Sequential recommendation and explanations

Corentin Lonjarret

N° d'ordre NNT : 2021LYSEI003

Thèse de doctorat de l'Université de Lyon
opérée au sein de
L'INSA Lyon

Ecole Doctorale N° 512
Mathématiques et Informatique (InfoMaths)

Spécialité / discipline de doctorat : Informatique

À soutenir publiquement le 12/01/2021 par
Corentin Lonjarret

Sequential recommendation and explanations

Devant le jury composé de:

Josiane Mothe	Professeur, INSPE de l'Académie de Toulouse	Rapporteure
Arnaud Soulet	Maître de conférences HDR, Université de Tours	Rapporteur
Sihem Amer-Yahia	Directrice de recherche, CNRS	Examinatrice
Elisa Fromont	Professeur, Université Rennes 1	Examinatrice
Céline Robardet	Professeur, INSA-Lyon	Directrice de thèse
Marc Plantevit	Maître de conférences HDR, Université Claude Bernard Lyon 1	Co-directeur de thèse
Roch Auburtin	Directeur R&D, Visiativ	Invité

Département FEDORA – INSA Lyon - Ecoles Doctorales – Quinquennal 2016-2020

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **http://www.edchimie-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage **secretariat@edchimie-lyon.fr** INSA : R. GOURDON	**M. Stéphane DANIELE** Institut de recherches sur la catalyse et l'environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 Avenue Albert EINSTEIN 69 626 Villeurbanne CEDEX **directeur@edchimie-lyon.fr**
E.E.A.	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** **http://edeea.ec-lyon.fr** Sec. : M.C. HAVGOUDOUKIAN **ecole-doctorale.eea@ec-lyon.fr**	**M. Gérard SCORLETTI** École Centrale de Lyon 36 Avenue Guy DE COLLONGUE 69 134 Écully Tél : 04.72.18.60.97 Fax 04.78.43.37.17 **gerard.scorletti@ec-lyon.fr**
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** **http://e2m2.universite-lyon.fr** Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : H. CHARLES **secretariat.e2m2@univ-lyon1.fr**	**M. Philippe NORMAND** UMR 5557 Lab. d'Ecologie Microbienne Université Claude Bernard Lyon 1 Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69 622 Villeurbanne CEDEX **philippe.normand@univ-lyon1.fr**
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** **http://www.ediss-lyon.fr** Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : M. LAGARDE **secretariat.ediss@univ-lyon1.fr**	**Mme Sylvie RICARD-BLUM** Institut de Chimie et Biochimie Moléculaires et Supramoléculaires (ICBMS) - UMR 5246 CNRS - Université Lyon 1 Bâtiment Curien - 3ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tel : +33(0)4 72 44 82 32 **sylvie.ricard-blum@univ-lyon1.fr**
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** **http://edinfomaths.universite-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 **infomaths@univ-lyon1.fr**	**M. Hamamache KHEDDOUCI** Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tel : 04.72.44.83.69 **hamamache.kheddouci@univ-lyon1.fr**
Matériaux	**MATÉRIAUX DE LYON** **http://ed34.universite-lyon.fr** Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction **ed.materiaux@insa-lyon.fr**	**M. Jean-Yves BUFFIÈRE** INSA de Lyon MATEIS - Bât. Saint-Exupéry 7 Avenue Jean CAPELLE 69 621 Villeurbanne CEDEX Tél : 04.72.43.71.70 Fax : 04.72.43.85.28 **jean-yves.buffiere@insa-lyon.fr**
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** **http://edmega.universite-lyon.fr** Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction **mega@insa-lyon.fr**	**M. Jocelyn BONJOUR** INSA de Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69 621 Villeurbanne CEDEX **jocelyn.bonjour@insa-lyon.fr**
ScSo	**ScSo*** **http://ed483.univ-lyon2.fr** Sec. : Véronique GUICHARD INSA : J.Y. TOUSSAINT Tél : 04.78.69.72.76 **veronique.cervantes@univ-lyon2.fr**	**M. Christian MONTES** Université Lyon 2 86 Rue Pasteur 69 365 Lyon CEDEX 07 **christian.montes@univ-lyon2.fr**

