation of several metrics simultaneously, and allows for straightforward reproducibility if the same data sequence is available (Vinagre et al., 2014a).

6.1.1 Datasets

We use nine datasets from several domains in our experiments, as summarized in Table 6.1. The two movie-domain datasets, ML-1M and ML-10M, are binary versions of the MovieLens-1M[1] and MovieLens-10M[2] movie rating datasets (Harper and Konstan, 2015), respectively. These datasets are composed of several observations in the form of $\langle u, i, r, t \rangle$, indicating that a given user u attributed rating r to item i at time t. As these are explicit feedback datasets, pre-processing is required to ensure its usage in our setting. To that end, we discarded ratings below 5 and sorted events chronologically, thus simulating continuous implicit feedback, with events defined as $\langle u, i, t \rangle$.

We use four music-domain datasets: PLC-PL, PLC-STR, LFM and LFM-1K. The PLC-PL and PLC-STR datasets[3] were extracted from the social network Palco Principal by Vinagre et al. (2014b). PLC-STR consist of a timestamped log of music listening events, where each incoming pair corresponds to a music track being listened to by the user. PLC-PL consist of a timestamped log of track additions to personal playlists. LFM consist of the first 8 months of events observed from the Last.fm[4] dataset collected by Celma (2010), while LFM-1K are the first 25% of events observed from the same original dataset. We perform no pre-processing in these four datasets, only sorting events chronologically. We note that these datasets include repeated interactions, i.e., a user may interact more than once with the same item. We skip the evaluation in such cases but still use the events to update the models.

The two e-commerce datasets, BOOK and ELEC, are binary versions of categories *books* and *electronics* from the Amazon dataset[5] (McAuley et al., 2015). The dataset consists of several reviews from Amazon, spanning 18 years of activity up to March 2013, which include product and user information, ratings, and a review. Similarly to ML-1M and ML-10M, these are explicit feedback datasets, and we adapt BOOK and ELEC to our setting. The adaptation is done by discarding ratings below 5 and sorting events chronologically, simplifying all events to the form $\langle u, i, t \rangle$.

Finally, we use a news domain dataset, GLOBO, extracted from news portal globo.com[6] by de Souza Pereira Moreira et al. (2018, 2019). In such a domain, the set of items goes through constant churn, new items are continuously added to it and old ones are dropped. In other words, in this domain items have a very short shelf life, and become old very quickly (Das et al., 2007). Also, the user-item matrix is very sparse, and are prone to popularity factors, such as breaking news or popular topics (Lommatzsch and Albayrak, 2015). We perform minimal pre-processing on this dataset, only sorting events chronologically. We note that GLOBO is originally a session-based dataset. We disregard session information in our experiments and only consider user information, assuming that events are tuples in the form $\langle u, i, t \rangle$.

6.1.2 Metrics

We measure accuracy through metrics HitRate@N (HR@N) and Discounted Cumulative Gain (DCG@N). HitRate@N returns 1 if item i is within the N first recommended items, and 0 otherwise. DCG@N measures the quality of the ranking by considering the position of a correct

[1] https://grouplens.org/datasets/movielens/1m/
[2] https://grouplens.org/datasets/movielens/10m/
[3] https://rdm.inesctec.pt/dataset/cs-2017-003
[4] https://last.fm
[5] http://snap.stanford.edu/data/web-Amazon-links.html
[6] https://globo.com

Table 6.1: Dataset description.

Dataset	Domain	Events	Users	Items	Sparsity
ML-1M	Movies	226.310	6.014	3.232	98.84%
ML-10M	Movies	1.544.812	67.312	8.721	99.74%
PLC-PL	Music	111.942	10.392	26.117	99.96%
PLC-STR	Music	1,466,893	25,463	40,213	99.92%
LFM	Music	493.067	164	65.010	99.11%
LFM-1K	Music	4,234,033	546	399,171	99.46%
BOOK	E-commerce	6,278,141	1,827,029	734,918	99.99%
ELEC	E-commerce	618,560	451,486	60,842	99.99%
GLOBO	News	1,830,308	259,690	35,644	99.98%

item prediction in the recommendation list. For prequential evaluation, DCG@N is defined as (Frigó et al., 2017):

$$DCG@N = \begin{cases} \frac{1}{\log_2(rank(i)+1)} & \text{if } rank(i) \leq N \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

where i is the positive item and $rank(i)$ denotes the position of i in the recommendation list of size N. As we use several datasets from different domains for evaluation, we set N = 20 for all subsequent experiments, and use metrics HR@N and DCG@N in complementary manner: HR@N measures if the evaluated algorithm was able to recommend the target item, while DCG@N considers the actual position of the item in the recommendation list.

We measure scalability through average update time and recommendation time per sample.

We also evaluate diversity through metric Intra-List Diversity (ILD) (Smyth and McClave, 2001). ILD of a given recommendation list is defined as the average pairwise distance of items in the list (Castells et al., 2015):

$$ILD = \frac{1}{|R|(|R|-1)} \sum_{i \in R} \sum_{j \in R} d(i,j) \tag{6.2}$$

where R is the recommendation list and $d(i,j)$ is a given distance measure between items i and j. In this work, we use the cosine distance:

$$d(i,j) = 1 - \frac{|U_i \cap U_j|}{\sqrt{|U_i|} \times \sqrt{|U_j|}} \tag{6.3}$$

where U_i and U_j denote the set of users that interacted with items i and j, respectively.

To assess the statistical significance of the results, we use the McNemar test over a sliding window as described by Gama et al. (2009); Vinagre et al. (2014a, 2021). The prequential process produces several metrics, including a sequence of hit rate values, representing the learning process of the models, and the average hit rate can be continuously reported through the use of sliding windows. Each window produces a sequence of hit rate $\in \{0, 1\}$ of size n. Thus, the test can be used in each window to compare the performance of different models. Given two algorithms A and B, the test requires computing two quantities: n_{10}, denoting the

number of observations correctly classified by algorithm A and not by B, and n_{01}, where the opposite situation happens. At every step of the prequential process, the test can be performed by calculating the statistic:

$$Mcn = \frac{(n_{10} - n_{01})^2}{n_{10} + n_{01}} \tag{6.4}$$

where Mcn follows a χ^2 distribution with 1 degree of freedom. For a significance level of $\alpha = 0.01$, the critical value is $Mcn = 6.635$.

6.1.3 Experimental setup

We split our experimental setup into two distinct parts. We want to evaluate the performance of our incremental proposal, as described in Section 5.1, and also assess the performance of our proposed forgetting technique, defined in 5.2. As our proposed forgetting technique is designed to extend our incremental SBRS, IGSI_{π^t}, we first evaluate IGSI_{π^t} in Section 6.2, and then evaluate the impact of forgetting in Section 6.3.

In the first experimental analysis, we evaluate the performance of our incremental SBRS, IGSI_{π^t}, comparing it to several algorithms defined in Chapters 3 and 4, under several metrics. In this experimental setting, we built initial models on the first 10% of each dataset, and use the remaining 90% for prequential evaluation, as suggested by Vinagre (2016). The reasoning for such split is that we want to simulate a data stream as realistically as possible, and assess the incremental capabilities of each algorithm. Thus, we use only a small part of the original dataset, in this case 10%, to built initial models with batch training and avoid cold-start issues, and use the remaining data to simulate a data stream.

We also evaluate the impact of each hyperparameter. Recall that IGSI_{π^t} has four: the number of steps in a random walk t, the time window ρ, number of recent items r to generate recommendations and number of independent t-step random walk samples M. We optimize hyperparameters on the first 10% of the dataset that is used to initialize the models. We further divide these 10% in another 10% to built the model and use the remaining 90% for validation. The impact of each hyperparameter is discussed before we compare IGSI_{π^t} with other incremental approaches.

After discussing the results of IGSI_{π^t}, we extend it with our proposed forgetting technique. Such extension results in an *evolving* model that learns continuously and incrementally with newly generated data, and also removes information that is deemed obsolete. We also compare our proposal with other forgetting techniques, discussed in Section 4.1.

For this second experimental setting, we use a different dataset split. We conducted initial experiments in the original 10-90 split. However, we verified that with such split forgetting was deployed too quickly, as the models initially lacks data and are not stable enough. We thus adopt the dataset split procedure proposed by Matuszyk et al. (2018), which is used to evaluate forgetting: the first 20% are used to built initial models; the following 30% to optimize hyperparameters and the remaining 50% for prequential evaluation. Similarly to the incremental evaluation, we first assess the impact of each forgetting hyperparameter: fading factor α, diversity parameter β, acceptance parameter τ and x that controls weight threshold ϕ. Then, we compare the performance of our proposal to other forgetting techniques.

All subsequent algorithms were implemented in Python 3 and the experiments were executed on an Intel Core i7-4770 of 3.4 GHz with 16 GB RAM running Ubuntu.

6.2 EVALUATION OF IGSI

In this section, we evaluate our proposed SBRS, IGSI_{π^t}, as defined in Section 5.1. The goal of this section of experiments is to assess the influence of its different hyperparameters, and compare its performance under several metrics with related incremental algorithms.

6.2.1 Related algorithms

We compare the performance of our proposed approach with two non-personalized baselines, four incremental algorithms designed to work with data streams and two graph-based algorithms that although not evaluated in data stream settings, can work incrementally and we have adapted them to our setting. We limit the evaluation to algorithms that can be updated based on each incoming observation, that is then discarded, in acceptable time. This eliminates complex algorithms that either require storing training data or whose update procedure incurs infeasible processing time.

1. **TOP-POP**: Top popular (TOP-POP) is a non-personalized recommender systems that simply recommends the N most popular items available in the catalog (Cremonesi et al., 2010), which disregards preferences of the active user. It has no hyperparameters and is very straightforward to implement.

2. **RE-POP**: *Recently popular* (RE-POP) is another non-personalized, popularity-based strategy. The difference from TOP-POP is that RE-POP recommends the N most popular items in the catalog considering events that occurred in the last pre-defined time window, which is the only hyperparameter required by RE-POP. Despite its conceptual simplicity, this method has been shown to be effective in news recommendation scenarios (Jugovac et al., 2018).

3. **ItemKNN**: An incremental version of the traditional item-based nearest-neighbors algorithm (Sarwar et al., 2001) for implicit feedback proposed by Miranda and Jorge (2009) and described in Section 3.1. This algorithm recommends items by searching for the k most similar items in terms of user preference based on cosine similarity. Hyperparameter k defines the neighborhood size. We tested values for $k \in [10, 100]$ in steps of 10. We defined $k = 10$ for datasets LFM and GLOBO, $k = 20$ for datasets PLC-PL, PLC-STR and ML-1M, $k = 40$ for ML-10M and $k = 100$ for ELEC.

4. **ISGD**: An incremental matrix factorization algorithm for positive-only feedback proposed in Vinagre et al. (2014b) and outlined in Section 3.2. ISGD decomposes the rating matrix R into a user matrix P and an item matrix Q covering a common k-dimensional latent feature space such that $R \approx PQ^T$, and performs single-pass stochastic gradient descent (SGD) updates on each incoming observation. Hyperparameters for ISGD are learning rate, regularization, number of updates for each observation and number of latent factors k. We adopt the optimal hyperparameters defined in Vinagre (2016).

5. **RAISGD**: An extension of ISGD proposed in Vinagre et al. (2015a) and described in Section 3.2. RAISGD introduces negative feedback into ISGD by maintaining a global queue of all items seen in the stream, ordered according to the recency of item occurrences. For every new positive observation in the stream, the l oldest items are selected from the queue to be considered as negative feedback by the user. In essence, RAISGD gives greater importance to more recent events. Besides the hyperparameters of ISGD, RAISGD has an extra hyperparameter l, which is the number of items from

the queue to be selected as negative feedback. We adopt the values for l defined in Vinagre (2016).

6. **BPR-MF**: An incremental version of algorithm BPR-MF (Rendle et al., 2009), as described in Sections 2.3.2.2 and 3.2. BPR-MF learns a personalized total ranking $>_u \subset \mathcal{I} \times \mathcal{I}$ for each user u, under the assumption that all items i that were consumed by u precede all items j that were *not* consumed by u in the total ranking. BPR-MF is modeled to optimize the posterior distribution $p((\mathbf{V}, \mathbf{W})| >_u)$, where $\mathbf{V}$ and $\mathbf{W}$ represent user and item latent factors, respectively, which are the parameters of the model. The incremental procedure for a new observation $\langle u, i, t\rangle$ uniformly samples an unseen item j by the active user u to be used as negative feedback, and updates latent factors for user u ($\mathbf{v}_u$) and items i and j ($\mathbf{w}_i$ and $\mathbf{w}_i$, respectively) based on the relative ranking of i and j by u as predicted by Eq.(2.28). Hyperparameters for BPR-MF are learning rate, regularization factors for u, i and j, number of updates for each observation and number of latent factors k. We adopt the optimal hyperparameters defined in Vinagre (2016).

7. $\hat{\mathbf{P}}^3_\alpha$: An approximation based on random walk sampling of algorithm P^3_α (Cooper et al., 2014), as described in Section 3.1.1. P^3_α is a ranking algorithm based on short random walks in a bipartite user-item graph (Section 2.3.1.1). Items for user u are ranked based on the transition probability of a random walk with three steps starting from node u, which is equivalent to computing matrix P^3 (Eq.(2.11)). Algorithm P^3_α raises each transition probability to the power of a parameter α, which improves the performance in comparison to P^3. We tested values for $\alpha \in [0, 2.0]$ in steps of 0.1. For datasets PLC-PL, PLC-STR, LFM, GLOBO and ML-1M we set $\alpha = 1.1$, $\alpha = 1.7$ for ELEC and $\alpha = 2.0$ for ML-10M.

8. $\hat{\mathbf{RP}}^3_\beta$: An approximation based on random walk sampling of algorithm RP^3_β (Christoffel et al., 2015), described in Section 3.1.1. RP^3_β is a re-ranking of algorithm P^3 based on item-popularity (Eq.(2.19)). RP^3_β re-weights the scores given by P^3 by dividing each transition probability by the degree of the item raised to the power of a parameter β. This procedure essentially reduces the score of popular items and increases the score of items in the long-tail. If $\beta = 0$, the algorithm is equivalent to P^3. We tested values for $\beta \in [0, 2.0]$ in steps of 0.1. We defined $\beta = 0.1$ for dataset LFM, GLOBO and ELEC, $\beta = 0.2$ for datasets PLC-PL, PLC-STR and ML-1M, and $\beta = 0.4$ for ML-10M.

6.2.2 Effect of step-size parameter

We first analyze the performance of algorithm $\text{IGSI}^{li_u}_{\pi^t}$ with varying step-sizes t. In this set of experiments, recommendations are generated with random walks starting on the last item interacted by the user. We tested values for $t \in [1, 5]$ in steps of 1. The results of these experiments for all datasets are presented in Tables 6.2 and 6.3, with accuracy measured by HR@20 and DCG@20. We also evaluated the statistical significance of the results based on the increasing values of t.

From Tables 6.2 and 6.3, we can see that for algorithm $\text{IGSI}^{li_u}_{\pi^t}$, with the exception of datasets ML-10M and GLOBO, accuracy increases with t, where $t = 5$ presents the highest HR@20 results, and $t = 2$ presents the best results for ML-10M and GLOBO. We can also see that accuracy increases considerably comparing $t = 1$ to $t = 4$, with minor improvements afterwards that are not statistically significant. HR@20 highlighted in bold indicate the highest value for t that is superior with statistical significance in comparison to lower values.

Table 6.2: Impact of parameter step-size t in the accuracy of algorithm IGSI$_{\pi^t}$ for datasets ML-1M, ML-10M, PLC-PL and PLC-STR. Accuracy is measured by HitRate@20 (HR) and DCG@20 (DCG). HR@20 highlighted in bold indicate the highest value for t that is superior with statistical significance in comparison to lower values.

	ML-1M		**ML-10M**		**PLC-PL**		**PLC-STR**	
t	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**
1	0.111	0.055	0.564	0.285	0.178	0.161	0.429	0.355
2	0.152	0.070	**0.588**	**0.293**	0.226	0.186	0.496	0.381
3	0.159	0.073	0.584	0.290	0.250	0.196	0.512	0.384
4	**0.164**	**0.074**	0.577	0.286	**0.262**	**0.199**	**0.517**	**0.384**
5	0.167	0.075	0.567	0.281	0.265	0.199	0.519	0.382

Table 6.3: Impact of parameter step-size t in the accuracy of algorithm IGSI$_{\pi^t}$ for datasets LFM, ELEC and GLOBO. Accuracy is measured by HitRate@20 (HR) and DCG@20 (DCG). HR@20 highlighted in bold indicate the highest value for t that is superior with statistical significance in comparison to lower values.

	LFM		**ELEC**		**GLOBO**	
t	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**
1	0.022	0.02	0.394	0.313	0.649	0.348
2	0.027	0.022	0.427	0.330	**0.676**	**0.356**
3	**0.030**	**0.023**	0.438	0.334	0.673	0.352
4	0.031	0.023	**0.448**	**0.337**	0.668	0.346
5	0.032	0.023	0.448	0.337	0.663	0.340

These results suggest that short step-sizes are sufficient in providing relevant recommendations. Thus, while noting that minor improvements in accuracy could be obtained with larger step-sizes as reported in Tables 6.2 and 6.3, we set $t = 3$ for all datasets for scalability reasons, which is a major factor in data streams.

6.2.3 Effect of time window parameter

Next, we evaluate the impact of time window parameter ρ. Recall from Section 5.1 that Eq.(5.1) updates edge weights incrementally, and gives more weight to the edge if the interactions occurred within a time window and decreases the weight to be added based on users' inactivity. The idea of this parameter is to emphasize interactions that occurred within a time window as set by ρ. We tested values (in hours) for $\rho \in \{1, 2, 3, 6, 12, 24, 48, 168, \infty\}$. Results of these experiments for all datasets are presented in Tables 6.4 and 6.5.

From Tables 6.4 and 6.5, there is generally no difference between varying values for ρ. When there are differences, we see that accuracy increases with ρ, as is the case for dataset GLOBO, but these differences are not statistically significant. We believe that these results relate to the sparsity of datasets. It seems that there is no clear advantage in differentiating information that is already very sparse. Hence, for subsequent experiments, we simply set $\rho = \infty$ for all datasets, which in essence does not consider time between interactions and simply increases the weight based on the number of sequential interactions.

Table 6.4: Impact of parameter ρ in the accuracy of algorithm $\text{IGSI}^{li_u}_{\pi^t}$ for datasets ML-1M, ML-10M, PLC-PL and PLC-STR. Accuracy is measured by HitRate@20 (HR) and DCG@20 (DCG).

	ML-1M		**ML-10M**		**PLC-PL**		**PLC-STR**	
ρ	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**
1	0.160	0.073	0.583	0.290	0.250	0.196	0.512	0.385
2	0.160	0.073	0.583	0.290	0.250	0.197	0.512	0.385
3	0.160	0.073	0.583	0.290	0.250	0.196	0.512	0.385
6	0.160	0.073	0.583	0.290	0.250	0.196	0.512	0.385
12	0.160	0.073	0.583	0.290	0.250	0.196	0.512	0.385
24	0.160	0.073	0.583	0.290	0.250	0.196	0.512	0.385
48	0.160	0.073	0.584	0.290	0.250	0.196	0.512	0.385
168	0.160	0.073	0.584	0.290	0.250	0.196	0.512	0.385
∞	0.159	0.073	0.584	0.290	0.250	0.196	0.512	0.384

Table 6.5: Impact of parameter ρ in the accuracy of algorithm $\text{IGSI}^{li_u}_{\pi^t}$ for datasets LFM, ELEC and GLOBO. Accuracy is measured by HitRate@20 (HR) and DCG@20 (DCG).