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Abstract

Recommender systems have received a lot of attention over the past decades with the proposal of many models that take advantage of the most advanced models of Deep Learning and Machine Learning. With the automation of the collect of user actions such as purchasing of items, watching movies, clicking on hyperlinks, the data available for recommender systems is becoming more and more abundant. These data, called implicit feedback, keeps the sequential order of actions. It is in this context that sequence-aware recommender systems have emerged. Their goal is to combine user preference (long-term users' profiles) and sequential dynamics (short-term tendencies) in order to recommend next actions to a user.

In this thesis, we investigate sequential recommendation that aims to predict the user's next item/action from implicit feedback. Our main contribution is REBUS, a new metric embedding model, where only items are projected to integrate and unify user preferences and sequential dynamics. To capture sequential dynamics, REBUS uses frequent sequences in order to provide personalized order Markov chains. We have carried out extensive experiments and demonstrate that our method outperforms state-of-the-art models, especially on sparse datasets. Moreover we share our experience on the implementation and the integration of REBUS in myCADservices, a collaborative platform of the French company Visiativ.

We also propose methods to explain the recommendations provided by recommender systems in the research line of explainable AI that has received a lot of attention recently. Despite the ubiquity of recommender systems only few researchers have attempted to explain the recommendations according to user input. However, being able to explain a recommendation would help increase the confidence that a user can have in a recommendation system. Hence, we propose a method based on subgroup discovery that provides interpretable explanations of a recommendation for models that use implicit feedback.

Keywords: Recommender Systems, Collaborative Filtering, Sequential Recommendation, Explanations

Résumé

Ces dernière années, les systèmes de recommandation ont reçu beaucoup d'attention avec l'élaboration de nombreuses propositions qui tirent parti des nouvelles avancées dans les domaines du Machine Learning et du Deep Learning. Grâce à l'automatisation de la collecte des données des actions des utilisateurs tels que l'achat d'un objet, le visionnage d'un film ou le clic sur un article de presse, les systèmes de recommandation ont accès à de plus en plus d'information. Ces données sont des retours implicites des utilisateurs (appelé « implicit feedback » en anglais) et permettent de conserver l'ordre séquentiel des actions de l'utilisateur. C'est dans ce contexte qu'ont émergé les systèmes de recommandations qui prennent en compte l'aspect séquentiel des données. Le but de ces approches est de combiner les préférences des utilisateurs (le goût général de l'utilisateur) et la dynamique séquentielle (les tendances à court terme des actions de l'utilisateur) afin de prévoir la ou les prochaines actions d'un utilisateur.

Dans cette thèse, nous étudions la recommandation séquentielle qui vise à prédire le prochain article/action de l'utilisateur à partir des retours implicites des utilisateurs. Notre principale contribution, REBUS, est un nouveau modèle dans lequel seuls les items sont projetés dans un espace euclidien d'une manière qui intègre et unifie les préférences de l'utilisateur et la dynamique séquentielle. Pour saisir la dynamique séquentielle, REBUS utilise des séquences fréquentes afin de capturer des chaînes de Markov d'ordre personnalisé. Nous avons mené une étude empirique approfondie et démontré que notre modèle surpasse les performances des différents modèles de l'état de l'art, en particulier sur des jeux de données éparses. Nous avons également intégré REBUS dans myCADservices, une plateforme collaborative de la société française Visiativ. Nous présentons notre retour d'expérience sur cette mise en production du fruit de nos travaux de recherche.

Enfin, nous avons proposé une nouvelle approche pour expliquer les recommandations fournies aux utilisateurs. Le fait de pouvoir expliquer une recommandation permet de contribuer à accroître la confiance qu'un utilisateur peut avoir dans un système de recommandation. Notre approche est basée sur la découverte de sous-groupes pour fournir des explications interprétables d'une recommandation pour tous types de modèles qui utilisent comme données d'entrée les retours implicites des utilisateurs.

Mots clés: Système de recommandation, Filtrage collaboratif, recommandation séquentielle, Explications.

Remerciements

Je remercie Josiane Mothe, Professeur à l'INSPE de l'Académie de Toulouse et Arnaud Soulet, Maître de conférences HDR à l'Université François Rabelais de Tours, d'avoir accepté le rôle de rapporteur de ma thèse, ainsi que pour leur travail de lecture approfondie.

Je remercie également Sihem Amer-Yahia, Directrice de recherche CNRS et Elise Fromont, Professeur à l'Université Rennes 1, d'avoir accepté de participer au jury de ma thèse.

Je remercie l'entreprise Visiativ qui m'a accueilli durant ces trois années et m'a permis de réaliser la majorité de mes expériences rapportées dans cette thèse. Je tiens à remercier tout particulièrement Roch Auburtin, Directeur R&D de Visiaitv, pour son encadrement et sa bienveillance.

Par ailleurs, je tiens à remercier mes encadrants de thèse, Céline Robardet, Professeur à l'INSA de Lyon et Marc Plantevit, Maître de conférences HDR à l'Université Claude Bernard Lyon 1 qui m'ont guidé et aidé durant ces trois années de thèse. Leur encadrement scientifique m'a été d'un grand soutien, en particulier dans les situations difficiles, et m'a permis de prendre le recul nécessaire sur mes travaux de recherche. Je remercie les doctorants de l'équipe DM2L notamment Romain, Alexandre et Aimene pour leur aide précieuse.

Enfin, je remercie ma famille et mes amis proches qui m'ont toujours soutenu durant ces trois années. Je tiens tout particulièrement à remercier ma femme, Mathilde, qui a toujours été à mes côtés et qui m'a encouragé et motivé dans mes nouveaux projets.

Table of contents

Notations

$\mathcal{R}$	Recommender system.
U	User set.
I	Item set.
S	Set of user sequences.
F	Set of frequent substrings of S. $F = \{\langle 1\rangle, \langle 2\rangle, \langle 4\rangle, \langle 5\rangle, \langle 4,5\rangle\}$
s_u	Sequence of items associated to user u. $s_u = \langle 1,2,3,2,2,\ 4,5\rangle$
I_{s_u}	Set of items that appear in s_u. $I_{s_u} = \{1,2,3,4,5\}$, $\quad I_{s_u{}^{[1,4]}} = \{1,2,3\}$
m_{s_u}	Sequence of F that best matches s_u. $m_{s_u} = \langle 4,5\rangle$
${s_u}^t$	Item that appears in s_u at position t. ${s_u}^5 = 2$
${m_{s_u}}^t$	Item that appears in m_{s_u} at position t. ${m_{s_u}}^2 = 5$
$s_u^{[1,t[}$	Substring of s_u starting at position 1 and ending at position $t-1$. $s_u{}^{[1,5[} = \langle 1,2,3,2\rangle$
$m_{s_u^{[1,t[}}$	Sequence of F that best matches $s_u^{[1,t[}$. $m_{s_u}{}^{[1,5[} = \langle 2\rangle$
$\widehat{p}_{u,i,t}$	Prediction that measures the probability for user u to choose item i while only considering $s_u^{[1,t[}$.
$>_{u,t}$	Personalized total order of user u at position t.
$\mathcal{R}(s_u)$	Recommendations produces by $\mathcal{R}$ for s_u.
$\mathcal{R}_{order}(s_u)$	Ordered recommendations produces by $\mathcal{R}$ for s_u.
$r_i^{s_u}$	Numerical value that measures the probability for user u to choose item i.
$i^\star$	Top-1 recommended item (item to explain).
$N_1(s_u)$	Neighborhood of s_u made of sequences by removing one or several items from s_u.
$N_2(s_u)$	Neighborhood of s_u made of sequences where only the items order of s_u changes.
$\mathcal{I}$	Subgroup description language for **UPSD** defined as all possible sets of items from I.
$\mathcal{S}$	Subgroup description language for **SDSD** made of all sequences of items of I without repetition.
d	A descriptive pattern.
$SG(d)$	Pattern cover of d.