	LFM		**ELEC**		**GLOBO**	
ρ	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**
1	0.030	0.023	0.425	0.327	0.657	0.344
2	0.030	0.023	0.425	0.327	0.662	0.346
3	0.030	0.023	0.425	0.327	0.664	0.348
6	0.030	0.023	0.425	0.327	0.669	0.350
12	0.030	0.023	0.425	0.327	0.672	0.351
24	0.030	0.023	0.426	0.328	0.673	0.352
48	0.030	0.023	0.427	0.328	0.673	0.352
168	0.030	0.023	0.429	0.329	0.673	0.352
∞	0.030	0.023	0.438	0.334	0.673	0.352

6.2.4 Effect of recency parameter

As discussed in Section 5.1.3, we propose the generation of recommendations to a user u based on the r most recent interactions of u in order to evaluate the impact of short-term interests in recommendations. To that extent, we conducted experiments to evaluate the impact of r in the accuracy of algorithm $\text{IGSI}^{r}_{\pi^t}$ with metrics HitRate@20 and DCG@20. We also evaluated the statistical significance of the results based on the increasing values of r. We tested values for $r \in [1, 10]$ in steps of 1. The results of these experiments for all datasets are outlined in Tables 6.6 and 6.7. HR@20 highlighted in bold indicate the highest value for r that is superior with statistical significance in comparison to lower values.

As shown in Tables 6.6 and 6.7, we can see that for all datasets accuracy tends to slightly increase as r increases until reaching a threshold, and after that it tends to decrease. We can also see that after a certain value for r, the increase in HR@20 is not statistically significant, and the values for r that present statistically significant results for HR@20 also present the highest values for DCG@20. These results suggest that including outdated information can be detrimental to the accuracy of the algorithm, and modeling user interest on a few recent events can lead to

Table 6.6: Impact of parameter r in the accuracy of algorithm $\text{IGSI}^{r}_{\pi^t}$ for datasets ML-1M, ML-10M, PLC-PL and PLC-STR. Accuracy is measured by HitRate@20 (HR) and DCG@20 (DCG). HR@20 highlighted in bold indicate the highest value for r that is superior with statistical significance in comparison to lower values.

	ML-1M		**ML-10M**		**PLC-PL**		**PLC-STR**	
r	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**
1	0.159	0.073	0.584	0.290	0.250	0.196	0.512	0.384
2	0.179	0.078	**0.606**	**0.298**	0.278	0.201	0.525	0.383
3	0.186	0.079	0.602	0.290	0.287	0.214	**0.527**	**0.385**
4	**0.190**	**0.079**	0.592	0.280	0.292	0.216	0.526	0.366
5	0.189	0.078	0.579	0.270	**0.294**	**0.216**	0.524	0.359
6	0.186	0.077	0.568	0.261	0.295	0.215	0.522	0.353
7	0.185	0.076	0.557	0.254	0.296	0.213	0.520	0.347
8	0.181	0.074	0.546	0.247	0.296	0.212	0.518	0.343
9	0.180	0.073	0.538	0.242	0.295	0.211	0.516	0.34
10	0.178	0.072	0.530	0.238	0.295	0.210	0.515	0.337

Table 6.7: Impact of parameter r in the accuracy of algorithm $\text{IGSI}^{r}_{\pi^t}$ for datasets LFM, ELEC and GLOBO. Accuracy is measured by HitRate@20 (HR) and DCG@20 (DCG). HR@20 highlighted in bold indicate the highest value for r that is superior with statistical significance in comparison to lower values.

	LFM		**ELEC**		**GLOBO**	
r	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**
1	0.030	0.023	0.438	0.334	0.673	0.349
2	0.035	0.025	0.445	0.338	**0.676**	**0.352**
3	0.037	0.026	**0.447**	**0.339**	0.671	0.344
4	**0.039**	**0.026**	0.446	0.338	0.668	0.340
5	0.039	0.026	0.446	0.338	0.666	0.339
6	0.039	0.026	0.446	0.338	0.665	0.338
7	0.040	0.026	0.446	0.338	0.664	0.337
8	0.040	0.026	0.446	0.338	0.664	0.337
9	0.040	0.026	0.446	0.338	0.663	0.336
10	0.040	0.025	0.446	0.338	0.663	0.336

improvement in accuracy. For overall results presented in Section 6.2.6, we set $r = 2$ for ML-10M and GLOBO, $r = 3$ for PLC-STR and ELEC, $r = 4$ for LFM and ML-1M and $r = 5$ for PLC-PL.

6.2.5 Sampling approximation

In order to verify the performance of the sampling approach, described in Section 5.1.1, we executed experiments with the sampling algorithm $\text{IGSI}^{li_u}_{\hat{\pi}^3}$ with varying number of random walks per recommendation $M \in \{100, 250, 500, 1000, 2500, 5000, 10000, 12500, 15000, 20000\}$ and compared the HR@20 of the approximations with the exact algorithm $\text{IGSI}^{li_u}_{\pi^3}$. The results of these experiments are presented in Table 6.8.

From Table 6.8, we can see that as expected, the accuracy of $\text{IGSI}^{li_u}_{\hat{\pi}^3}$ increases with the number of samples, and for all datasets, it reaches the same HitRate@20 of the exact algorithm

Table 6.8: Accuracy of algorithm $\text{IGSI}^{li_u}_{\hat{\pi}^3}$ with increasing number of random walks M for all datasets. Accuracy is measured by HitRate@20.

	ML-1M	ML-10M	PLC-PL	PLC-STR	LFM	ELEC	GLOBO
M	**HR@20**	**HR@20**	**HR@20**	**HR@20**	**HR@20**	**HR@20**	**HR@20**
100	0.143	0.528	0.247	0.502	0.030	0.429	0.645
250	0.150	0.558	0.250	0.508	0.030	0.436	0.662
500	0.152	0.571	0.250	0.510	0.030	0.438	0.668
1000	0.156	0.578	0.250	0.511	0.030	0.437	0.671
2500	0.157	0.581	0.250	0.511	0.030	0.437	0.672
5000	0.158	0.583	0.250	0.512	0.030	0.437	0.673
10000	0.159	0.584	0.250	0.512	0.030	0.438	0.673
12500	0.159	0.584	0.250	0.512	0.030	0.438	0.673
15000	0.159	0.584	0.250	0.512	0.030	0.438	0.674
20000	0.159	0.584	0.250	0.512	0.030	0.438	0.673
∞	0.159	0.584	0.250	0.512	0.030	0.438	0.673

within 20000 samples. Since for all datasets the amount of samples required for the percentage deviation between the exact computation π^3 and the approximation $\hat{\pi}^3$ to be less than 1‰ is at least 5000 samples, for overall results presented next, we set the number of samples to the approximate algorithm $\hat{\pi}^3$ as $M = 5000$.

6.2.6 Overall results

After evaluating the impact of different parameters on the accuracy of our model $\text{IGSI}_{\hat{\pi}^t}$, we now evaluate our model on the entire datasets with the defined parameters and present overall results. We tested our model with the three recommendation strategies defined in Section 5.1.3 in order to evaluate the impact of long-term and short-term interests. We also present results for related incremental algorithms, outlined in Section 6.2.1. We also consider algorithm $\text{IGSI}^r_{P^t}$ (Section 5.1), with $t = 1$. Such algorithm is analogous to a frequent mining procedure that is easy to compute, as it depends only on the weights of adjacent nodes.

Tables 6.9, 6.10, 6.11 and 6.12 report overall results for all algorithms. Accuracy is measured through metrics HR@20 and DCG@20, diversity is reported through metric ILD and time is measured through average update and recommendation times, with the best results highlighted in bold. For algorithms $\hat{\text{P}}^3_\alpha$ and $\hat{\text{RP}}^3_\beta$, we set the number of samples to 5000, which is the same amount of samples that we set to our model.

As shown in Tables 6.9, 6.10, 6.11 and 6.12, $\text{IGSI}_{\hat{\pi}^3}$-based algorithms outperform the related algorithms for all datasets in terms of accuracy, except for ELEC for which it is second best, with competitive diversity, update and recommendation times. An interesting insight from the results is that generating recommendations based solely on the last item interacted by the user, which results in a non-personalized approach, outperforms recommendations based on the entire interaction history. In fact, for dataset GLOBO, it is the best resulting algorithm. This is related to both the impact of short-term interests and to the fact that the recommendation list changes substantially after every new interaction.

Further improvements in accuracy can be obtained by modeling user interest based on a few recent interactions, as shown by the results obtained by algorithm $\text{IGSI}^r_{\hat{\pi}^3}$, which is the best performing method overall for metrics HitRate@20 and DCG@20, except for datasets

Table 6.9: Results for all algorithms for datasets ML-1M and ML-10M. Accuracy is measure by HR@20 and DCG@20. Diversity is measure by ILD and time is reported by average update and recommendation times in milliseconds, with the best results highlighted in bold.

Dataset	Algorithm	HR@20	DCG@20	ILD@20	Time (ms)	
					Update	Rec.
ML-1M	TOP-POP	0.133	0.053	0.682	0.001	0.2
	RE-POP	0.132	0.053	0.693	0.001	0.2
	ISGD	0.055	0.022	**0.912**	0.2	**5.0**
	RAISGD	0.054	0.022	0.897	0.2	**5.0**
	BPR-MF	0.183	0.072	0.820	0.6	**4.1**
	ItemKNN	0.171	0.066	0.838	3.73	13.36
	$\hat{P}^3_\alpha$	0.162	0.066	0.713	**0.002**	10.06
	$\hat{RP}^3_\beta$	0.165	0.067	0.736	**0.002**	10.06
	$IGSI^r_{P^t}$	0.228	0.101	0.843	0.13	9.82
	$IGSI^{li_u}_{\hat{\pi}}$	0.222	0.101	0.814	0.13	24.70
	$IGSI^r_{\hat{\pi}}$	**0.240**	**0.106**	0.791	0.13	24.74
	$IGSI^{Su}_{\hat{\pi}}$	0.180	0.074	0.727	0.13	25.91
ML-10M	TOP-POP	0.113	0.046	0.775	0.002	0.25
	RE-POP	0.150	0.062	0.802	0.002	0.16
	ISGD	0.061	0.026	**0.949**	0.3	**7.45**
	RAISGD	0.066	0.028	0.938	0.3	**7.31**
	BPR-MF	0.160	0.063	0.817	0.66	**6.52**
	ItemKNN	0.159	0.061	0.894	7.3	51.75
	$\hat{P}^3_\alpha$	0.133	0.053	0.879	**0.002**	10.61
	$\hat{RP}^3_\beta$	0.084	0.032	**0.950**	**0.002**	10.61
	$IGSI^r_{P^t}$	0.219	0.099	0.872	0.4	10.10
	$IGSI^{li_u}_{\hat{\pi}}$	0.211	0.097	0.841	0.4	28.32
	$IGSI^r_{\hat{\pi}}$	**0.222**	**0.100**	0.833	0.4	28.81
	$IGSI^{Su}_{\hat{\pi}}$	0.180	0.074	0.794	0.4	29.09

PLC-STR and GLOBO, for which the non-personalized $IGSI^{li_u}_{\hat{\pi}^3}$ obtained the best results. These results highlight the importance of modeling short-term user interests, as also suggested by the improvements of algorithm RAISGD in comparison to ISGD, which penalizes infrequent items in the data stream. This is further evident in dataset GLOBO, which is characterized by a dynamic set of items. Introducing negative feedback in the form of infrequent items significantly increased accuracy when compared to ISGD.

As expected, the incremental algorithms generally outperforms non-personalized popularity-based algorithms. We note, however, that both TOP-POP and RE-POP outperformed ISGD and RAISGD on movie-domain datasets. Also, RE-POP obtained very competitive results

Table 6.10: Results for all algorithms for datasets PLC-PL and PLC-STR. Accuracy is measure by HR@20 and DCG@20. Diversity is measure by ILD and time is reported by average update and recommendation times in milliseconds, with the best results highlighted in bold.

Dataset	Algorithm	HR@20	DCG@20	ILD@20	Time (ms)	
					Update	Rec.
PLC-PL	TOP-POP	0.020	0.007	0.870	0.005	0.5
	RE-POP	0.041	0.018	0.872	0.005	0.5
	ISGD	0.282	0.182	0.863	0.9	29.58
	RAISGD	0.289	0.187	0.863	0.9	26.82
	BPR-MF	0.082	0.039	0.841	1.3	23.53
	ItemKNN	0.298	0.145	0.811	2.49	76.91
	$\hat{P}^3_\alpha$	0.303	0.153	0.818	**0.002**	**9.79**
	$\hat{RP}^3_\beta$	0.310	0.157	0.821	**0.002**	**9.89**
	$IGSI^r_{P^t}$	0.342	0.239	**0.869**	0.02	**9.5**
	$IGSI^{li_u}_{\hat{\pi}}$	0.358	0.248	0.829	0.02	20.05
	$IGSI^r_{\hat{\pi}}$	**0.404**	**0.261**	0.836	0.02	21.55
	$IGSI^{Su}_{\hat{\pi}}$	0.355	0.209	0.851	0.02	21.99
PLC-STR	TOP-POP	0.011	0.003	0.822	0.003	5.7
	RE-POP	0.031	0.011	0.814	0.003	5.6
	ISGD	0.331	0.228	0.842	0.18	44.28
	RAISGD	0.340	0.229	0.844	0.18	43.20
	BPR-MF	0.002	0.001	**0.936**	0.3	35.42
	ItemKNN	0.183	0.079	0.779	18.17	150.77
	$\hat{P}^3_\alpha$	0.159	0.075	0.779	**0.004**	18.45
	$\hat{RP}^3_\beta$	0.168	0.079	0.834	**0.004**	18.49
	$IGSI^r_{P^t}$	0.565	0.425	0.829	0.06	**9.6**
	$IGSI^{li_u}_{\hat{\pi}}$	0.596	**0.456**	0.791	0.06	23.75
	$IGSI^r_{\hat{\pi}}$	**0.598**	0.433	0.795	0.06	23.88
	$IGSI^{Su}_{\hat{\pi}}$	0.312	0.161	0.822	0.06	25.09

on datasets ML-10 and GLOBO, only being outperformed by $IGSI^r_{\hat{\pi}^3}$, and by BPR-MF on ML-10M. Again, this relates to the dynamic set of items and popularity factors of a news-domain dataset.

Considering the related incremental algorithms, ISGD and RAISGD present fast recommendation times with competitive results for datasets from the music domain, with RAISGD generally providing increases in accuracy compared to ISGD. The exceptions for these increases are ML-1M and ELEC. BPR-MF on the other hand performed well for movie-domain datasets and ELEC, and performed poorly on music-domain datasets and on GLOBO. We believe that this pertains to the sampling of negative feedback. On ML-1M and ML-10M, we have

Table 6.11: Results for all algorithms for dataset LFM. Accuracy is measure by HR@20 and DCG@20. Diversity is measure by ILD and time is reported by average update and recommendation times in milliseconds, with the best results highlighted in bold.

Dataset	Algorithm	HR@20	DCG@20	ILD@20	Time (ms)	
					Update	Rec.
LFM	TOP-POP	0.001	0.0004	0.744	0.006	13.92
	RE-POP	0.001	0.0003	0.755	0.006	12.58
	ISGD	0.067	0.046	0.820	2.6	81.61
	RAISGD	0.070	0.048	0.838	2.6	82.40
	BPR-MF	0.042	0.023	0.847	5.80	69.99
	ItemKNN	0.020	0.009	0.680	14.54	165.18
	$\hat{\mathrm{P}}^3_\alpha$	0.003	0.001	0.846	**0.001**	49.42
	$\hat{\mathrm{RP}}^3_\beta$	0.003	0.001	0.852	**0.001**	49.12
	$\mathrm{IGSI}^r_{P^t}$	0.087	0.069	0.795	0.02	**9.98**
	$\mathrm{IGSI}^{li_u}_{\hat{\pi}}$	0.090	0.070	0.705	0.02	21.95
	$\mathrm{IGSI}^r_{\hat{\pi}}$	**0.104**	**0.075**	0.744	0.02	22.73
	$\mathrm{IGSI}^{Su}_{\hat{\pi}}$	0.045	0.022	**0.871**	0.02	23.86

smaller sets of items, and users interact with more items. On the remaining datasets, we have larger sets of items, but users interact with a very small number of items available. Hence, the sampling procedure may be selecting irrelevant items as negative feedback, that provide little information to the preferences of the active user.

ItemKNN, $\hat{\mathrm{P}}^3_\alpha$ and $\hat{\mathrm{RP}}^3_\beta$ presented competitive results for datasets ML-1M, ML-10M and ELEC. This is related to the characteristics of the datasets and the recommendations provided by each algorithm. Since ISGD and RAISGD adjust the latent factors to give greater importance to more recent observations, they are better suited for recommendation in domains such as music that display short-term temporal dynamics, while ItemKNN, $\hat{\mathrm{P}}^3_\alpha$ and $\hat{\mathrm{RP}}^3_\beta$ provide recommendations based on long-term interests.

Comparing algorithms $\hat{\mathrm{P}}^3_\alpha$ and $\hat{\mathrm{RP}}^3_\beta$, we can see that with the exception of datasets ML-10M and ELEC, $\hat{\mathrm{RP}}^3_\beta$ outperforms $\hat{\mathrm{P}}^3_\alpha$ in accuracy and diversity. Further improvements in diversity can be obtained for algorithm $\hat{\mathrm{RP}}^3_\beta$ by increasing parameter β. However, as demonstrated by the results obtained by $\hat{\mathrm{RP}}^3_\beta$ for the ML-10M, the number of samples required for the algorithm to converge is highly dependent on the choice of parameter β, such that higher values for β requires more samples to converge (Paudel et al., 2017).

Another observation from the results is that there is no algorithm that presents the best results for both accuracy and diversity. In fact, for all datasets the worst performing algorithm in terms of accuracy presents the best results in terms of diversity. This is justified by the evaluated metric, ILD, which measures the average distance between items in the recommendation list, by cosine distance. Ideally, a model should increase diversity while at least maintaining its original

Table 6.12: Results for all algorithms for datasets ELEC and GLOBO. Accuracy is measure by HR@20 and DCG@20. Diversity is measure by ILD and time is reported by average update and recommendation times in milliseconds, with the best results highlighted in bold.

Dataset	Algorithm	HR@20	DCG@20	ILD@20	Time (ms)	
					Update	Rec.
ELEC	TOP-POP	0.043	0.020	0.986	0.004	1.00
	RE-POP	0.073	0.032	0.987	0.005	1.00
	ISGD	0.184	0.160	**0.993**	0.3	70.63
	RAISGD	0.183	0.161	**0.993**	0.3	69.71
	BPR-MF	0.194	0.160	**0.993**	5.04	62.11
	ItemKNN	0.193	0.103	0.975	0.3	140.48
	$\hat{P}^3_\alpha$	**0.228**	0.185	0.981	**0.002**	**7.24**
	$\hat{RP}^3_\beta$	0.226	0.184	0.979	**0.002**	**7.35**
	$IGSI^r_{P^t}$	0.211	0.181	**0.994**	0.01	8.57
	$IGSI^{li_u}_{\hat{\pi}}$	0.213	0.179	0.983	0.01	18.9
	$IGSI^r_{\hat{\pi}}$	0.225	0.187	0.984	0.01	19.70
	$IGSI^{Su}_{\hat{\pi}}$	0.226	**0.188**	0.984	0.01	19.59
GLOBO	TOP-POP	0.089	0.031	0.912	0.002	0.6
	RE-POP	0.377	0.163	0.924	0.002	0.6
	ISGD	0.007	0.002	0.996	2.1	34.71
	RAISGD	0.290	0.146	0.954	2.3	38.94
	BPR-MF	0.105	0.036	0.980	3.8	29.68
	ItemKNN	0.041	0.013	**0.999**	4.86	58.94
	$\hat{P}^3_\alpha$	0.279	0.130	0.932	**0.002**	**9.60**
	$\hat{RP}^3_\beta$	0.293	0.137	0.937	**0.002**	**9.93**
	$IGSI^r_{P^t}$	0.466	0.242	0.950	0.02	10.23
	$IGSI^{li_u}_{\hat{\pi}}$	**0.482**	**0.250**	0.937	0.02	26.2
	$IGSI^r_{\hat{\pi}}$	0.470	0.240	0.936	0.02	26.1
	$IGSI^{Su}_{\hat{\pi}}$	0.347	0.166	0.930	0.02	27.2

accuracy, as is generally the case for algorithms $\hat{P}^3_\alpha$ and $\hat{RP}^3_\beta$. For all datasets, our proposal $IGSI_{\hat{\pi}^3}$-based algorithms obtained better accuracy with competitive diversity.