Chapter 1

Introduction

1.1 Context

This thesis has been implemented in both Visiativ company and DM2L group thanks to a CIFRE grant, i.e., an academia/industry partnership. Visiativ is a French software editor company that provides solutions to stimulate innovation, to streamline customer/supplier relationships, to improve information sharing and to develop knowledge. To this end, this company bases its development on several complementary businesses: the integration of specialized software solutions of *Dassault Systèmes*, the edition of a community-oriented collaborative platforms, as well as management consulting and financing innovation.

One important feature of Visiativ is the *myCADservices* collaborative platform that provides to engineers and design offices, a set of applications and services related to Computer Aided Design (CAD). CAD softwares include many functionalities and evolve regularly, which requires users to be kept informed. To this end, they use the documentation made available on the platform, via the *resource center* functionality. This digital documentation is made of tutorials, best practices and divers documents on CAD products.

At the beginning of my PhD, three years ago, *myCADservices* was not featured with a recommender system: no resource was recommended to the users who only had access to digital documents using potentially time-consuming search engines. Hence, they were able to only (1) filter resources by category, domain, or product, (2) search through a text query, and (3) sort document according to their popularity or date. All these features were accessible within an information retrieval system that waited for explicit query from the user without suggesting itself relevant resources to the users. That was obviously a limitation of the system because users were not omniscient (i.e., they did not always know what they searched and how to search it), some users were not aware of new updates, or specific software features. This lack of interaction between the system and the users was at the origin of my work whose main goal is to feature the *myCADservices* platform with a recommender system to make to the user personalized suggestions.

Recommending to a specific user documents that may interest him requires to capture both his preferences (i.e., the general interest of the user – for instance, which software he uses – also known as the long-term dynamics) and his short-term dynamics which is related to his recent actions allowing to characterize what he is doing and what he is going to do. We claim that the simultaneous consideration of these two aspects is important in the context

of *myCADservices* because the digital resources require a variable level of expertise to be understood (i.e., there may exist a "learning path" between them). Therefore, this leads us to study and tackle a problem much wider than its application to Visiativ context: the sequential recommendation problem.

1.2 Sequential recommendation

As society becomes ever more digital, the number of situations where users are faced with a huge number of choices increases. Digital platforms such as Amazon, Youtube, or Netflix, provide more and more digital contents or digital descriptions of real objects called hereafter items. In this context, a selection process by exhaustive examination of all possibilities is not feasible. To overcome this, and improve the user experience, these platforms generally use two tools:

- *Information Retrieval (IR) system* that allows users to query the items or category of items they are looking for. Then, depending on the algorithm used, the retrieval system returns the items that best match the user's query.
- *Recommender Systems (RS)* that automatically generates recommendations for users based on their previous interactions with items. A benefit of recommender system is that it allows users to discover relevant but also unexpected items.

Information Retrieval and Recommender Systems are complementary: The first one allows to give items according to a query where the second one gives items according to users' tastes. Notice that their interaction is not empty. Indeed, the results of a query can be sorted according to some user preference that has to be modeled by the system.

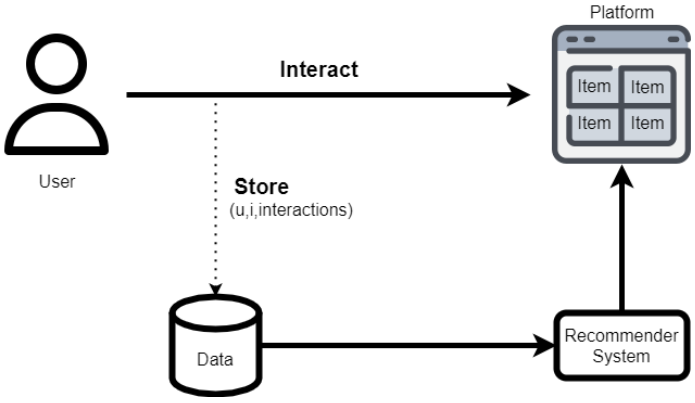

Figure 1.1: Global overview of interactions between users, recommender systems and a platform that display items.