In terms of update time, $IGSI_{\hat{\pi}^3}$-based algorithms, $\hat{P}^3_\alpha$, $\hat{RP}^3_\beta$, ISGD, BPR-MF and RAISGD achieved competitive results, and are all suitable for data stream environments. Considering recommendation time, with the exception of ItemKNN, which is the most computationally demanding algorithm, all algorithms present acceptable time, with ISGD, RAISGD and BPR-MF outperforming all methods for both ML-1M and ML-10M datasets, $\hat{P}^3_\alpha$ and $\hat{RP}^3_\beta$ outperforming other methods for datasets PLC-PL, PLC-STR, ELEC and GLOBO, respectively, and $IGSI^{li_u}_{\hat{\pi}^3}$

outperforming other methods for dataset LFM. We note that the recommendation procedure of ISGD, RAISGD, BPR-MF and ItemKNN are linear to the number of items in the dataset, while $\hat{\mathrm{P}}^3_\alpha$, $\hat{\mathrm{RP}}^3_\beta$ and $\mathrm{IGSI}_{\hat{\pi}^3}$-based algorithms depend on the number of predefined samples.

In order to analyze the learning behavior of each algorithm over time, in Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6 and 6.7, we present the accuracy of all algorithms over time with a moving average of the HitRate@20 metric for all datasets with a window of size $n = 5000$.

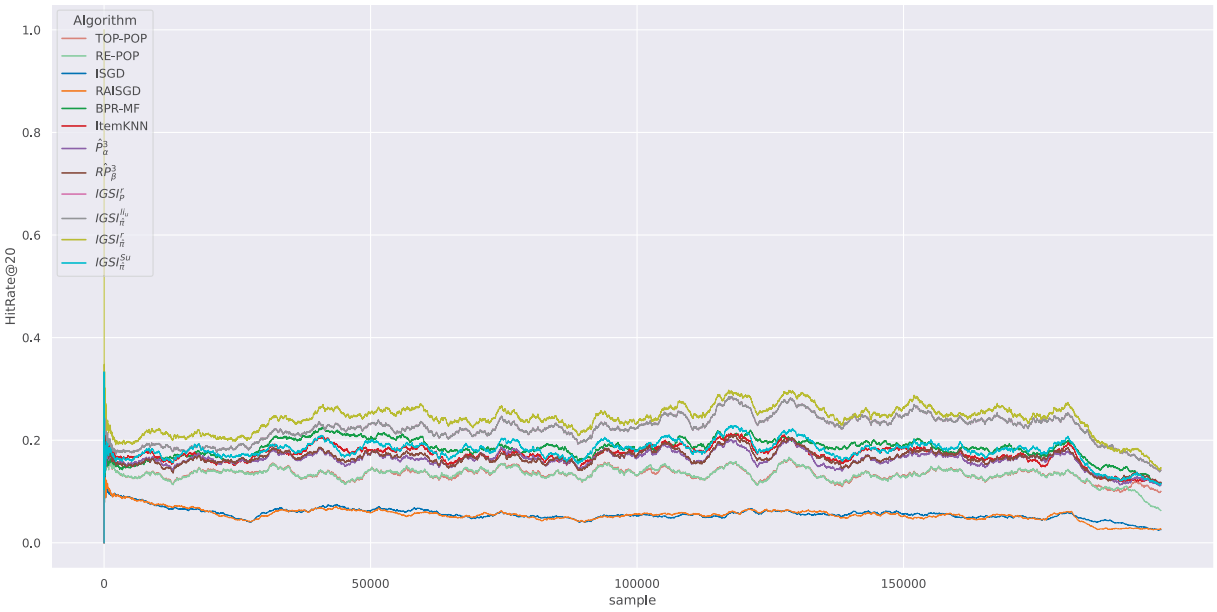


Figure 6.1: Evolution of HitRate@20 as events arrive for dataset ML-1M with window size $n = 5000$.

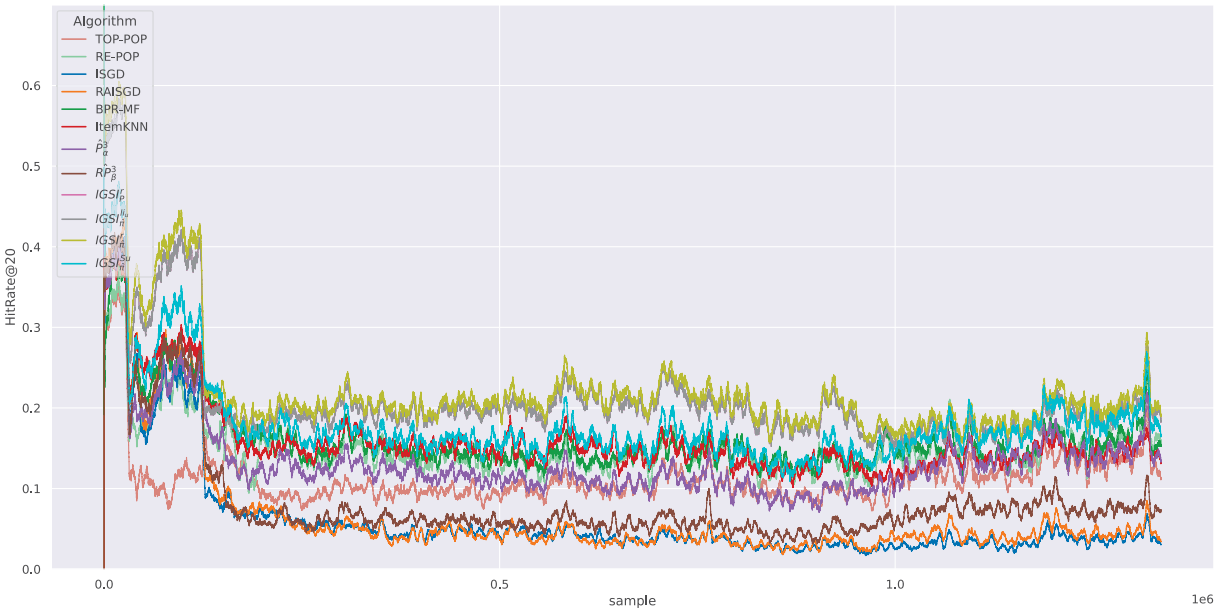


Figure 6.2: Evolution of HitRate@20 as events arrive for dataset ML-10M with window size $n = 5000$.

From Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6 and 6.7 we can confirm that the proposed method $\mathrm{IGSI}^r_{\hat{\pi}^3}$ consistently outperforms the related algorithms. Another interesting observation is the similarity between algorithms $\mathrm{IGSI}^{S_u}_{\hat{\pi}^3}$, $\hat{\mathrm{P}}^3_\alpha$ and $\hat{\mathrm{RP}}^3_\beta$ for the ML-1M and ML-10M (Figures 6.1 and 6.2). We can see improvement from $\mathrm{IGSI}^{S_u}_{\hat{\pi}^3}$ in accuracy in comparison to $\hat{\mathrm{P}}^3_\alpha$ and $\hat{\mathrm{RP}}^3_\beta$, which is

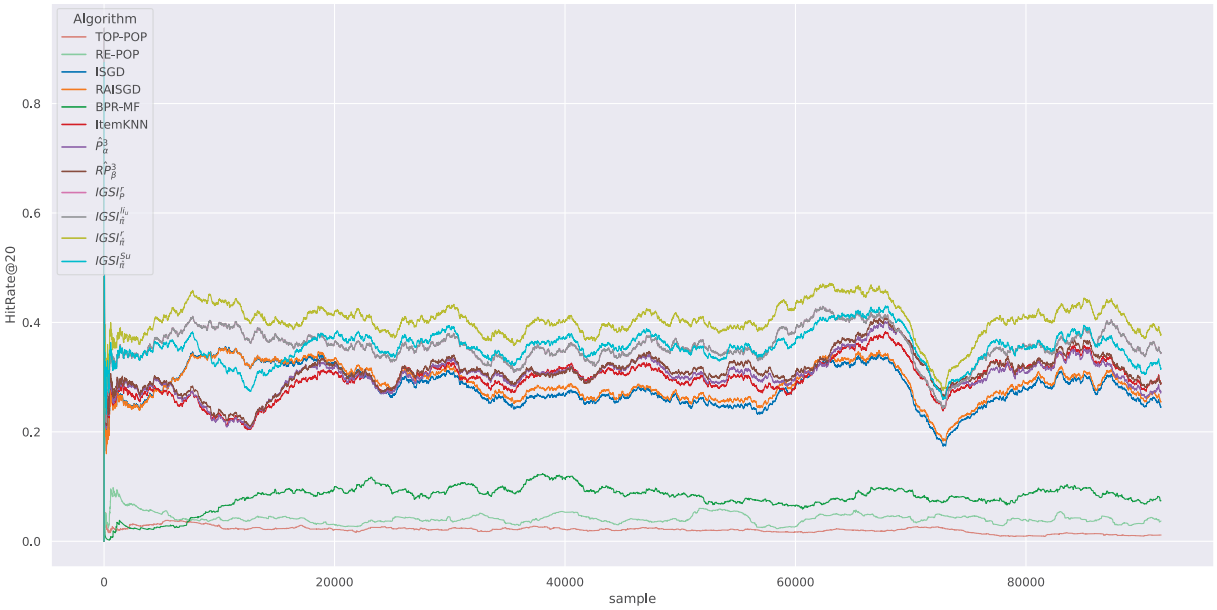


Figure 6.3: Evolution of HitRate@20 as events arrive for dataset PLC-PL with window size $n = 5000$.

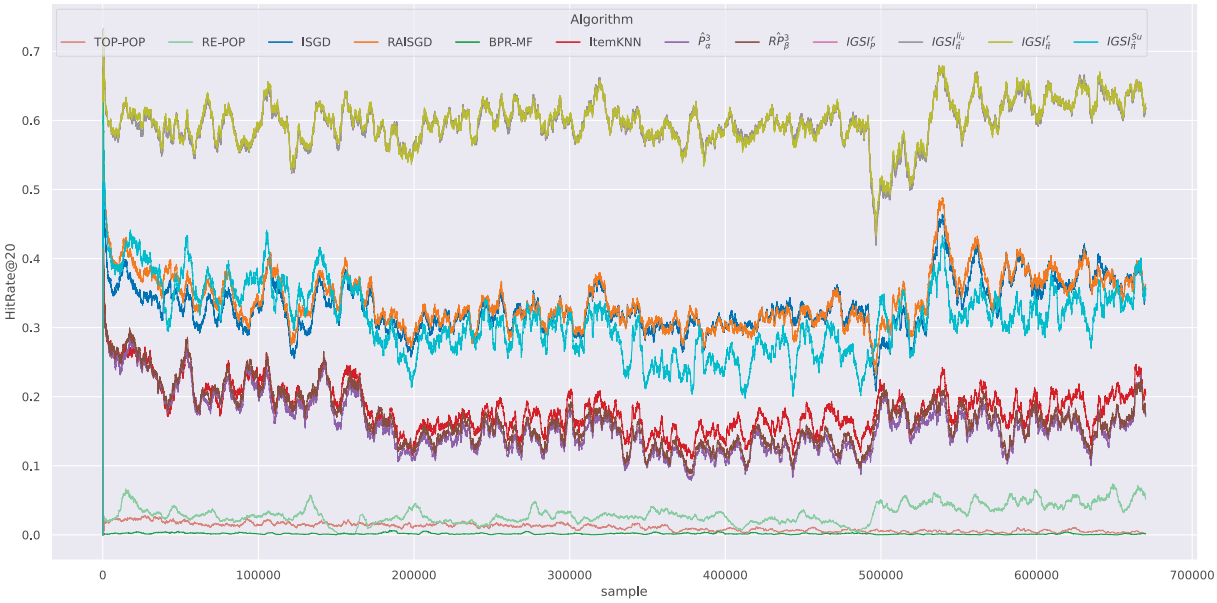


Figure 6.4: Evolution of HitRate@20 as events arrive for dataset PLC-STR with window size $n = 5000$.

related to the sequential characteristics of $\text{IGSI}^{S_u}_{\hat{\pi}^3}$, whereas in $\hat{\text{P}}^3_\alpha$ and $\hat{\text{RP}}^3_\beta$ there is no distinction between the sequence of interactions in the graph. Another consideration is the popularity effect present in dataset GLOBO, as seen in the spikes in accuracy obtained by the RE-POP algorithm, which evidences the interests of users in fresh articles that quickly becomes irrelevant.

The aforementioned plots also allows us to evaluate the statistical significance of the results. To assess it, we conducted McNemar tests (Section 6.1.2) between algorithm $\text{IGSI}^r_{\hat{\pi}^3}$ and the other related algorithms ISGD, RAISGD, ItemKNN, BPR-MF, $\hat{\text{P}}^3_\alpha$ and $\hat{\text{RP}}^3_\beta$ for every dataset using sliding windows with size $n = 5000$, the same size used for the moving averages in Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6 and 6.7.

We verified that $\text{IGSI}^r_{\hat{\pi}^3}$ outperforms all competing approaches throughout 100% of the prequential process on datasets ML-1M, PLC-PL, PLC-STR and LFM, as visible by the

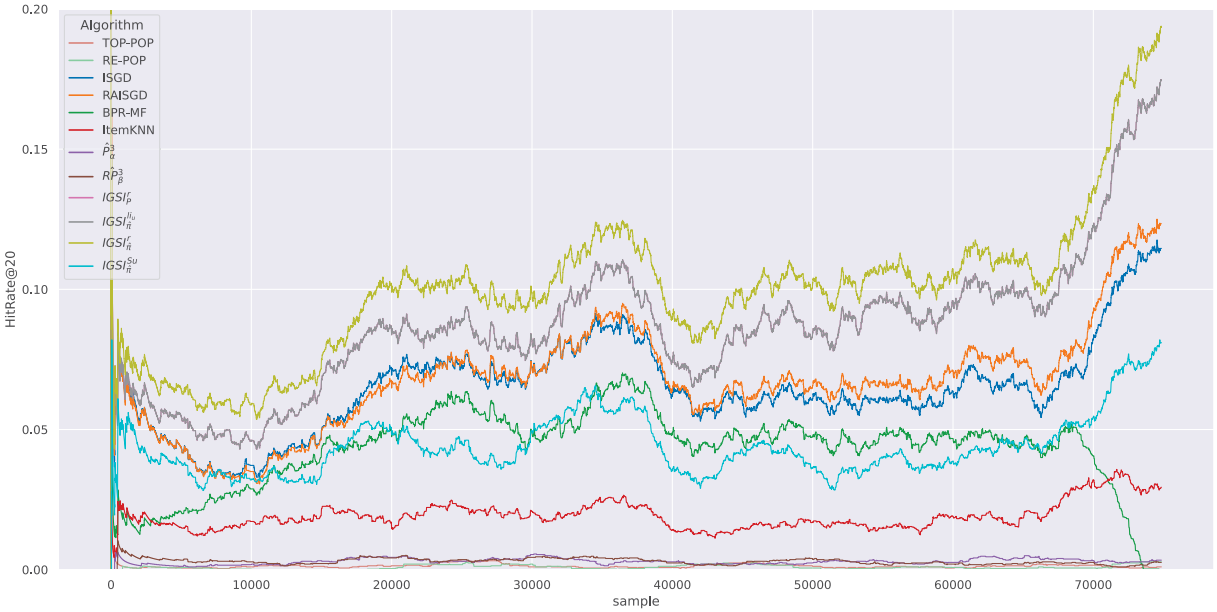


Figure 6.5: Evolution of HitRate@20 as events arrive for LFM dataset with window size $n = 5000$.

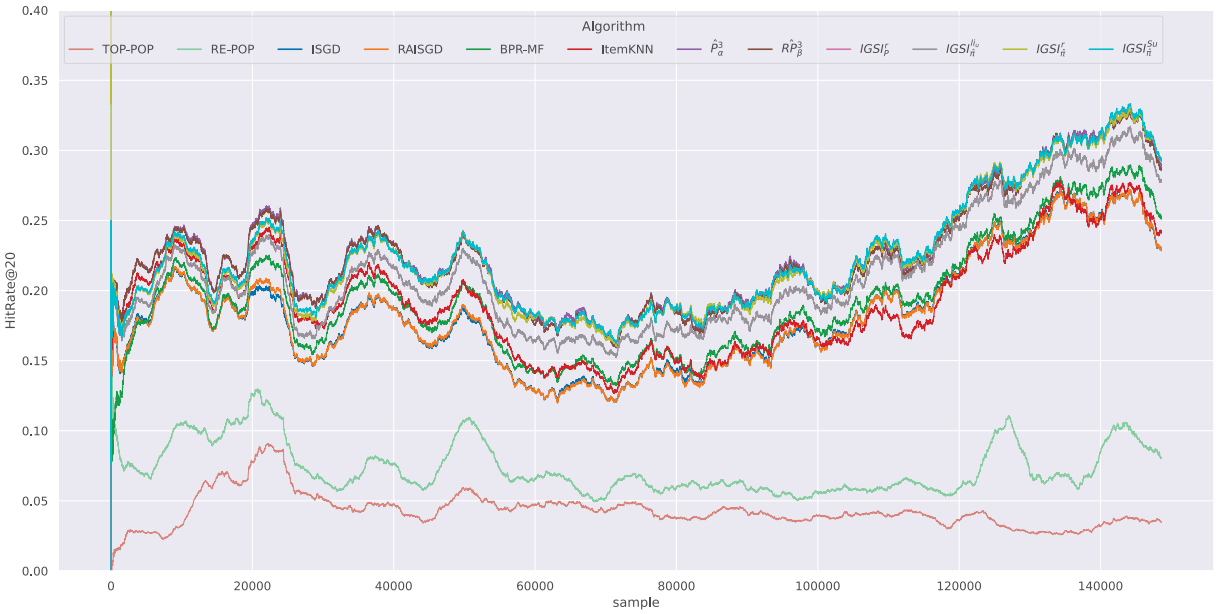


Figure 6.6: Evolution of HitRate@20 as events arrive for ELEC dataset with window size $n = 5000$.

differences in accuracy presented in Figures 6.1, 6.3, 6.4 and 6.5. For datasets ML-10M, ELEC and GLOBO, $\text{IGSI}^r_{\hat{\pi}^3}$ generally outperforms all competing approaches throughout 100% of the prequential process, as visible in Figures 6.2, 6.6 and 6.7 with a few exceptions.

First, for dataset ML-10M, $\text{IGSI}^r_{\hat{\pi}^3}$ outperforms RE-POP, BPR-MF and ItemKNN throughout 98.04%, 95.31% and 98.13% of the prequential process, respectively. On the remaining parts, there are no significant difference between the compared algorithms.

Second, for dataset ELEC, $\text{IGSI}^r_{\hat{\pi}^3}$ outperforms BPR-MF and ItemKNN throughout 89.20% and 77.86% of the prequential process, respectively. On the remaining parts, there are no significant difference between the compared algorithms. Also, there are no significant differences between $\text{IGSI}^r_{\hat{\pi}^3}$ and both $\hat{\text{P}}^3_\alpha$ and $\hat{\text{RP}}^3_\beta$ throughout 100% of the prequential process.

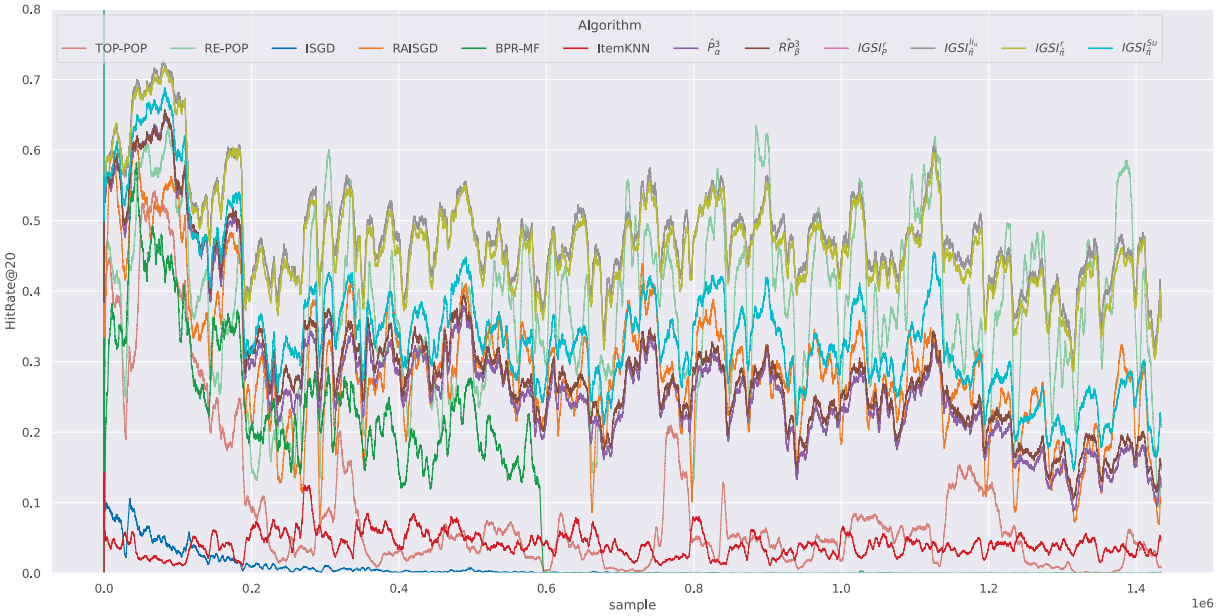


Figure 6.7: Evolution of HitRate@20 as events arrive for GLOBO dataset with window size $n = 5000$.

Finally, for dataset GLOBO, $\text{IGSI}^r_{\hat{\pi}^3}$ outperforms RE-POP throughout 76.48% of the prequential process, is outperformed on 10.79% and there is no significant difference on the remaining 12.72%. Parts of the prequential process where $\text{IGSI}^r_{\hat{\pi}^3}$ does not outperform RE-POP can be seen in Figure 6.7, as evident by the sudden spikes in accuracy obtained by RE-POP.

Overall, we conclude that our proposed SBRS, IGSI_{π^t}, is well suited for data stream scenarios, as it meets the prerequisites outlined in Chapter 3. As it learns incrementally, a single pass over the available data is required, which is done efficiently as suggested by the obtained update times. This, in turn, allows the model to keep up with the incoming data and always stay up-to-date. By staying up-to-date, an efficient procedure to generate recommendations is required. This is achieved with the random walk simulation procedure, which brings a balance between accuracy and scalability. As the datasets are very sparse, we found that such sampling procedure comes close to convergence with a reasonable number of samples, which results in very competitive recommendation times. Finally, our results suggest the effectiveness of our proposal, which generally outperformed the compared algorithms throughout the simulated streaming process in all datasets.

6.3 EVALUATION OF LND

In this section, we extend IGSI_{π^t} with our proposed forgetting technique, local neighborhood decay (LND), defined in Section 5.2. Such extension results in an *evolving* model that learns continuously and incrementally with newly generated data, and also removes information that is deemed as obsolete, which in turn enables our proposal to fully meet the last requirement posed in Chapter 3: *the model must adapt to changes and stay up-to-date, but also maintain information that is not outdated.*

The goal of this section of experiments is to assess the effectiveness of our proposed forgetting technique. We evaluate it under several metrics, comparing it with other forgetting techniques, and also analyze the influence of its hyperparameters.

6.3.1 Related algorithms

To assess the effectiveness of our proposed forgetting technique, LND, we use $\text{IGSI}_{\hat{\pi}^t}$ as a base model and extend it with LND, and also with the following ones to allow for comparisons:

1. *Sliding window.* A traditional abrupt forgetting technique (Nasraoui et al., 2007; Vinagre and Jorge, 2012; Siddiqui et al., 2014). We define sliding windows based on time intervals ω, e.g., 30 days, where only observations included in the windows are considered for the ranking of items, and observations outside of it are abruptly forgotten. $\text{IGSI}_{\hat{\pi}^t}$ with sliding window would consist in a graph composed only by interactions that occurred in the intervals defined by the window.

2. *Time-decay.* A gradual forgetting technique (Koychev, 2000; Tabassum et al., 2020) that reduces the relevance of edges based on time intervals. We define a fixed time window ω, e.g., 30 days, and each time the predetermined window period elapses, edges are gradually forgotten based on Eq.(5.3), and removed if necessary, i.e., $w(e) < 1.0$. With this technique, forgetting is performed gradually and only based on the predefined time window, as opposed to a sliding window that abruptly forgets information.

3. *Recency queue.* An adaptation of the algorithm proposed by Vinagre et al. (2015a) designed to select negative feedback from the stream based on the frequency of items. In this forgetting technique, a global FIFO queue is maintained, containing all items seen in the stream, where the tail contains the most recent interacted items, and the head contains the items that have not been interacted with for a while. For an incoming interaction $\langle u, i, t \rangle$, item i is moved to the tail of the queue, and the item at the head of the queue j is selected to lose relevance, by reducing the weight of the edges connecting its predecessors to j, as defined in Eq.(5.3). If necessary, edges are removed from the graph based on threshold ϕ. Finally, j is reinserted at the tail of the queue. In essence, this approach penalizes infrequent items along the stream and values recent information as these will not be forgotten.

6.3.2 IGSI Hyperparameters

The original $\text{IGSI}_{\hat{\pi}^t}$ has four hyperparameters: the number of steps in a random walk t, time window parameter ρ, the number of random walk samples M and the number of most recent user interactions to use as random walk source r. We set hyperparameters t, ρ and r with the optimal values reported in Sections 6.2.2, 6.2.3 and 6.2.4, respectively, and reduce the number of random walk samples to $M = 1000$ due to computational restrictions. Also, in this set of experiments, we introduce a new dataset, BOOK, from the e-commerce domain, and replace dataset LFM with LFM-1K, which is about ten times larger. Following the same methodology used in the previous section, we found the optimal values for r as $r = 5$ for BOOK and $r = 4$ for LFM-1K.

For parameters related to forgetting, we tested values for $\alpha \in [0.8, 0.99]$ in steps of 0.01. Parameter α is the fading factor that controls the rate of forgetting, where higher values for α results in slower forgetting. We observed that higher values for α presented the best results for all techniques. Hence, we set $\alpha = 0.99$ for all forgetting approaches. Hyperparameters for baselines were optimized based on grid search following the same methodology used to optimize the hyperparameters of our proposal, and are reported in the overall results. Besides α, our proposal has three other hyperparameters, and we discuss their impact next, before performing comparisons with baselines.

6.3.3 Impact of diversity hyperparameter

First we evaluate the impact of diversity parameter β, that acts as a convex combination of indegree and outdegree factors, $x_{s,p}$ and $y_{s,p}$, respectively. Nodes with low indegree are likely to be items from the long-tail, or new ones, while nodes with high outdegree increases the exploration of random walks. Thus, β introduces a balance between these two factors, and, based on Eq.(5.5), higher values of β should increase the diversity of recommendations, as we are including items from the long-tail in the recommendation lists. We tested values for $\beta \in [0, 1]$ in steps of 0.1. Results of these experiments, measured by HR@20, DCG@20 and ILD are presented in Tables 6.13 and 6.14.

Table 6.13: Impact of parameter β on $\text{IGSI}_{\hat{\pi}^t}$ with our proposed forgetting technique for datasets ML-1M, ML-10M, PLC-PL and PLC-STR. Column β refers to the tested values for hyperparameter β, while HR and ILD represents the HitRate@20 and intra-list diversity, respectively, grouped by dataset. The first row of the results refer to $\text{IGSI}_{\hat{\pi}^t}$ without forgetting. Values highlighted in bold indicate the highest value for β that is superior in accuracy with statistical significance in comparison to higher values.

	ML-1M		**ML-10M**		**PLC-PL**		**PLC-STR**	
β	**HR**	**ILD**	**HR**	**ILD**	**HR**	**ILD**	**HR**	**ILD**
-	0.234	0.813	0.198	0.829	0.396	0.826	0.590	0.817
1.0	0.232	0.839	0.199	0.874	**0.396**	**0.827**	0.591	0.819
0.9	0.235	0.833	0.212	0.863	0.396	0.827	0.591	0.819
0.8	0.235	0.826	0.220	0.854	0.396	0.827	0.591	0.818
0.7	**0.237**	**0.819**	0.224	0.844	0.396	0.826	0.591	0.818
0.6	0.236	0.812	**0.225**	**0.835**	0.397	0.826	**0.592**	**0.818**
0.5	0.236	0.805	0.222	0.826	0.397	0.826	0.592	0.817
0.4	0.233	0.798	0.216	0.821	0.397	0.826	0.592	0.817
0.3	0.232	0.792	0.209	0.819	0.397	0.826	0.592	0.817
0.2	0.226	0.786	0.205	0.818	0.397	0.826	0.592	0.817
0.1	0.226	0.782	0.198	0.816	0.397	0.826	0.592	0.816
0.0	0.221	0.778	0.191	0.815	0.397	0.826	0.592	0.816

As shown in Tables 6.13 and 6.14, for all datasets diversity increases with parameter β, as expected. The effect of β in accuracy depends on the dataset: for ML-1M, ML-10M and GLOBO, it tends to increase as β increases until reaching a certain threshold, after which it starts to decrease, but generally is still higher than $\text{IGSI}_{\hat{\pi}^t}$ without forgetting. For datasets PLC-PL and LFM-1K, β does not affect accuracy, while for PLC-STR, BOOK and ELEC it only slightly affects it.

Two interesting observations can be made from these results. First, that for all datasets there is at least one value for β that increases diversity while maintaining accuracy, or even increasing it. Second, the impact of β on accuracy is highly dependent on the dataset and directly related to the degree distribution on the graph. This effect can be seen when we compare results on ML-1M, ML-10M and GLOBO to the remaining datasets. There are greater changes in accuracy on these datasets as they have higher average neighborhood size. Thus, β can be adjusted according to the application.

For subsequent experiments, we select values for β that maximizes diversity without decreasing accuracy. To that end, we assess the statistical significance of the results. We select the maximum value for β whose difference to greater values of it results in statistically significant differences in accuracy. In other words, starting from $\beta = 1.0$, we decrease β until there are no

Table 6.14: Impact of parameter β on $\text{IGSI}_{\hat{\pi}^t}$ with our proposed forgetting technique for datasets LFM-1K, BOOK, ELEC and GLOBO. Column β refers to the tested values for hyperparameter β, while HR and ILD represents the HitRate@20 and intra-list diversity, respectively, grouped by dataset. The first row of the results refer to $\text{IGSI}_{\hat{\pi}^t}$ without forgetting. Values highlighted in bold indicate the highest value for β that is superior in accuracy with statistical significance in comparison to higher values.

	LFM-1K		**BOOK**		**ELEC**		**GLOBO**	
β	**HR**	**ILD**	**HR**	**ILD**	**HR**	**ILD**	**HR**	**ILD**
-	0.182	0.841	0.661	0.827	0.268	0.986	0.456	0.940
1.0	**0.182**	**0.842**	0.661	0.832	0.267	0.987	0.490	0.950
0.9	0.182	0.842	**0.662**	**0.831**	0.267	0.987	0.527	0.945
0.8	0.182	0.842	0.662	0.830	0.267	0.987	0.555	0.944
0.7	0.182	0.842	0.662	0.829	0.267	0.987	0.574	0.943
0.6	0.182	0.842	0.662	0.829	**0.268**	**0.987**	0.583	0.943
0.5	0.182	0.841	0.662	0.828	0.269	0.986	**0.594**	**0.942**
0.4	0.182	0.841	0.662	0.828	0.269	0.986	0.591	0.941
0.3	0.182	0.841	0.662	0.828	0.269	0.986	0.587	0.941
0.2	0.182	0.841	0.662	0.828	0.269	0.986	0.581	0.941
0.1	0.182	0.841	0.662	0.828	0.270	0.986	0.576	0.941
0.0	0.182	0.841	0.662	0.828	0.269	0.986	0.562	0.941

statistically significant differences in accuracy. Thus, we set $\beta = 0.7$ for ML-1M and ML-10M, $\beta = 1.0$ for LFM-1K and PLC-PL, $\beta = 0.6$ for PLC-STR, $\beta = 0.9$ for BOOK, $\beta = 0.1$ for ELEC and $\beta = 0.5$ for GLOBO.

6.3.4 Impact of acceptance factor hyperparameter

The idea of acceptance factor $z_{s,p}$ in Eq.(5.5) is to further decrease the score of popular items, on the assumption that popular items which are constantly ignored are likely to be irrelevant as opposed to unknown to users (He et al., 2016). As popular items are nodes in the graph with several connections, further penalizing them with $z_{s,p}$ should improve scalability when combined with the weight threshold ϕ, and should reduce the dominance of popular items in recommendation lists. We tested values for $\tau \in [0, 1]$ in steps of 0.1, after first tuning β. Note that $\tau = 0.0$ ignores $z_{s,p}$ and only score nodes based on $x_{s,p}$ and $y_{s,p}$. Results of these experiments in accuracy, measured by HR@20 and DCG@20 are presented in Tables 6.15 and 6.16.

From Tables 6.15 and 6.16, we see that for all datasets, apart from LFM-1K and BOOK, there is a value for $\tau > 0$ that increases accuracy with statistical significance, and similarly to β, the effect of τ depends on the dataset. For dataset BOOK there are minor increases in DCG that are not significant, while for LFM-1K there are no changes. Datasets ML-1M, ML-10M, PLC-STR and ELEC benefit from small values of τ, while GLOBO is highly affected by the hyperparameter and benefits from higher values of it. This results are related to the characteristics of these datasets. GLOBO is a news domain dataset, with a dynamic set of items. New items are constantly added to the set, and these items have a short lifespan. Thus, they benefit from faster forgetting. On the other hand, datasets from domains characterized by stable relations, e.g., movies, benefit from slower forgetting, as these relations remain relevant even if not reinforced by new data.

Table 6.15: Impact of parameter τ on $\text{IGSI}_{\hat{\pi}^t}$ with our proposed forgetting technique for datasets ML-1M, ML-10M, PLC-PL and PLC-STR. Column τ refers to the tested values for hyperparameter τ, while HR and DCG represents the HitRate@20 and DCG@20, respectively, grouped by dataset. Results for $\tau = 0$ refers to the best values for β obtained in Tables 6.13 and 6.14. Values highlighted in bold indicate the highest value for τ that is superior in accuracy with statistical significance in comparison to lower values.

	ML-1M		**ML-10M**		**PLC-PL**		**PLC-STR**	
τ	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**
0	0.237	0.104	0.225	0.100	0.396	0.260	0.592	0.432
0.1	**0.238**	**0.105**	0.226	0.101	**0.397**	**0.261**	0.592	0.432
0.2	0.236	0.104	**0.227**	**0.102**	0.397	0.260	0.592	0.432
0.3	0.236	0.104	0.226	0.101	0.397	0.260	**0.593**	**0.433**
0.4	0.236	0.104	0.226	0.101	0.397	0.260	0.593	0.433
0.5	0.236	0.104	0.226	0.101	0.397	0.260	0.593	0.433
0.6	0.235	0.104	0.226	0.101	0.397	0.260	0.593	0.433
0.7	0.235	0.104	0.226	0.101	0.397	0.260	0.593	0.433
0.8	0.235	0.104	0.225	0.101	0.397	0.260	0.593	0.433
0.9	0.235	0.104	0.225	0.101	0.396	0.260	0.593	0.433
1.0	0.235	0.103	0.225	0.101	0.396	0.260	0.593	0.433

Table 6.16: Impact of parameter τ on $\text{IGSI}_{\hat{\pi}^t}$ with our proposed forgetting technique for datasets LFM-1K, BOOK, ELEC and GLOBO. Column τ refers to the tested values for hyperparameter τ, while HR and DCG represents the HitRate@20 and DCG@20, respectively, grouped by dataset. Results for $\tau = 0$ refers to the best values for β obtained in Tables 6.13 and 6.14. Values highlighted in bold indicate the highest value for τ that is superior in accuracy with statistical significance in comparison to lower values.

	LFM-1K		**BOOK**		**ELEC**		**GLOBO**	
τ	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**	**HR**	**DCG**
0	**0.182**	**0.141**	**0.662**	**0.452**	0.269	0.215	0.591	0.294
0.1	0.182	0.141	0.662	0.455	0.269	0.215	0.610	0.307
0.2	0.182	0.141	0.662	0.455	**0.270**	**0.215**	0.615	0.312
0.3	0.182	0.141	0.662	0.455	0.269	0.215	0.619	0.314
0.4	0.182	0.141	0.662	0.455	0.269	0.215	0.621	0.314
0.5	0.182	0.141	0.662	0.455	0.268	0.214	0.624	0.318
0.6	0.182	0.141	0.662	0.456	0.268	0.214	0.625	0.318
0.7	0.182	0.141	0.662	0.456	0.268	0.214	0.626	0.318
0.8	0.182	0.141	0.662	0.456	0.268	0.214	**0.627**	**0.320**
0.9	0.182	0.141	0.662	0.456	0.268	0.214	0.627	0.320
1.0	0.182	0.141	0.662	0.456	0.268	0.214	0.628	0.320

Essentially, setting higher values for τ results in faster forgetting for items with low acceptance. In subsequent experiments, we set $\tau = 0.1$ for ML-1M and PLC-PL, $\tau = 0.2$ for ML-10M and ELEC, $\tau = 0.3$ for PLC-STR and $\tau = 0.8$ for GLOBO.

6.3.5 Impact of weight threshold hyperparameter

Next, based on the optimal values for β and τ, we evaluate the impact of weight threshold ϕ on the performance of $\text{IGSI}_{\hat{\pi}^t}$ with the proposed forgetting technique, where ϕ removes obsolete edges from the graph based on parameters α and x. Results are presented in Tables 6.17 and

6.18, with accuracy measured through HR@20. We also report the average number of edges in the graph.

Table 6.17: Impact of parameter x on IGSI$_{\hat{\pi}^t}$ with local neighborhood decay forgetting for datasets ML-1M, ML-10M, PLC-PL and PLC-STR. Column x refers to the tested values for hyperparameter x, while HR and $|\mathcal{E}|$ represents the HitRate@20 and the number of edges on the underlying graph, respectively, grouped by dataset. The first row of the results refer to IGSI$_{\hat{\pi}^t}$ without forgetting. Values highlighted in bold indicate the smallest value for x that is superior in accuracy with statistical significance in comparison to lower values.