In this thesis, we focus on recommender systems which are widely used in different domains (Aggarwal, 2016) such as e-commerce websites (e.g. Amazon and Alibaba), news websites (e.g. Google news), streaming platforms (e.g. Spotify and Youtube) and social networks (e.g. Facebook and Twitter). One of the main goals of recommendation systems is to create and maintain user faithfulness, and for this purpose, they have to provide items that

users want to interact with. Figure 1.1 gives a general description of what a recommender system does: users interact with the items available on the platform, these interactions are stored in order to be used to recommend new items. Such interactions are called feedback and are divided into two types:

- Implicit feedback which is automatically recorded as soon as the user interacts (e.g. buys/watches/clicks) with an item;
- Explicit feedback that is given by the user after interacting with an item. It usually takes the form of a rating (e.g., 1 to 5 stars) that gives an opinion on an item.

Despite the fact that the information contained in the explicit feedback is more precise for a recommender system, it is obtained at the cost of additional actions for the user who is not always ready to do so. This is why most of recommender systems use implicit data (Rendle et al., 2009).

In general, implicit feedback is ordered by time, allowing recommender systems to capture sequential dynamics while capturing general user tastes (i.e. user preferences). This kind of recommender systems is part of the sequential recommendation model. Therefore, it is through the combination of these two aspects (i.e. user preferences and sequential dynamics) that we obtain personalized and relevant recommendations for each user without using additional information such as user profiles or item content information. As with explicit feedback, information about users or items are often unavailable or may need to be processed before it can be used. Note that recommender systems that use information other than feedback are part of Content/Context-based recommender systems and those that only use feedback fall into the family of collaborative filtering models. We will discuss these two types of recommendation systems in more details in Chapter 2.

Considering (1) that on *myCADservices* platform, it is possible to collect implicit feedback from actions of the users, and (2) that recommendations may benefit from taking into account the order relationships between digital documents, we consider the problem of sequential recommendation whose goal is to predict the next user item/action from an implicit feedback. Behind the general problem of sequential recommendation leads, we will consider several challenges.

1.3 Problems addressed in this thesis

In this thesis, our main objective is to devise and transfer to Visiativ a sequential recommendation model. Sequential recommendation have been widely investigated for two decades. The first recommender systems focused exclusively on modeling user preferences (Koren and Bell, 2015, Koren et al., 2009) and did not consider sequential dynamics. However, by using sequential dynamics present in implicit feedback, the sequential recommendation models outperform the traditional recommender systems. The problem of sequential recommendation lies in the successful combination of the whole user's history and his recent actions (sequential dynamics) to provide personalized recommendations. To capture the sequential dynamics existent models use the concept of Markov Chains which models the probability of choosing an item knowing an ordered list of already selected items. However the seminal models only use fixed-order Markov Chains regardless of the users and their considered items. On the

other hand, recent approaches based on deep neural network architecture considers varying sequence lengths in nonlinear models which are much more difficult to interpret and requires a high level of expertise to be set up and maintained. To cope with these limitations, the first problem addressed in this thesis is:

- ***Problem #1****: How to define a simple yet efficient sequential recommendation model?*

As previously mentioned, this thesis is motivated by the industrial case study brought by Visiativ and the need to feature the myCADservices platform with a recommender system. Defining a prototype and assess it on several benchmarks according to several competitors is far from actual use in practice. This leads to the second problem addressed in this thesis.