	ML-1M		**ML-10M**		**PLC-PL**		**PLC-STR**	
x	**HR**	$\|\mathcal{E}\|$	**HR**	$\|\mathcal{E}\|$	**HR**	$\|\mathcal{E}\|$	**HR**	$\|\mathcal{E}\|$
-	0.234	61,939	0.198	224,781	0.396	28,644	0.590	185,060
5	0.238	21,288	0.221	53,231	0.395	26,338	0.591	121,592
10	0.242	27,832	0.226	66,854	**0.398**	**27,646**	0.592	140,580
15	**0.244**	**32,864**	0.230	77,676	0.397	28,196	**0.593**	**151,412**
20	0.244	36,793	0.232	86,481	0.397	28,430	0.593	158,420
25	0.243	39,434	**0.233**	**94,019**	0.397	28,539	0.593	163,266

Table 6.18: Impact of parameter x on IGSI$_{\hat{\pi}^t}$ with local neighborhood decay forgetting for datasets LFM-1K, BOOK, ELEC and GLOBO. Column x refers to the tested values for hyperparameter x, while HR and $|\mathcal{E}|$ represents the HitRate@20 and the number of edges on the underlying graph, respectively, grouped by dataset. The first row of the results refer to IGSI$_{\hat{\pi}^t}$ without forgetting. Values highlighted in bold indicate the smallest value for x that is superior in accuracy with statistical significance in comparison to lower values.

	LFM-1K		**BOOK**		**ELEC**		**GLOBO**	
x	**HR**	$\|\mathcal{E}\|$	**HR**	$\|\mathcal{E}\|$	**HR**	$\|\mathcal{E}\|$	**HR**	$\|\mathcal{E}\|$
-	0.182	1,019,439	0.661	784,214	0.268	36,401	0.456	143,262
5	**0.182**	**1,003,435**	0.660	670,291	0.268	29,715	0.614	63,163
10	0.182	1,015,856	**0.661**	**713,342**	**0.269**	**32,022**	0.622	73,394
15	0.182	1,018,308	0.661	732,023	0.269	33,315	0.625	79,956
20	0.182	1,019,023	0.661	743,387	0.269	34,037	**0.626**	**84,744**
25	0.182	1,019,262	0.662	750,876	0.269	34,539	0.626	88,451

From Tables 6.17 and 6.18, when comparing its first row, which is IGSI$_{\hat{\pi}^t}$ without forgetting, to increasing values of x, we can see that LND is able to considerably reduce the number of edges on the graph, thus reducing memory requirements, while also obtaining major improvements in accuracy for ML-1M, ML-10M and GLOBO, and slight improvements for PLC-STR. For LFM-1K, PLC-PL, BOOK and ELEC, LND reduces the size of the model without decreases in accuracy. We note two main observations from these results. First, it is beneficial to set a threshold to remove edges, as accuracy stagnates, or starts to decrease, after a certain value of ϕ, which suggests an accumulation of obsolete (or unnecessary) information on the graph. The removal of these connections impacts directly accuracy and scalability.

Second, besides parameter x, the rate of forgetting is affected by hyperparameter τ. Lower values for τ reduces the discrepancies between scores of items, i.e., it increases the values for acceptance factor $z_{s,p}$ in Eq.(5.5) and thus it reduces its effect, which is to penalize popular items. On the other hand, higher values for τ increases the effect of $z_{s,p}$, which results in faster

forgetting. Hence, we note here that if the objective would be to obtain the smallest possible model, this could be achieved by setting $\tau = 1.0$.

For subsequent experiments, we select values for x that results in the smallest possible models without losses in accuracy. To that end, we assess the statistical significance of the results. We increase x until there are no significant difference in comparison to smaller values. We set $x = 5$ for LFM, $x = 10$ for PLC-PL, BOOK and ELEC, $x = 15$ for ML-1M and PLC-STR, $x = 20$ for GLOBO and $x = 25$ for ML-10M.

6.3.6 Overall results

Tables 6.19, 6.20 and 6.21 presents overall results for $\text{IGSI}_{\hat{\pi}^t}$ without forgetting, $\text{IGSI}_{\hat{\pi}^t}$ with baseline forgetting techniques and $\text{IGSI}_{\hat{\pi}^t}$ with our proposed technique. Accuracy is measured through HR@20 and DCG@20, diversity through ILD, and scalability measured through average update and recommendation times and number of edges in the graph. The best results are highlighted in bold. Optimal hyperparameters used for each approach by dataset are reported in Table 6.22.

To complement the experimental analysis we also present in Table 6.23 properties of underlying graphs per dataset. These describe the obtained graphs without the application of forgetting. We report general information, such as number of nodes, number of edges, density, average degree and minimum, maximum and average weighted degree. We also report the number of low degree nodes. We define a low degree node as one that is below the average degree.

The first main observation is that the impact on the results is highly related to the datasets. For the two movie-domain datasets, ML-1M and ML-10M, the proposed forgetting technique LND obtained the best results in accuracy. Considering dataset ML-1M, the application of the remaining ones generally decreased accuracy, with the exception of the time-decay approach. We note, however, that all forgetting approaches significantly decrease the average number of edges in the graph, which consequently impacts the recommendation time. For ML-10M, the application of forgetting overall improved both accuracy and scalability. With the exception of recency queue approach, the remaining ones improved accuracy with smaller models.

For the three music-domain datasets, LFM-1K, PLC-PL, PLC-STR, the impact of forgetting varies with the approach. Considering LFM-1K, we see that forgetting reduces the number of edges, at the expense of accuracy, specially for both time related approaches. The exception is LND, which is capable of reducing the number of edges, while maintaining its original accuracy.

For PLC-PL, the application of forgetting does not yield relevant results. We argue that this relates to the characteristics of the dataset. It is composed of about 100,000 interactions spread across close to four years of activity. Thus, the rate of data is very low, affecting the behavior of all forgetting techniques. For the sliding window, it is difficult to set an ideal size, as a smaller one does not include sufficient information, while a higher one does not actually perform any forgetting. The time-decay approach suffers from the same issue. For the recency queue and LND, there there are not sufficient interactions with the same items several times in order for these approaches to detect what has become irrelevant. This is seen when we compare accuracy and number of edges: these approaches are unable to perceive obsolete edges, hence not providing any change in accuracy when compared to no forgetting.

PLC-STR, on the other hand, is composed of about 1,400,000 interactions also spread across close to four years of activity. While the rate of data is not high, it has more information and we can better observe the impact of forgetting. LND manages to reduce the number of edges in the model without losses in accuracy. The same goes for the time-decay approach.

Table 6.19: Overall results for all techniques grouped by datasets ML-1M and ML-10M. Best results are highlighted in bold.

Technique	HR	DCG	ILD	$\|\mathcal{E}\|$	Time(ms)		
					Upd.	Rec.	Tot.
ML-1M							
IGSI$_{\hat{\pi}^t}$	0.233	0.103	0.788	109,901	0.2	5.3	5.5
LND	**0.248** (**+6.4%**)	**0.110** (**+6.8%**)	0.817 (+3.7%)	52,735 (-52%)	0.5	4.6	5.1 (-7.3%)
recency_queue	0.231 (-0.9%)	0.102 (-1.0%)	0.771 (-2.2%)	68,373 (-37.8%)	0.7	4.9	5.6 (+1.8%)
time_decay	0.238 (+2.1%)	0.104 (+1.0%)	0.799 (+1.4%)	52,040 (-52.6%)	0.2	4.6	4.8 (-12.7%)
sliding_window	0.204 (-12.4%)	0.090 (-12.6%)	**0.821** (**+4.2%**)	**46,087** (**-58.1%**)	0.2	4.4	**4.6** (**-16.4%**)
ML-10M							
IGSI$_{\hat{\pi}^t}$	0.192	0.087	0.837	476,448	0.5	5.9	6.4
LND	**0.222** (**+15.6%**)	**0.104** (**+19.5%**)	0.860 (+2.7%)	169,457 (-64.6%)	0.5	5.0	**5.5** (**-14.1%**)
recency_queue	0.194 (+1.0%)	0.087 (0.0%)	0.820 (-2.0%)	217,519 (-54.3%)	2.8	5.8	8.6 (+34.4%)
time_decay	0.203 (+5.7%)	0.093 (+6.9%)	0.840 (+0.4%)	**118,758** (**-75.1%**)	0.6	5.1	5.7 (-10.9%)
sliding_window	0.212 (+10.4%)	0.098 (+12.6%)	**0.862** (**+3.0%**)	127,683 (-73.2%)	0.3	5.3	5.8 (-9.4%)

Sliding window still suffer from the same issue present in PLC-PL, where it is difficult to set its appropriate size.

Considering the e-commerce domain, the impact of forgetting is very similar on both its datasets BOOK and ELEC. While recency queue fails to detect obsolete edges, LND removes it without losses in accuracy. We can also see some limitations of time-based approaches in some scenarios. In this case, long-term connections that are not co-occurring again in the stream, and infrequent items that are seldom interacted by any user are abruptly forgotten, even thought they are not obsolete. On the other hand, LND only removes connections from edges when newer ones are available, or are updated again.

Finally, for the news-domain dataset GLOBO, forgetting has an important role. A news-domain dataset is the exact opposite of what we have in the aforementioned ones. The set of items goes through constant churn, new items are continuously added to it and old ones are dropped. In other words, in this domain items have a very short shelf life, and become old very quickly (Das et al., 2007). Also, the user-item matrix is very sparse, and are prone to popularity factors, such as breaking news or popular topics (Lommatzsch and Albayrak, 2015). Hence, news-domain datasets seems to benefit from forgetting, and we confirm that on the obtained results. All techniques considerably reduce number of edges from its graphs, without compromising accuracy in case of recency queue and time-decay. LND and sliding window, on the other hand, significantly increased accuracy, with LND obtaining the best results while also reducing the number of edges by about half of the original. An important consideration is the

Table 6.20: Overall results for all techniques grouped by datasets PLC-PL, PLC-STR and LFM-1K. Best results are highlighted in bold.

Technique	HR	DCG	ILD	$\|\mathcal{E}\|$	Time(ms)		
					Upd.	Rec.	Tot.
PLC-PL							
IGSI$_{\hat{\pi}^t}$	0.402	0.249	0.838	56,922	0.02	4.2	4.2
LND	**0.403** (**+0.2%**)	**0.250** (**+0.4%**)	0.840 (+0.2%)	53,749 (-5.6%)	0.06	4.1	4.2 (0.0%)
recency_queue	0.402 (0.0%)	0.249 (0.0%)	0.838 (0.0%)	56,922 (0.0%)	0.3	4.3	4.6 (+9.5%)
time_decay	0.384 (-4.5%)	0.240 (-3.6%)	0.835 (-0.4%)	32,841 (-42.3%)	0.07	4.1	4.2 (0.0%)
sliding_window	0.355 (-11.7%)	0.221 (-11.2%)	**0.853** (**+1.8%**)	**25,365** (**-55.4%**)	0.05	4.0	**4.1** (**-2.4%**)
PLC-STR							
IGSI$_{\hat{\pi}^t}$	0.597	0.430	0.784	355,926	0.1	4.7	4.8
LND	**0.601** (**+0.7%**)	**0.432** (**+0.5%**)	0.788 (+0.4%)	276,382 (-22.3%)	0.1	4.6	4.7 (-2.1%)
recency_queue	0.595 (-0.3%)	0.429 (-0.2%)	0.786 (+0.3%)	322,699 (-9.3%)	1.0	4.9	5.9 (+22.9%)
time_decay	0.595 (-0.3%)	0.427 (-0.7%)	0.784 (0.0%)	207,944 (-41.6%)	0.1	4.6	4.7 (-2.1%)
sliding_window	0.570 (-4.5%)	0.416 (-3.3%)	**0.800** (**+2.0%**)	**118,604** (**-66.7%**)	0.1	4.5	**4.6** (**-4.2%**)
LFM-1K							
IGSI$_{\hat{\pi}^t}$	0.230	0.179	0.836	1,990,079	0.1	5.8	5.9
LND	**0.232** (**+0.9%**)	**0.180** (**+0.6%**)	0.836 (0.0%)	1,699,049 (-14.6%)	0.1	5.5	**5.6** (**-5.1%**)
recency_queue	0.220 (-4.3%)	0.172 (-3.9%)	0.834 (-0.2%)	1,581,937 (-20.5%)	15.6	5.6	21.2 (+259.3%)
time_decay	0.221 (-3.9%)	0.174 (-2.8%)	0.830 (-64.1%)	**1,068,103** (**-46.3%**)	0.1	5.6	5.7 (-3.4%)
sliding_window	0.202 (-12.2%)	0.156 (-12.8%)	**0.846** (**+1.2%**)	1,162,618 (-41.6%)	0.1	5.5	**5.6** (**-5.1%**)

effectiveness of sliding window, as its advantages align with the properties of the news-domain, and thus improved accuracy, with the smallest model by a wide margin.

We now present more general observations based on all obtained results, considering the evaluated metrics and the competing forgetting techniques. In terms of recommendation quality, measured by HR@20 and DCG, the proposed forgetting technique, local neighborhood decay, obtained the best results. When comparing LND to IGSI$_{\hat{\pi}^t}$ without forgetting, we note that LND always reduces the number of edges from the models without compromising accuracy. In fact, accuracy always increased, even if the improvements were minor. The effect of forgetting largely depends on the properties of the underlying graph, which can be seen in Table 6.23.

Table 6.21: Overall results for all techniques grouped by datasets BOOK, ELEC and GLOBO. Best results are highlighted in bold.

Technique	HR	DCG	ILD	$\|\mathcal{E}\|$	Time(ms)		
					Upd.	Rec.	Tot.
BOOK							
IGSI$_{\hat{\pi}^t}$	0.647	0.411	0.736	1,790,105	0.05	4.7	4.7
LND	**0.652** (**+0.8%**)	**0.417** (**+1.5%**)	**0.745** (**+1.2%**)	1,554,975 (-13.1%)	0.1	4.5	4.6 (-2.1%)
recency_queue	0.647 (0.0%)	0.411 (0.0%)	0.736 (0.0%)	1,783,292 (-0.4%)	30.1	4.7	34.8 (+640%)
time_decay	0.621 (-4.0%)	0.394 (-4.1%)	0.686 (-6.8%)	587,480 (-67.2%)	0.1	2.6	**2.7** (**-42.6%**)
sliding_window	0.559 (-7.4%)	0.346 (-15.8%)	0.680 (-7.6%)	**259,018** (**-85.5%**)	0.1	3.0	3.1 (-34.0%)
ELEC							
IGSI$_{\hat{\pi}^t}$	0.237	0.187	0.980	102,098	0.01	3.9	3.9
LND	**0.238** (**+0.4%**)	**0.188** (**+0.5%**)	0.980 (0.0%)	85,720 (-16.0%)	0.01	3.8	3.8 (-2.6%)
recency_queue	0.236 (-0.4%)	0.187 (0.0%)	0.980 (0.0%)	101,789 (-0.3%)	1.0	4.1	5.1 (+30.8%)
time_decay	0.230 (-3.0%)	0.183 (-2.1%)	0.981 (+0.1%)	35,569 (-65.2%)	0.01	3.2	3.2 (-17.9%)
sliding_window	0.209 (-11.8%)	0.166 (-11.2%)	**0.984** (**+0.4%**)	**17,143** (**-83.2%**)	0.01	2.9	**2.9** (**-25.6%**)
GLOBO							
IGSI$_{\hat{\pi}^t}$	0.443	0.228	0.938	340,327	0.2	5.0	5.2
LND	**0.628** (**+41.8%**)	**0.327** (**+43.4%**)	**0.941** (**+0.3%**)	195,755 (-42.5%)	0.2	4.6	4.8 (-7.7%)
recency_queue	0.442 (-0.2%)	0.228 (0.0%)	0.938 (0.0%)	211,555 (-37.8%)	0.2	5.0	5.2 (0.0%)
time_decay	0.443 (0.0%)	0.227 (-0.4%)	0.938 (0.0%)	150,933 (-55.7%)	0.2	4.9	5.1 (-1.9%)
sliding_window	0.592 (+33.6%)	0.297 (+30.3%)	0.940 (+0.2%)	**20,189** (**-94.1%**)	0.1	4.4	**4.5** (**-13.5%**)

Datasets with higher density and average degrees, ML-1M, ML-10M, PLC-STR and GLOBO were better affected by forgetting. This highlights the difficulty of deploying forgetting and removing obsolete information from already very sparse graphs.

In terms of diversity, the best performing technique is always the worst performer in accuracy. This is justified by the evaluated metric, ILD, which measures the average distance between items in the recommendation list, by cosine distance. Thus, these are in a sense conflicting objectives, as IGSI$_{\hat{\pi}^t}$ is an item-based recommendation approach. Ideally, the desirable result would be to increase both accuracy and diversity, or at least one in a controlled manner. This is achieved with LND, where parameter β specifically allows for an increase in diversity. Also,

when comparing LND to no forgetting, we see that we never have the decrease of either accuracy or diversity, and that in some cases there was an increase in both. We note that while the remaining forgetting techniques did obtained greater diversity in some datasets, they only did so at the expense of accuracy.

Besides parameter β, the effect of LND on diversity is clearly correlated with the degree distribution of the graph. Since nodes are scored based on popularity, measured by both indegree and outdegree factors (Eqs. (5.6) and (5.7)), the increases in diversity are more substantial in graphs with higher density and higher average degree. This can be seen when comparing the differences in diversity for ML-1M and ML-10 with BOOK and ELEC. For BOOK and ELEC, the two datasets with smallest average degree, there are no differences in diversity. This is because a large portion of nodes have low indegree and outdegree, and thus the options for random walk exploration are very limited. Consequently, LND is unable to increase the diversity. For ML-1M and ML-10M, which have the two highest average degree of all datasets, on the other hand, LND improved diversity. As the average degree distribution is higher, resulting in a larger number of neighbors, LND increases the likelihood of exploring less popular nodes in the random walk simulation, which impacts both diversity and accuracy.

Considering update time, it changes based on technique at use, and generally it incurs only a minor addition compared to no forgetting. The exception would be the recency queue technique, which presents high update time for datasets with a large number of items. We note that recency queue and LND are applied for each incoming observation in the stream, while time-decay and sliding window are only deployed based on the predefined time window hyperparameter, hence they have lower average update times of all forgetting techniques.

Although update time is increased for all forgetting techniques, a consequence of their application is the removal of edges in the graph, reducing time and memory requirements. Hence, recommendation times are improved, and the trade-off between accuracy and scalability can be controlled parametrically based on weight threshold ϕ. As the recommendation procedure accounts for the majority of processing time per sample, the total processing time per sample (update + recommendation times) is reduced with all techniques but recency queue. Even if there is a minor overhead in update time with forgetting, its deployment is beneficial in the end as the reduction in recommendation times reduces the overall processing time, which is associated with the size of the graph. As is the case for accuracy and diversity, the influence of LND on the number of edges and recommendation time depends on the density of the dataset, where the reduction is more considerable in denser ones.

We note that these results are averages from the entire simulated stream process. To better visualize the behavior of all algorithms, we plot moving averages of size 5000 for HR@20, number of edges and processing time per sample over time for all datasets. These are presented in Figures 6.8, 6.10 and 6.11, respectively.

From Figure 6.8, local neighborhood decay generally outperforms other techniques throughout most of the time in accuracy. The increase obtained by our technique results from the manner in which items are selected to lose relevance. Since it is applied locally, it ensures that concepts are only forgotten in the presence of newer concepts and when they are not reinforced by new data. The recency queue technique presents similar accuracy to $\text{IGSI}_{\hat{\pi}^t}$ without forgetting as it punishes infrequent items that do not seem to directly reflect the current interests of users. Conversely, $\text{IGSI}_{\hat{\pi}^t}$ without forgetting generally outperforms both time-decay and sliding window, except for ML-10M and GLOBO. These techniques do not distinguish the relevance of edges and simply forget information over time based on intervals.

We assess the statistical significance of these results through McNemar's tests between our proposal LND and every other competing approach, for every dataset using sliding windows

Table 6.22: Optimal hyperparameters per algorithm grouped by dataset.