- ***Problem #2:*** *How to implement and assess our recommender system in the myCADservices platform?*

Despite their obvious interest, recommender systems behave as black boxes which raises many ethical issues and may limit their usage in practice. Indeed, functioning as a black box results in a lack a transparency and may have a negative impact on the confidence the user has for the related services. As a user, one may wonder why a specific item is recommended? Is it fully relevant to the user? It even raises ethical questions about the choice made in some cases (Gender/Race etc). In this context, Explainable AI has received a lot of attention over the past decade, with many proposed methods explaining black box models. One of the best known is certainly Lime from Ribeiro et al. Ribeiro et al. (2016), that aims to understand the model by perturbing the input of data samples and observing how the predictions change. However LIME returns explanation trough a vector of relative importance of features which create an unclear coverage because there is no way of knowing how generalizable the explanation is. Unclear generalization can lead to low human precision and misleading interpretation. This is why the same authors propose a new method call aLime Ribeiro et al. (2018) that cope this problem of unclear generalization. It uses the same idea as Lime but the explanations are described by if-then rules called "Anchor". In their paper, the authors argue that Anchors are intuitive, easy to understand, and have extremely clear coverage. This leads to the last problem tackled in this thesis:

- ***Problem #3:*** *How to explain the recommendation made by any (sequential-based) recommender system?*

1.4 Contributions

Given the problems we tackle in this thesis, our main contributions are the following.

A novel model for sequential recommendation based on the embedding of frequent sequences:

We propose **REBUS** – for Recommendation Embedding Based on freqUent Sequences – that uses frequent sequences to capture the sequential dynamics and identify the part of user history that is the most relevant for recommendation. These sequences allow to estimate Markov Chains of variable orders. **REBUS** uses a unified metric embedding model based on user preferences and user sequential dynamics. An extensive empirical study demonstrate

that our model outperforms state-of-the-art sequential recommendation models including deep neural neural based models, especially on sparse datasets. We also show that the subsequences identified by the model are relevant and makes it possible to have insight of the recommendations

The implementation of the recommender system in an industrial context: the Visiativ experience:

REBUS has not remained in the prototype, it has been implemented in an industrial context and now used in *myCADservices* platform. We present how we have chosen and implemented such a recommender system and the strategy adopted to monitor its performances in production (A/B Testing and choice of key performance indicators). Furthermore, it was a great opportunity to test performance of **REBUS** in a real use case instead of public benchmarks.

A novel model-agnostic method for explaining the sequential recommendations:

We propose a method rooted in subgroup discovery to explain the recommendations made by any recommender system. This method, inspired from aLime, considers different pattern languages (i. e., itemsets and sequences) and consist of two main steps. The user sequence is perturbed. For each perturbation, the recommendation of the studied recommender system is stored. Then, we use subgroup discovery to give intuitive explanation about the recommendation. This makes it possible to identify which items in the user history were used for the recommendation, and if the order of the user actions accounts or not.

1.5 Structure of the thesis

The remainder of this thesis is organized as follows:

- Chapter 2 first provides in a general overview of recommender systems. Then, we introduce the problem of sequential recommendation and present an extensive review of the literature related to this problem.

- In Chapter 3, we presents our first contribution which tackle the problem of sequential recommendation (prediction of the next item). We propose **REBUS** a unified metric model that embeds items based on user preferences and sequential dynamics. The thorough empirical study demonstrates that our model outperforms state of art models of sequential recommendation. This contribution has been partially published in the proceeding of the French conference *Extraction et Gestion de Connaissances, EGC 2019* (Lonjarret et al., 2019). Then an extended and improved version of **REBUS** is currently under review for the journal Data Mining and Knowledge Discovery DAMI (accepted on October 24th) Lonjarret et al. (2021).

- Chapter 4 is dedicated to the explanation of sequential recommendations based on subgroup discovery. This contribution has been published in the proceeding of the 7th IEEE International Conference on Data Science and Advanced Analytics DSAA 2020 (Lonjarret et al., 2020)

- We explain, in Chapter 5, how we chose and implement a recommender system in an industrial context as well as the strategy adopted to monitor its performances in production (A/B Testing and choice of key performance indicators) in Visiativ's *myCADservices* platform.
- Chapter 6 concludes and give future directions of this work.

1.6 List of publications

Peer-reviewed French national conferences:

- **Corentin Lonjarret**, Marc Plantevit, Céline Robardet and Roch Auburtin. *Recommandation séquentielle à base de séquences fréquentes.* In Extraction et Gestion des Connaissances : EGC 2019, Metz, France, January 21-25, 2019, pages 267–272.