Technique	Parameter
ML-1M	
LND	$\beta = 0.7, \tau = 0.1, \phi = \alpha^{15}$
recency_queue	$\phi = \alpha^{5}$
time_decay	$\omega = 30$
sliding_window	$\omega = 90$
ML-10M	
LND	$\beta = 0.6, \tau = 0.2, \phi = \alpha^{25}$
recency_queue	$\phi = \alpha^{5}$
time_decay	$\omega = 14$
sliding_window	$\omega = 365$
LFM-1K	
LND	$\beta = 1.0, \tau = 0.0, \phi = \alpha^{5}$
recency_queue	$\phi = \alpha^{0}$
time_decay	$\omega = 60$
sliding_window	$\omega = 180$
PLC-PL	
LND	$\beta = 1.0, \tau = 0.1, \phi = \alpha^{10}$
recency_queue	$\phi = \alpha^{10}$
time_decay	$\omega = 210$
sliding_window	$\omega = 365$
PLC-STR	
LND	$\beta = 0.6, \tau = 0.3, \phi = \alpha^{15}$
recency_queue	$\phi = \alpha^{10}$
time_decay	$\omega = 90$
sliding_window	$\omega = 180$
BOOK	
LND	$\beta = 0.9, \tau = 0.0, \phi = \alpha^{10}$
recency_queue	$\phi = \alpha^{5}$
time_decay	$\omega = 365$
sliding_window	$\omega = 365$
ELEC	
LND	$\beta = 0.6, \tau = 0.2, \phi = \alpha^{10}$
recency_queue	$\phi = \alpha^{5}$
time_decay	$\omega = 365$
sliding_window	$\omega = 365$
GLOBO	
LND	$\beta = 0.5, \tau = 0.8, \phi = \alpha^{20}$
recency_queue	$\phi = \alpha^{10}$
time_decay	$\omega = 1$
sliding_window	$\omega = 1$

Table 6.23: Graph's properties per dataset. The reported properties are number of nodes ($|\mathcal{V}|$), number of edges ($|\mathcal{E}|$), density, average degree, average weighted degree, minimum weighted degree (Min), maximum weighted degree (Max) and number of low weighted degree nodes (#Low). For each dataset, cells from column (Min, Max, #Low) are split into two rows, where the first refer to indegree node information, and the second refers to outdegree node information.

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	**Density**	**Avg. degree**	**Avg. weight. degree**	**(Min, Max, #Low) weighted degree**
ML-1M	3,232	140,787	1.348	43.56	68.16	(0, 1813, 2473) (0, 1916, 2487)
ML-10M	8,721	648,368	0.852	74.34	169.40	(0, 15009, 7338) (0, 15745, 7356)
PLC-PL	26,117	75,858	0.011	2.90	3.89	(0, 139, 19372) (0, 148, 19389)
PLC-STR	40,213	469,825	0.029	11.68	35.84	(0, 11932, 32732) (0, 11937, 32720)
LFM-1K	399,171	2,623,241	0.001	6.57	10.60	(0, 1698, 318626) (0, 1698, 318625)
BOOK	734,918	2,192,619	0.0004	2.98	6.05	(0, 2864, 644863) (0, 2863, 644820)
ELEC	60,842	133,369	0.003	2.19	2.74	(0, 920, 49797) (0, 802, 49496)
GLOBO	35,644	488,778	0.038	13.71	44.06	(0, 17516, 32689) (0, 17904, 32842)

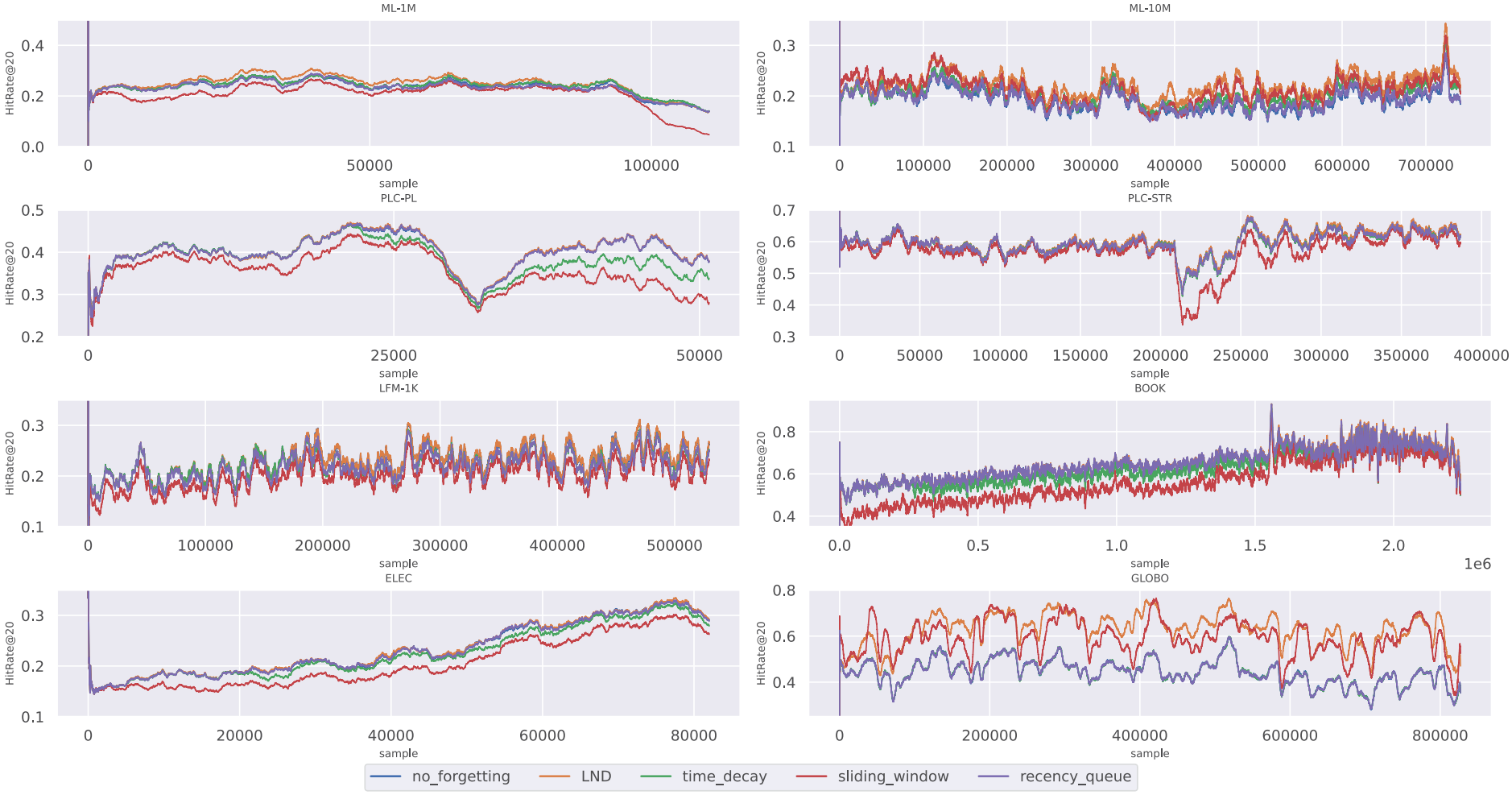


Figure 6.8: Evolution of HitRate@20 with window size $n = 5000$.

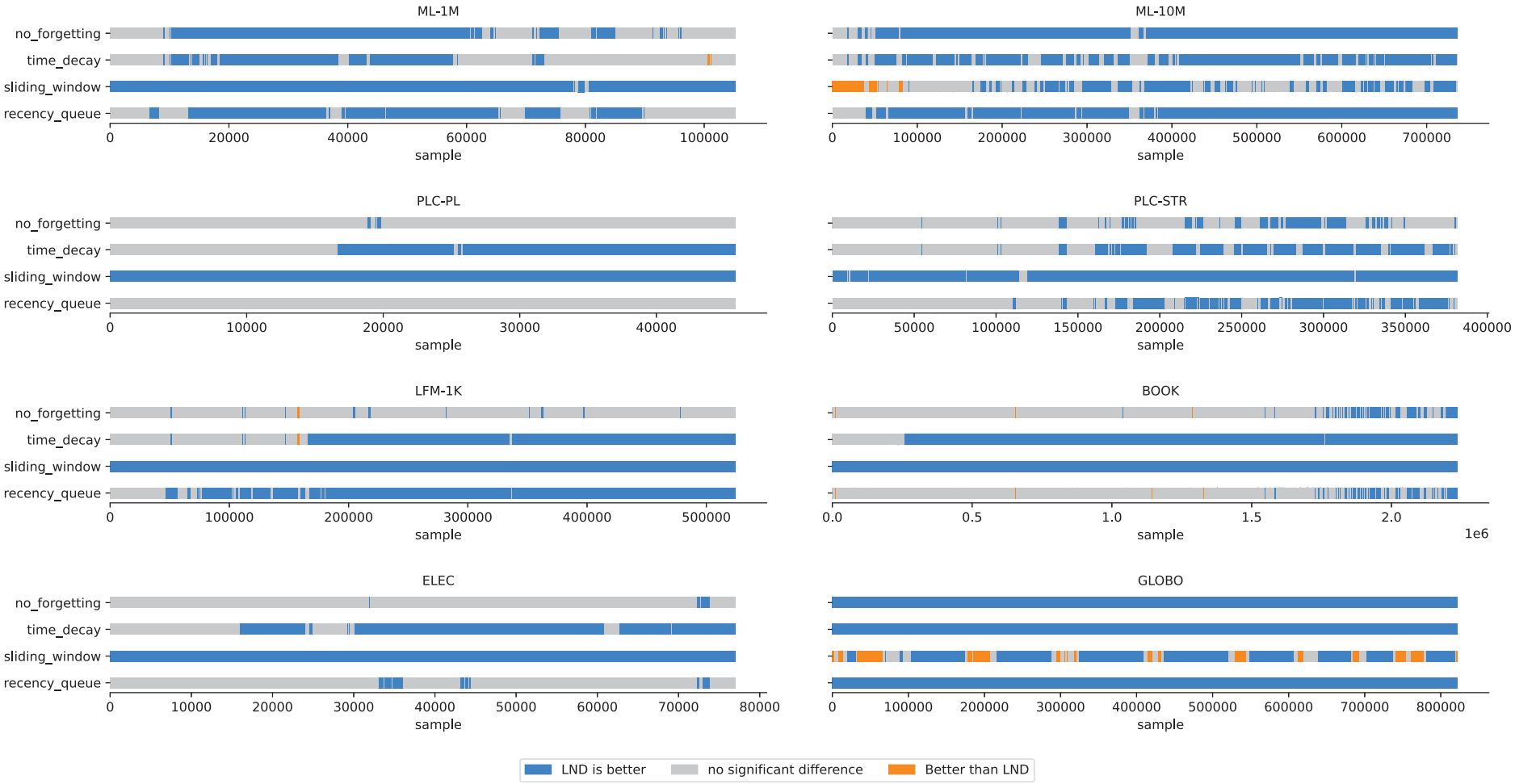


Figure 6.9: McNemar's pairwise test results between LND and other forgetting approaches for all datasets with a confidence level of 99%.

with size $n = 5000$, the same size set for moving averages presented in Figure 6.8. These are shown in Figure 6.9.

The statistical significance tests show that LND is rarely outperformed by any other baseline. Some are clearly visible in Figure 6.8. For example, LND is outperformed by sliding window in some parts of the prequential process for dataset GLOBO. Conversely, it generally outperforms the remaining approaches throughout the prequential process, confirming some of the visible superiority in Figure 6.8, and there is mostly no statistical significant differences where there is overlap between accuracies. Even if there are instances without statistically significant differences in accuracy, there are still other advantages in its deployment, such as potential increases in diversity and reduction in size of the underlying models.

In Figure 6.10 we can see how the underlying graph grows based on the deployed forgetting technique. For $\text{IGSI}_{\hat{\pi}^t}$ without forgetting, the average number of edges tends to grow proportionally to new samples over time. For local neighborhood decay and recency queue, following an initial decrease resulting from the deletion of obsolete edges, it grows steadily, as the insertions and deletions occur continuously, adding new information and removing outdated ones based on recency. For time-decay and sliding window, there is usually a major decrease based on the time interval, followed by increases proportional to the new data. A limitation of these techniques is illustrated on the final samples of ML-1M, where the rate of forgetting is faster than the arrival of new data. This could be corrected by using windows with size based on number of interactions instead of time intervals.

Finally, Figure 6.11 shows the effect of removing edges on the average processing time per sample. For $\text{IGSI}_{\hat{\pi}^t}$ without forgetting, average time increases with the size of the model. With forgetting, the reduction of edges decreases average recommendation time, as the complexity of the recommendation procedure depends on the number of random walk samples and size of neighborhoods. Although there is a minor increase in average update time, as shown in Tables 6.19, 6.20 and 6.21, with exception of recency queue for which is exceptionally high and omitted from some plots, forgetting eventually reduces the average processing time per sample, and the behavior of such reduction is directly related to the technique and dataset.

Overall, extending $\text{IGSI}_{\hat{\pi}^t}$ with our proposed forgetting technique improved accuracy, diversity and average processing time per sample. We present in Table 6.24 the properties of underlying graphs per dataset *after* the application of LND. Comparing to the properties of the original graphs presented in Table 6.23, LND impacts density and the degree distribution, as it removes edges from the graph. Reductions are more significant for the outdegree distribution, as expected, since LND when necessary prunes outgoing edges from a single source. Datasets with a greater reduction in these properties are the ones that were better affected by LND in accuracy, diversity and processing time per sample, e.g., ML-1M, ML-10M, PLC-STR and GLOBO.

We argue that a balance between the relevant evaluated metrics can be obtained through parameter β, which increases accuracy and diversity, τ that may increase accuracy, and the weight threshold ϕ that allows controlling the growth of the graph, which tends to decrease the recommendation time, at the expense of a minor increase in update time. Such decrease may allow the usage of a higher number of random walk samples, which would likely improve accuracy.

6.4 DISCUSSION

In this chapter we have conducted a series of experiments in order to evaluate our proposed approach and related incremental algorithms under simulated data stream scenarios. Our results highlight the effectiveness of our proposed SBRS - $\text{IGSI}_{\hat{\pi}}$ - which generally outperformed

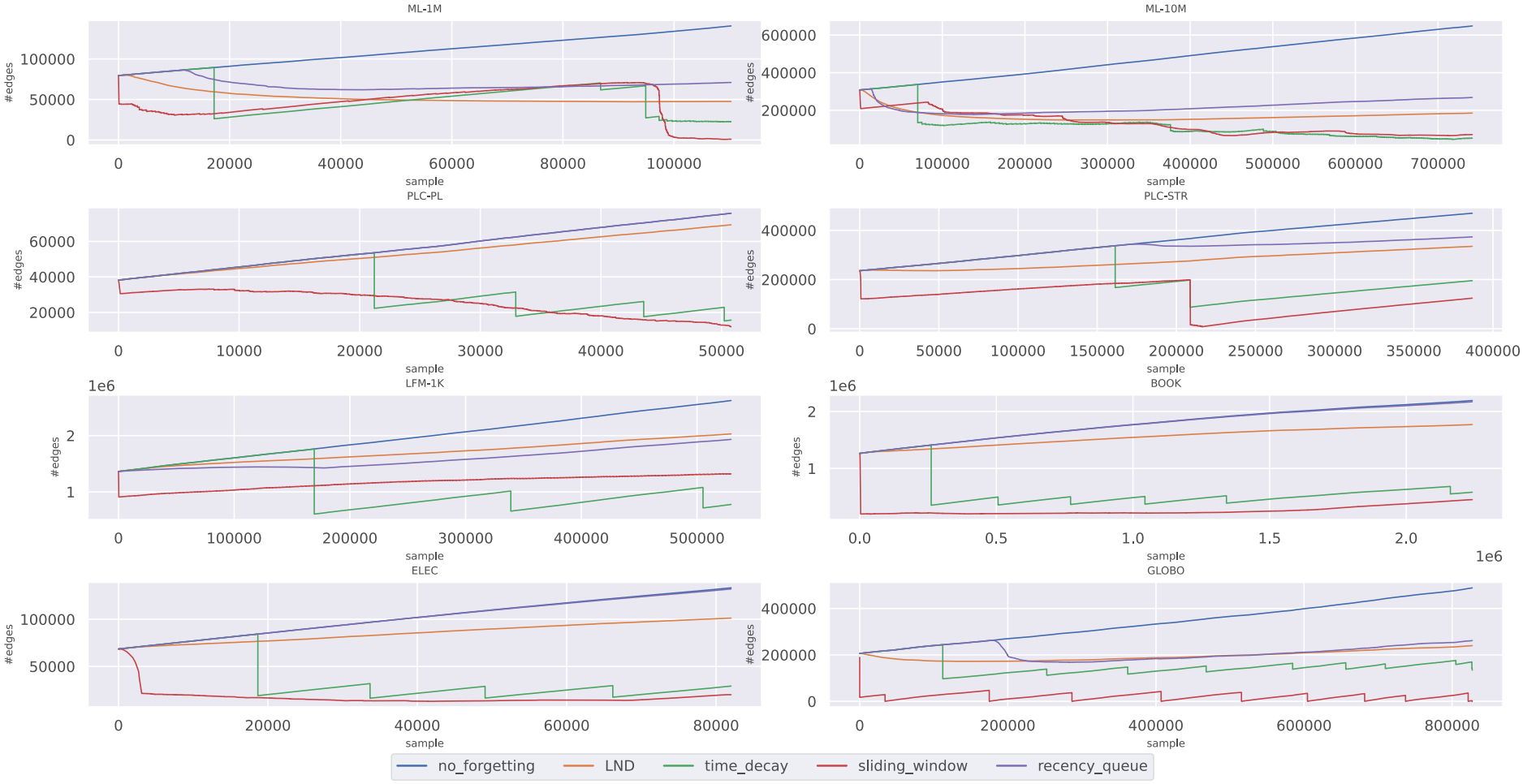


Figure 6.10: Evolution of the number of edges with window size $n = 5000$.

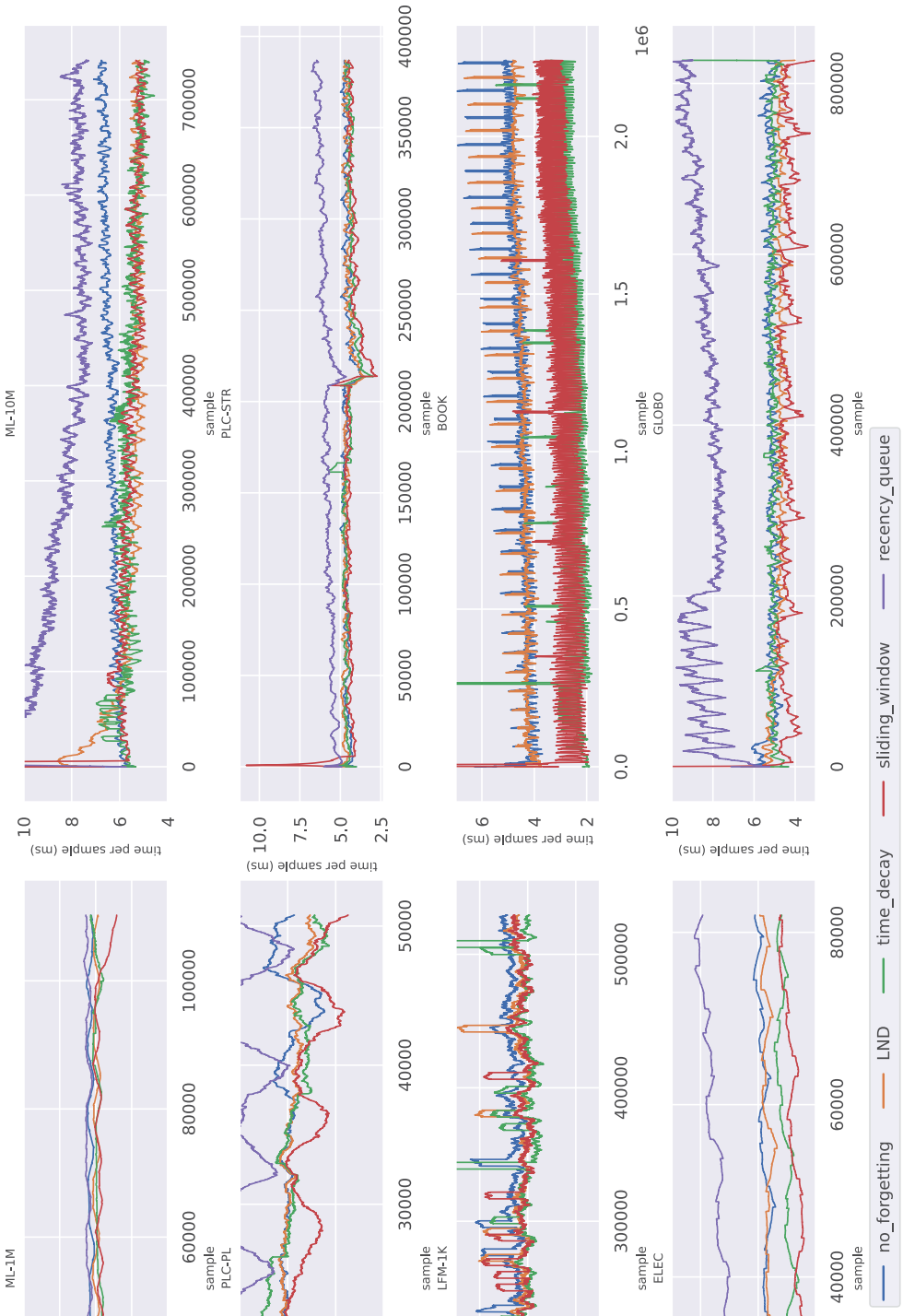


Figure 6.11: Evolution of processing time per sample with window size $n = 5000$.

competing algorithms in terms of accuracy, with very competitive update and recommendation times, thus being a viable option for deployment in online scenarios. Its main limitation, the fact that it stores data that eventually becomes obsolete, is overcome by our proposed forgetting technique, LND. LND reuses the random walk sampling originally generated for providing recommendations to infer structural information of the underlying graphs. This information, together with recency and popularity factors, are used to select and remove obsolete concepts from the graph. Our results suggest that LND is capable of improving scalability and accuracy, by removing such obsolete information, and also diversity by punishing popular items.

Besides the effectiveness of $\text{IGSI}_{\hat{\pi}}$ and LND, these results allows us to revisit the research questions posed in Chapter 1. The first research question asked if graph-based algorithms for collaborative filtering can be used in data stream scenarios. Besides the positive results obtained by $\text{IGSI}_{\hat{\pi}}$, $\hat{\text{P}}^3_\alpha$ and $\hat{\text{RP}}^3_\beta$ were competitive in some domains, with very fast update and recommendation times. Our evaluation suggest that graph-based models with sampling algorithms for recommendation are strong approaches for data stream scenarios.

The performance of $\hat{\text{P}}^3_\alpha$ and $\hat{\text{RP}}^3_\beta$ in other domains leads us to the second question, which asks how to incorporate time-related information into graph-based models, and if such information improves the quality of the recommendations. Our proposal, $\text{IGSI}_{\hat{\pi}}$, presents one way of incorporating sequential information, which is to represent interactions in an incremental directed item-graph, connecting the sequences of user interactions, and reinforcing the relevance of each sequential interaction on the weight of the edges.

Moreover, our results suggest that considering sequential information indeed improves the quality of recommendations, as seen when comparing $\text{IGSI}_{\hat{\pi}}$ to $\hat{\text{P}}^3_\alpha$ and $\hat{\text{RP}}^3_\beta$. Also, by comparing $\text{IGSI}_{\hat{\pi}}$ with different recommendation strategies, we see that further increases can be obtained by considering short-term information, as evident by the improvements obtained by $\text{IGSI}^r_{\hat{\pi}}$ compared to $\text{IGSI}^{Su}_{\hat{\pi}}$. Our results in general showed some limitations of competing approaches, which motivated our contributions that we believe have bridged the gap posed in question three, which asked if current incremental algorithms are sufficient in providing accurate, diverse and scalable recommendations.

Finally, the fourth question asks what techniques can be used to remove obsolete information from ever-growing models and how to deal with issues inherently related to recommender systems and data stream mining. We have shown in our experiments some challenges of deploying forgetting mechanisms and limitations of current approaches, and proposed LND to overcome these issues. The application of LND on $\text{IGSI}_{\hat{\pi}}$ is able to increase its scalability, by reducing the size of the model, its accuracy, by removing obsolete information which is no longer considered in the recommendations, and its diversity, by punishing popular items and boosting the recommendation of items from the long-tail.

Table 6.24: Graph's properties per dataset with LND. The reported properties are number of nodes ($|\mathcal{V}|$), number of edges ($|\mathcal{E}|$), density, average degree, average weighted degree, minimum weighted degree (Min), maximum weighted degree (Max) and number of low weighted degree nodes (#Low). For each dataset, cells from column (Min, Max, #Low) are split into two rows, where the first refer to indegree node information, and the second refers to outdegree node information.

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	**Density**	**Avg. degree**	**Avg. weight. degree**	**(Min, Max, #Low) weighted degree**
ML-1M	3,232	47,540	0.455	14.70	19.21	(0, 490.51, 2307)
						(0, 422.38, 1913)
ML-10M	8,721	185,777	0.244	21.30	25.65	(0, 1118.99, 7056)
						(0, 609.92, 5584)
PLC-PL	26,117	73,028	0.010	2.79	3.68	(0, 132.0, 19374)
						(0, 148.0, 19389)
PLC-STR	40,213	335,214	0.020	8.33	27.38	(0, 11655.94, 32141)
						(0, 11778.08, 31469)
LFM-1K	399,171	2,030,271	0.001	5.08	8.77	(0, 1352.94, 312460)
						(0, 1313.98, 303074)
BOOK	734,918	1,770,607	0.0003	2.40	3.97	(0, 555.06, 613482)
						(0, 422.83, 598751)
ELEC	60,842	101,324	0.003	1.66	2.03	(0, 673.71, 51709)
						(0, 314.41, 49552)
GLOBO	35,644	240,216	0.018	6.73	9.95	(0, 3894.54, 31556)
						(0, 573.17, 28853)

7 CONCLUSIONS

In this chapter, we summarize this research and discuss our contributions, limitations of our proposal and define possible improvements and future work.

7.1 CONTRIBUTIONS

The main objective of this research was to propose a stream-based recommender system designed to work on online settings, tackle the issues inherently related to recommender systems and data stream mining, such as sparsity, accuracy, scalability and concept drift, and provide alternatives to shortcomings of previous algorithms. Throughout this thesis, we have discussed the challenges of developing stream-based recommender systems, and pointed out the importance and relevance of such research.

The main contributions of this research are:

- $\text{IGSI}_{\hat{\pi}^t}$ - a stream-based recommender system that consists in an incremental item-graph that continuously extracts information from the sequential user interactions, allowing the recommendation of items based on short-term and long-term interests, designed to work in data stream scenarios and deal with its underlying issues.

- $\text{IGSI}_{\hat{\pi}^t}$ generates scalable recommendations through simulation of short random walks. We have experimentally showed the advantages of using simulation of random walks in data stream settings. As datasets are typically very sparse, the sampling procedure comes close to convergence with a reasonable number of samples, which results in very competitive recommendation times.

- $\text{IGSI}_{\hat{\pi}^t}$ includes a set of hyperparameters that allows the generation of recommendations based on short-term and long-term interests, and we have studied the impact of such interests. We also have analyzed the impact of different hyperparameters in recommendations. Our results suggest that including long-term information may deteriorate accuracy over time compared to recommendations generated through short-term interests, but a balance between short-term and long-term information can result in improvements in accuracy, as evidenced by the effect of parameter r.

- We have implemented and evaluated two graph-based algorithms that are strong baselines in several recommendation settings (Dacrema et al., 2021), $\hat{\mathbf{P}}^3_\alpha$ (Cooper et al., 2014) and $\hat{\mathbf{RP}}^3_\beta$ (Christoffel et al., 2015), in incremental manner and compared their performance with $\text{IGSI}_{\hat{\pi}^t}$ and other incremental algorithms;

- We evaluated several incremental algorithms on simulated data stream scenarios under different metrics to assess their accuracy, scalability and diversity. Besides analyzing the performance of several algorithms, the evaluation also suggested the effectiveness of $\text{IGSI}_{\hat{\pi}^t}$, which consistently outperformed related incremental algorithms with competitive scalability.

- We proposed a forgetting mechanism, local neighborhood decay (LND), that considers the relevance of items based not only on its recency, but also on *popularity*, *acceptance factors* and *structural information* of the graph;

- LND is designed to overcome the limitations of $\text{IGSI}_{\hat{\pi}^t}$ while also exploiting its advantages, by using random walk sampling not only to *recommend relevant items*, but to *also capture structural information from the graph*, and use it to forget obsolete edges. Our results suggest the effectiveness of LND, which improved scalability, as the removal of obsolete data reduces computational requirements, e.g., time and memory ones, diversity, by including less popular items in the recommendation lists, and accuracy, since it reduces the effect of obsolete data;

- The application of LND in $\text{IGSI}_{\hat{\pi}^t}$ - results in an evolving SBRS, that incrementally and continuously learns based on information generated along the data stream, and also removes information that is deemed as obsolete. Our results suggest the effectiveness of $\text{IGSI}_{\hat{\pi}^t}$ and the positive impact of LND.

7.2 LIST OF PUBLICATIONS

The following list contains the publications that were generated throughout this research:

- Schmitt, M. F. L. and Spinosa, E. J. (2020). Incremental graph of sequential interactions for online recommendation with implicit feedback. In *3rd Workshop on Online Recommender Systems and User Modeling*.

- Schmitt, M. F. L. and Spinosa, E. J. (2022b). Scalable stream-based recommendations with random walks on incremental graph of sequential interactions with implicit feedback. *User Modeling and User-Adapted Interaction*, 32(4):543–573.

- Schmitt, M. F. L. and Spinosa, E. J. (2022a). Forgetting on evolving graphs for accurate and diverse stream-based recommendation. In *Anais do X Symposium on Knowledge Discovery, Mining and Learning*, pages 138–145, Porto Alegre, RS, Brasil. SBC.

- Schmitt, M. F. L. and Spinosa, E. J. (2024). A novel forgetting technique with random walk sampling for scalable and adaptive stream-based recommender systems. *Under review*.

7.3 LIMITATIONS AND FUTURE WORK

Some limitations and possible future work motivated by them are described in the following.

- **Use of time information**. Although our model processes data incrementally as it is generated by the data stream, hence following a naturally chronological sequence, time information could be further exploited and incorporated in our method. For instance, in some domains, the duration of the interaction could be used to filter the relevance of the information. Also, implementing $\text{IGSI}_{\hat{\pi}^t}$ as a session-aware (or session-based) model (Quadrana et al., 2018) by explicitly considering the start and end of user sessions could improve modeling of user interests, specially short-term interests, by inferring such interests based on the most current user session, while long-term interests would be based on all user sessions. Parameter r somewhat accounts for this information, by limiting the generation of recommendations based only on the user's most recent interactions, but we expect that explicitly modeling session information could improve its predictive capabilities.

- **Automatic parameter adjustment**. Every evaluated algorithm has its own set of hyperparameters, that were optimized typically via grid search. Once these hyperparameters were set, the same values defined during training were used throughout the simulated streaming process. However, in real-world systems, as the data distribution is non-stationary and subject to concept drift, the optimal values of hyperparameters are also likely to change over time. Automatically adjusting hyperparameters in the presence of concept drifts are possible extensions that should improve performance (Veloso et al., 2021).

- **Concept drift and cold-start**. These issues are inherently related to data streams and recommender systems, respectively, and are expected to occur in non-stationary settings. Our proposal accounts for concept drift by learning incrementally and also forgetting obsolete information. Inclusion of active mechanisms to detect changes (Al-Ghossein et al., 2018b) and evaluation of algorithms on specific scenarios where concept drift and cold-start are prevalent (Viniski et al., 2021) would strength the applicability of our proposal and SBRS approaches in general.

- **Offline evaluation**. Prequential evaluation works by assessing how well a given method predicts the single next observation, which disregards relevant items that were selected recently or that will be selected in the near future. Also, as users are affected by the recommendations themselves, high predictive performance in offline settings do not necessarily translate to relevant recommendations in practice (Al-Ghossein et al., 2021), and the limitation of such settings should be considered. We have attempted to mitigate this effect by considering datasets from several different scenarios, and evaluating algorithms with different metrics apart from accuracy.

REFERENCES

Adomavicius, G., Bauman, K., Tuzhilin, A., and Unger, M. (2021). Context-aware recommender systems: From foundations to recent developments context-aware recommender systems. In *Recommender Systems Handbook*, pages 211–250. Springer.

Adomavicius, G. and Tuzhilin, A. (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 engineering*, 17(6):734–749.

Agarwal, D., Chen, B.-C., and Elango, P. (2010). Fast online learning through offline initialization for time-sensitive recommendation. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 703–712.

Aggarwal, C. C. et al. (2016). *Recommender systems: the textbook*, volume 1. Springer.

Aggarwal, C. C., Wolf, J. L., Wu, K.-L., and Yu, P. S. (1999). Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 201–212.

Al-Ghossein, M., Abdessalem, T., and Barré, A. (2018a). Dynamic local models for online recommendation. In *Companion Proceedings of the The Web Conference 2018*, pages 1419–1423.

Al-Ghossein, M., Abdessalem, T., and Barre, A. (2021). A survey on stream-based recommender systems. *ACM Computing Surveys (CSUR)*, 54(5):1–36.

Al-Ghossein, M., Murena, P.-A., Abdessalem, T., Barré, A., and Cornuéjols, A. (2018b). Adaptive collaborative topic modeling for online recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 338–346.

Andersen, R., Chung, F., and Lang, K. (2006). Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE.

Anyosa, S. C., Vinagre, J., and Jorge, A. M. (2018). Incremental matrix co-factorization for recommender systems with implicit feedback. In *Companion Proceedings of the The Web Conference 2018*, pages 1413–1418.

Avrachenkov, K., Litvak, N., Nemirovsky, D., and Osipova, N. (2007). Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis*, 45(2):890–904.

Avrachenkov, K., Litvak, N., Nemirovsky, D., Smirnova, E., and Sokol, M. (2011). Quick detection of top-k personalized pagerank lists. In *International Workshop on Algorithms and Models for The Web-Graph*, pages 50–61. Springer.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16.

Backstrom, L. and Leskovec, J. (2011). Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644.

Baeza-Yates, R., Ribeiro-Neto, B., et al. (1999). *Modern information retrieval*, volume 463. ACM press New York.

Bahmani, B., Chowdhury, A., and Goel, A. (2010). Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3).

Baltrunas, L. and Amatriain, X. (2009). Towards time-dependant recommendation based on implicit feedback. In *Workshop on Context-Aware Recommender Systems (RecSys 2009)*.

Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. (2008). Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web*, pages 895–904.

Bell, R. M. and Koren, Y. (2007). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 43–52. IEEE.

Bennett, J., Lanning, S., and Netflix, N. (2007). The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*.

Bifet, A., de Francisci Morales, G., Read, J., Holmes, G., and Pfahringer, B. (2015). Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 59–68.

Bifet, A. and Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM.

Brand, M. (2003). Fast online svd revisions for lightweight recommender systems. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 37–46. SIAM.

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52.

Burke, R. (2007). Hybrid web recommender systems. *The adaptive web*, pages 377–408.

Campos, P. G., Díez, F., and Cantador, I. (2014). Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1):67–119.

Çapan, G., Gündoğdu, İ., Türkmen, A. C., and Cemgil, A. T. (2022). Dirichlet–luce choice model for learning from interactions. *User Modeling and User-Adapted Interaction*, pages 1–38.

Castells, P., Hurley, N. J., and Vargas, S. (2015). Novelty and diversity in recommender systems. In *Recommender systems handbook*, pages 881–918. Springer.

Celma, Ò. (2010). Music recommendation. In *Music recommendation and discovery*, pages 43–85. Springer.

Celma, Ò. and Cano, P. (2008). From hits to niches? or how popular artists can bias music recommendation and discovery. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, pages 1–8.

Chandramouli, B., Levandoski, J. J., Eldawy, A., and Mokbel, M. F. (2011). Streamrec: a real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1243–1246.

Chang, S., Zhang, Y., Tang, J., Yin, D., Chang, Y., Hasegawa-Johnson, M. A., and Huang, T. S. (2017). Streaming recommender systems. In *Proceedings of the 26th international conference on world wide web*, pages 381–389.

Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. *Advances in neural information processing systems*, 24.

Chen, C., Yin, H., Yao, J., and Cui, B. (2013). Terec: A temporal recommender system over tweet stream. *Proceedings of the VLDB Endowment*, 6(12):1254–1257.

Cheng, C., Yang, H., Lyu, M. R., and King, I. (2013). Where you like to go next: Successive point-of-interest recommendation. In *Twenty-Third international joint conference on Artificial Intelligence*.

Chien, Y.-H. and George, E. I. (1999). A bayesian model for collaborative filtering. In *AISTATS*.

Christakopoulou, E. and Karypis, G. (2016). Local item-item models for top-n recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 67–74.

Christoffel, F., Paudel, B., Newell, C., and Bernstein, A. (2015). Blockbusters and wallflowers: Accurate, diverse, and scalable recommendations with random walks. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 163–170.

Cooper, C., Lee, S. H., Radzik, T., and Siantos, Y. (2014). Random walks in recommender systems: exact computation and simulations. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 811–816.

Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198.

Cremonesi, P., Koren, Y., and Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46.

Dacrema, M. F., Boglio, S., Cremonesi, P., and Jannach, D. (2021). A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems (TOIS)*, 39(2):1–49.

Das, A. S., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280.

Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., et al. (2010). The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296.

de Souza Pereira Moreira, G., Ferreira, F., and da Cunha, A. M. (2018). News session-based recommendations using deep neural networks. In *Proceedings of the 3rd workshop on deep learning for recommender systems*, pages 15–23.

de Souza Pereira Moreira, G., Jannach, D., and Da Cunha, A. M. (2019). Contextual hybrid session-based news recommendation with recurrent neural networks. *IEEE Access*, 7:169185–169203.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

Deshpande, M. and Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177.

Desrosiers, C. and Karypis, G. (2011). A comprehensive survey of neighborhood-based recommendation methods. *Recommender systems handbook*, pages 107–144.

Devooght, R., Kourtellis, N., and Mantrach, A. (2015). Dynamic matrix factorization with priors on unknown values. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 189–198.

Diaz-Aviles, E., Drumond, L., Gantner, Z., Schmidt-Thieme, L., and Nejdl, W. (2012a). What is happening right now... that interests me? online topic discovery and recommendation in twitter. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1592–1596.

Diaz-Aviles, E., Drumond, L., Schmidt-Thieme, L., and Nejdl, W. (2012b). Real-time top-n recommendation in social streams. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 59–66.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer.

Ding, Y. and Li, X. (2005). Time weight collaborative filtering. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 485–492.

Domingos, P. M. and Hulten, G. (2001). Catching up with the data: Research issues in mining data streams. In *DMKD*.

Eksombatchai, C., Jindal, P., Liu, J. Z., Liu, Y., Sharma, R., Sugnet, C., Ulrich, M., and Leskovec, J. (2018). Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 world wide web conference*, pages 1775–1784.

Ekstrand, M. D., Riedl, J. T., and Konstan, J. A. (2011). *Collaborative filtering recommender systems*. Now Publishers Inc.

Fogaras, D., Rácz, B., Csalogány, K., and Sarlós, T. (2005). Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358.

Fouss, F., Pirotte, A., Renders, J.-M., and Saerens, M. (2007). Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering*, 19(3):355–369.

Fouss, F., Pirotte, A., and Saerens, M. (2005). A novel way of computing similarities between nodes of a graph, with application to collaborative recommendation. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 550–556. IEEE.

Frigó, E., Pálovics, R., Kelen, D., Kocsis, L., and Benczúr, A. (2017). Online ranking prediction in non-stationary environments. In *Proceedings of the 1st Workshop on Temporal Reasoning in Recommender Systems co-located with RecSys '17 (RecTemp '17).*, pages 28–34. CEUR-WS. org.

Funk, S. (2006). Netflix update: Try this at home. `https://sifter.org/simon/journal/20061211.html`. Accessed in May 2021.

Gama, J. (2012). A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55.

Gama, J., Sebastião, R., and Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37.

Gantner, Z., Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2011). Mymedialite: A free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 305–308.

Garcin, F., Dimitrakakis, C., and Faltings, B. (2013). Personalized news recommendation with context trees. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 105–112.

George, T. and Merugu, S. (2005). A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 4–pp. IEEE.

Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70.

Gomez-Uribe, C. A. and Hunt, N. (2015). The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

Gori, M. and Pucci, A. (2007). Itemrank: A random-walk based scoring algorithm for recommender engines. In *IJCAI*.

Gunawardana, A., Shani, G., and Yogev, S. (2022). Evaluating recommender systems. In *Recommender systems handbook*, pages 547–601. Springer.

Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D., and Zadeh, R. (2013). Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*, pages 505–514.

Harper, F. M. and Konstan, J. A. (2015). The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19.

Haveliwala, T. H. (2003). Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering*, 15(4):784–796.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182.

He, X., Zhang, H., Kan, M.-Y., and Chua, T.-S. (2016). Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53.

Hidasi, B. and Karatzoglou, A. (2018). Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 843–852.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2016). Session-based recommendations with recurrent neural networks. In *ICLR'16*.

Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee.

Huang, X., Wu, L., Chen, E., Zhu, H., Liu, Q., Wang, Y., and Center, B. T. I. (2017). Incremental matrix factorization: A linear feature transformation perspective. In *IJCAI*, pages 1901–1908.

Huang, Y., Cui, B., Jiang, J., Hong, K., Zhang, W., and Xie, Y. (2016). Real-time video recommendation exploration. In *Proceedings of the 2016 International Conference on Management of Data*, pages 35–46.

Huang, Y., Cui, B., Zhang, W., Jiang, J., and Xu, Y. (2015). Tencentrec: Real-time stream recommendation in practice. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 227–238.

Huang, Z., Chen, H., and Zeng, D. (2004). Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):116–142.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106.

Jannach, D., Lerche, L., and Kamehkhosh, I. (2015). Beyond" hitting the hits" generating coherent music playlist continuations with the right tracks. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 187–194.

Jannach, D., Lerche, L., and Zanker, M. (2018). Recommending based on implicit feedback. In *Social Information Access*, pages 510–569. Springer.

Jannach, D. and Ludewig, M. (2017). When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the eleventh ACM conference on recommender systems*, pages 306–310.

Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. (2010). *Recommender systems: an introduction*. Cambridge University Press.

Jawaheer, G., Weller, P., and Kostkova, P. (2014). Modeling user preferences in recommender systems: A classification framework for explicit and implicit user feedback. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 4(2):1–26.

Jeh, G. and Widom, J. (2002). Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543.

Jeunen, O., Van Balen, J., and Goethals, B. (2022). Embarrassingly shallow auto-encoders for dynamic collaborative filtering. *User Modeling and User-Adapted Interaction*, pages 1–33.

Jin, C. (2019). Simulating random walks on graphs in the streaming model. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*.

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

Jugovac, M., Jannach, D., and Karimi, M. (2018). Streamingrec: a framework for benchmarking stream-based news recommenders. In *Proceedings of the 12th ACM conference on recommender systems*, pages 269–273.

Karatzoglou, A., Amatriain, X., Baltrunas, L., and Oliver, N. (2010). Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86.

Khoshneshin, M. and Street, W. N. (2010). Incremental collaborative filtering via evolutionary co-clustering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 325–328.

Kitazawa, T. (2016). Incremental factorization machines for persistently cold-starting online item recommendation. *arXiv preprint arXiv:1607.02858*.

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434.

Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456.

Koren, Y. and Bell, R. (2015). Advances in collaborative filtering. *Recommender systems handbook*, pages 77–118.

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

Koychev, I. (2000). Gradual forgetting for adaptation to concept drift. In *Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning*.

Koychev, I. and Schwab, I. (2000). Adaptation to drifting user's interests. In *Proceedings of ECML2000 Workshop: Machine Learning in New Information Age*, pages 39–46.

Kurucz, M., Benczúr, A. A., and Csalogány, K. (2007). Methods for large scale svd with missing values. In *Proceedings of KDD cup and workshop*, volume 12, pages 31–38. Citeseer.

Latifi, S. and Jannach, D. (2022). Streaming session-based recommendation: When graph neural networks meet the neighborhood. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 420–426.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2020). *Mining of massive data sets*. Cambridge university press.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670.

Li, S., Karatzoglou, A., and Gentile, C. (2016). Collaborative filtering bandits. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 539–548.

Li, X., Barajas, J. M., and Ding, Y. (2007). Collaborative filtering on streaming data with interest-drifting. *Intelligent Data Analysis*, 11(1):75–87.

Linden, G., Smith, B., and York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80.

Liu, N. N., Zhao, M., Xiang, E., and Yang, Q. (2010). Online evolutionary collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 95–102.

Liu, X. and Aberer, K. (2014). Towards a dynamic top-n recommendation framework. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 217–224.

Lofgren, P. A., Banerjee, S., Goel, A., and Seshadhri, C. (2014). Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1436–1445.

Lommatzsch, A. and Albayrak, S. (2015). Real-time recommendations for user-item streams. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1039–1046.

Lops, P., De Gemmis, M., and Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. *Recommender systems handbook*, pages 73–105.

Lovász, L. et al. (1993). Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1):1–46.

Ludewig, M. and Jannach, D. (2018). Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 28(4-5):331–390.

Ludewig, M., Mauro, N., Latifi, S., and Jannach, D. (2021). Empirical analysis of session-based recommendation algorithms: A comparison of neural and non-neural approaches. *User Modeling and User-Adapted Interaction*, 31:149–181.

Matuszyk, P. and Spiliopoulou, M. (2014). Selective forgetting for incremental matrix factorization in recommender systems. In *International conference on discovery science*, pages 204–215. Springer.

Matuszyk, P. and Spiliopoulou, M. (2017). Stream-based semi-supervised learning for recommender systems. *Machine Learning*, 106(6):771–798.

Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A. M., and Gama, J. (2015). Forgetting methods for incremental matrix factorization in recommender systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 947–953.

Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A. M., and Gama, J. (2018). Forgetting techniques for stream-based matrix factorization in recommender systems. *Knowledge and Information Systems*, 55(2):275–304.

McAuley, J., Pandey, R., and Leskovec, J. (2015). Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794.

McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1097–1101.

Miller, A., Fisch, A., Dodge, J., Karimi, A.-H., Bordes, A., and Weston, J. (2016). Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*.

Miranda, C. and Jorge, A. M. (2009). Item-based and user-based incremental collaborative filtering for web recommendations. In *Portuguese Conference on Artificial Intelligence*, pages 673–684. Springer.

Miyahara, K. and Pazzani, M. J. (2000). Collaborative filtering with the simple bayesian classifier. In *Pacific Rim International conference on artificial intelligence*, pages 679–689. Springer.

Mnih, A. and Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. *Advances in neural information processing systems*, 20:1257–1264.

Moore, J. L. and Chen, S. (2013). Taste over time: The temporal dynamics of user preferences. In *ISMIR'13*, pages 401–406.

Nasraoui, O., Cerwinske, J., Rojas, C., and Gonzalez, F. (2007). Performance of recommendation systems in dynamic streaming environments. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 569–574. SIAM.

Natarajan, N., Shin, D., and Dhillon, I. S. (2013). Which app will you use next? collaborative filtering with interactional context. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 201–208.

Nathanson, T., Bitton, E., and Goldberg, K. (2007). Eigentaste 5.0: constant-time adaptability in a recommender system using item clustering. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 149–152.

Nikolakopoulos, A. N. and Karypis, G. (2019). Recwalk: Nearly uncoupled random walks for top-n recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 150–158.

Ning, X., Desrosiers, C., and Karypis, G. (2015). A comprehensive survey of neighborhood-based recommendation methods. *Recommender systems handbook*, pages 37–76.

Oard, D. W., Kim, J., et al. (1998). Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, volume 83. WoUongong.

Ohsaka, N., Maehara, T., and Kawarabayashi, K.-i. (2015). Efficient pagerank tracking in evolving networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 875–884.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Pálovics, R., Benczúr, A. A., Kocsis, L., Kiss, T., and Frigó, E. (2014). Exploiting temporal influence in online recommendation. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 273–280.

Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M., and Yang, Q. (2008). One-class collaborative filtering. In *2008 Eighth IEEE International Conference on Data Mining*, pages 502–511. IEEE.

Panniello, U. and Gorgoglione, M. (2012). Incorporating context into recommender systems: an empirical comparison of context-based approaches. *Electronic Commerce Research*, 12:1–30.

Papagelis, M., Rousidis, I., Plexousakis, D., and Theoharopoulos, E. (2005). Incremental collaborative filtering for highly-scalable recommendation algorithms. In *International Symposium on Methodologies for Intelligent Systems*, pages 553–561. Springer.

Paudel, B., Christoffel, F., Newell, C., and Bernstein, A. (2017). Updatable, accurate, diverse, and scalable recommendations for interactive applications. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 7(1):1–34.

Pazzani, M. J. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial intelligence review*, 13:393–408.

Pazzani, M. J. and Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer.

Pereira, B. L., Ueda, A., Penha, G., Santos, R. L., and Ziviani, N. (2019). Online learning to rank for sequential music recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 237–245.

Phuong, N. D. and Phuong, T. M. (2008). Collaborative filtering by multi-task learning. In *2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*, pages 227–232. IEEE.

Quadrana, M., Cremonesi, P., and Jannach, D. (2018). Sequence-aware recommender systems. *ACM Computing Surveys (CSUR)*, 51(4):1–36.

Rendle, S. (2012). Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):1–22.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461.

Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2010). Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820.

Rendle, S. and Schmidt-Thieme, L. (2008). Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 251–258.

Rendle, S. and Schmidt-Thieme, L. (2010). Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 81–90.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186.

Ricci, F., Rokach, L., and Shapira, B. (2015). Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer.

Ricci, F., Rokach, L., and Shapira, B. (2021). Recommender systems: Techniques, applications, and challenges. *Recommender Systems Handbook*, pages 1–35.

Salakhutdinov, R. and Mnih, A. (2008). Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798.

Sanz-Cruzado, J., Castells, P., and López, E. (2019). A simple multi-armed nearest-neighbor bandit for interactive recommendation. In *Proceedings of the 13th ACM conference on recommender systems*, pages 358–362.

Sarkar, P., Moore, A. W., and Prakash, A. (2008). Fast incremental proximity search in large graphs. In *Proceedings of the 25th international conference on Machine learning*, pages 896–903.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000). Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2002). Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth international conference on computer and information science*, volume 1, pages 27–8. Citeseer.

Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260.

Schmitt, M. F. L. and Spinosa, E. J. (2020). Incremental graph of sequential interactions for online recommendation with implicit feedback. In *3rd Workshop on Online Recommender Systems and User Modeling*.

Schmitt, M. F. L. and Spinosa, E. J. (2022a). Forgetting on evolving graphs for accurate and diverse stream-based recommendation. In *Anais do X Symposium on Knowledge Discovery, Mining and Learning*, pages 138–145, Porto Alegre, RS, Brasil. SBC.

Schmitt, M. F. L. and Spinosa, E. J. (2022b). Scalable stream-based recommendations with random walks on incremental graph of sequential interactions with implicit feedback. *User Modeling and User-Adapted Interaction*, 32(4):543–573.

Schmitt, M. F. L. and Spinosa, E. J. (2024). A novel forgetting technique with random walk sampling for scalable and adaptive stream-based recommender systems. *Under review*.

Shani, G., Heckerman, D., Brafman, R. I., and Boutilier, C. (2005). An mdp-based recommender system. *Journal of Machine Learning Research*, 6(9).

Shi, Y., Larson, M., and Hanjalic, A. (2014). Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)*, 47(1):3.

Siddiqui, Z. F., Tiakas, E., Symeonidis, P., Spiliopoulou, M., and Manolopoulos, Y. (2014). xstreams: Recommending items to users with time-evolving preferences. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, pages 1–12.

Smith, B. and Linden, G. (2017). Two decades of recommender systems at amazon. com. *Ieee internet computing*, 21(3):12–18.

Smyth, B. and McClave, P. (2001). Similarity vs. diversity. In *International conference on case-based reasoning*, pages 347–361. Springer.

Song, Q., Chang, S., and Hu, X. (2019). Coupled variational recurrent collaborative filtering. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 335–343.

Song, Q., Cheng, J., and Lu, H. (2015). Incremental matrix factorization via feature space re-learning for recommender system. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 277–280.

Steck, H. (2019). Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference*, pages 3251–3257.

Subbian, K., Aggarwal, C., and Hegde, K. (2016). Recommendations for streaming data. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2185–2190.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Symeonidis, P., Kirjackaja, L., and Zanker, M. (2020). Session-aware news recommendations using random walks on time-evolving heterogeneous information networks. *User Modeling and User-Adapted Interaction*, pages 1–29.

Tabassum, S., Veloso, B., and Gama, J. (2020). On fast and scalable recurring link's prediction in evolving multi-graph streams. *Network Science*, 8(S1):S65–S81.

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2007). Major components of the gravity recommendation system. *Acm Sigkdd Explorations Newsletter*, 9(2):80–83.

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2008). Investigation of various matrix factorization methods for large recommender systems. In *2008 IEEE International Conference on Data Mining Workshops*, pages 553–562. IEEE.

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

Takács, G. and Tikk, D. (2012). Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 83–90.

Tofani, A., Borges, R., and Queiroz, M. (2022). Dynamic session-based music recommendation using information retrieval techniques. *User Modeling and User-Adapted Interaction*, pages 1–35.

Trevisiol, M., Aiello, L. M., Schifanella, R., and Jaimes, A. (2014). Cold-start news recommendation with domain-dependent browse graph. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 81–88.

Ungar, L. H. and Foster, D. P. (1998). Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, volume 1, pages 114–129. Menlo Park, CA.

Vahedian, F., Burke, R., and Mobasher, B. (2017). Weighted random walk sampling for multi-relational recommendation. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 230–237.

Veloso, B., Gama, J., Malheiro, B., and Vinagre, J. (2021). Hyperparameter self-tuning for data streams. *Information Fusion*, 76:75–86.

Veloso, B., Malheiro, B., Burguillo, J. C., and Foss, J. (2017). Personalised fading for stream data. In *Proceedings of the Symposium on Applied Computing*, pages 870–872.

Vinagre, J. (2016). *Scalable adaptive collaborative filtering*. PhD thesis, Universidade do Porto (Portugal).

Vinagre, J., Al-Ghossein, M., Jorge, A. M., Bifet, A., and Peška, L. (2022a). Orsum 2022-5th workshop on online recommender systems and user modeling. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 661–662.

Vinagre, J., Jorge, A., and Gama, J. (2014a). Evaluation of recommender systems in streaming environments. In *Proceedings of the Workshop on Recommender Systems Evaluation: Dimensions and Design in conjunction with the 8th ACM Conference on Recommender Systems (RecSys 2014)*.

Vinagre, J. and Jorge, A. M. (2012). Forgetting mechanisms for scalable collaborative filtering. *Journal of the Brazilian Computer Society*, 18(4):271–282.

Vinagre, J., Jorge, A. M., Al-Ghossein, M., Bifet, A., and Cremonesi, P. (2022b). Preface to the special issue on dynamic recommender systems and user models. *User Modeling and User-Adapted Interaction*, 32(4):503–507.

Vinagre, J., Jorge, A. M., and Gama, J. (2014b). Fast incremental matrix factorization for recommendation with positive-only feedback. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 459–470. Springer.

Vinagre, J., Jorge, A. M., and Gama, J. (2015a). Collaborative filtering with recency-based negative feedback. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 963–965.

Vinagre, J., Jorge, A. M., and Gama, J. (2015b). An overview on the exploitation of time in collaborative filtering. *Wiley interdisciplinary reviews: Data mining and knowledge discovery*, 5(5):195–215.

Vinagre, J., Jorge, A. M., and Gama, J. (2018a). Online bagging for recommender systems. *Expert Systems*, 35(4):e12303.

Vinagre, J., Jorge, A. M., Rocha, C., and Gama, J. (2021). Statistically robust evaluation of stream-based recommender systems. *IEEE Transactions on Knowledge and Data Engineering*, 33(7):2971–2982.

Vinagre, J., Mário Jorge, A., and Gama, J. (2018b). Online gradient boosting for incremental recommender systems. In *International Conference on Discovery Science*, pages 209–223. Springer.

Viniski, A. D., Barddal, J. P., de Souza Britto Jr, A., and de Campos, H. V. A. (2023). Incremental specialized and specialized-generalized matrix factorization models based on adaptive learning rate optimizers. *Neurocomputing*, 552:126515.

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

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57.

Wang, H., Wang, N., and Yeung, D.-Y. (2015). Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1235–1244.

Wang, J., Hoi, S. C., Zhao, P., and Liu, Z.-Y. (2013). Online multi-task collaborative filtering for on-the-fly recommender systems. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 237–244.

Wang, Q., Yin, H., Hu, Z., Lian, D., Wang, H., and Huang, Z. (2018a). Neural memory streaming recommender networks with adversarial training. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2467–2475.

Wang, S., Cao, L., Wang, Y., Sheng, Q. Z., Orgun, M. A., and Lian, D. (2021). A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)*, 54(7):1–38.

Wang, W., Yin, H., Huang, Z., Wang, Q., Du, X., and Nguyen, Q. V. H. (2018b). Streaming ranking based recommender systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 525–534.

Wang, X., Wang, Y., Hsu, D., and Wang, Y. (2014). Exploration in interactive personalized music recommendation: a reinforcement learning approach. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(1):1–22.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101.

Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., and Tan, T. (2019). Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 346–353.

Wu, Y., DuBois, C., Zheng, A. X., and Ester, M. (2016). Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ninth ACM international conference on web search and data mining*, pages 153–162.

Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., and Sun, J. (2010). Temporal recommendation on graphs via long-and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 723–732.

Xiong, L., Chen, X., Huang, T.-K., Schneider, J., and Carbonell, J. G. (2010). Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM international conference on data mining*, pages 211–222. SIAM.

Yagci, A. M., Aytekin, T., and Gurgen, F. S. (2017). Scalable and adaptive collaborative filtering by mining frequent item co-occurrences in a user feedback stream. *Engineering Applications of Artificial Intelligence*, 58:171–184.

Yu, T., Mengshoel, O. J., Jude, A., Feller, E., Forgeat, J., and Radia, N. (2016). Incremental learning for matrix factorization in recommender systems. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1056–1063. IEEE.

Zhang, J.-D., Chow, C.-Y., and Li, Y. (2014a). Lore: Exploiting sequential influence for location recommendations. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 103–112.

Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38.

Zhang, W., Sun, H., Liu, X., et al. (2014b). An incremental tensor factorization approach for web service recommendation. In *2014 IEEE International Conference on Data Mining Workshop*, pages 346–351. IEEE.

Zhao, X., Zhang, W., and Wang, J. (2013). Interactive collaborative filtering. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 1411–1420.