Peer-reviewed international conferences with proceedings:

- **Corentin Lonjarret**, Céline Robardet, Marc Plantevit, Roch Auburtin, and Martin Atzmueller. *Why should i trust this item ? explaining the recommendations of any model.* In 2020 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2020, pages 526–535.

International journals:

- **Corentin Lonjarret**, Céline Robardet, Marc Plantevit and Roch Auburtin. *Sequential recommendation with metric models based on frequent sequences.* In Data Mining and Knowledge Discovery. (Accepted in October 2020).

Chapter 2

Sequential recommendation: State of the Art

In this chapter, we introduce the problem of sequential recommendation whose goal is to predict the next user item/action from an implicit feedback. Before reviewing related work on sequential recommendation models, we provide in Section 2.1 an overview of Recommender Systems (RS) that includes a presentation of their objectives, the different types of input data they use, a typology of the different prediction function and their formulation in term of prediction tasks. In section 2.2, we introduce general recommendation models which include traditional models where sequential information (the order of actions performed by a user) is not considered. Then, in Section 2.3, we present sequential recommendation models. Finally, we describe in Section 2.4 the evaluation protocol for sequential recommender systems.

2.1 Overview of Recommender Systems

From a business point of view, the primary goal of a recommender system is to increase the profits generated by a company. Indeed, by providing the right item to the right user, the recommender system allows to increase sales in the case of an e-commerce website (e.g. Amazon, Alibaba , Ebay, etc...) or to increase the time spent by a user on a digital platform in the case of a Entertainment website or Social Network (e.g. Youtube, Spotify, Netflix, Facebook, Twitter, etc...).

From the user's point of view, the goal of a recommender system is to improve his experience on the platform by selecting the most relevant items and filtering the most irrelevant ones. Providing a good recommendation reduces the search time for users and increases their efficiency. In fact, digital platforms are increasingly providing more and more content, which makes it almost impossible for a user to exhaustively explore all available items. Additionally, being able to provide an explanation of a recommended item would help to increase the confidence that the user have in the recommender system, and thus increase the user experience. Full details will be further discussed in Chapter 4.

Because of their great usefulness, recommender systems are now present in many applications in various domains: E-commerce, Entertainment, Services, Social Network, Online News, etc... Each domain has its own objectives but it is still linked to the two main objectives presented above: (1) To increase the revenue of a company and (2) to improve the

user experience. To achieve this, recommender system has to be able to handle the following operational and technical objectives (Aggarwal, 2016):

- ***Accuracy:*** The primary function of a recommender system is to provide users with relevant items. The more attractive an item is to a user, the more likely it is to be consumed;

- ***Novelty and Serendipity:*** A recommender system should be able to provide items that will surprise the user (Hurley and Zhang, 2011). Novelty refers to the ability of a recommender system to provide items unknown to the user but in a category of items that he likes. Serendipity, on the other hand, refers to the ability of a recommender system to provide unexpected items (i.e. an item belonging to a category unknown to the user). Novelty and Serendipity are two important functions for a recommender system, which allow recommender systems not to recommend only items already known by the user. Indeed, the user may lose interest in the platform if the system only provides him items that he already knows. In addition, providing serendipitous items can create new interests for users and thus increases the profits generated by the platform in the long term. However, attention should be paid to the proportion of novel/serendipitous items provided.

- ***Diversity:*** Recommender systems generally provide a list of the k most relevant items. This list should be composed of items of different types to increase the chances that the user likes one of the listed items (Aggarwal, 2016). Diversity should not be confused with novelty. The novelty refers to how different the recommended item are, with respect to "what has already been seen by the user" while Diversity is related to how different the items are with respect to each other (Castells et al., 2011).

Much of the work on recommender systems puts aside novelty and diversity aspects to focus on the accuracy of recommendations. Indeed, it is more natural to optimize a model on the accuracy of its recommendations, while novelty and diversity can be obtained by a post-processing in order to find the right proportion of new items to insert in the list of recommended items. In certain application domains such as news recommendation or computational advertising, it is important to bring novelty and diversity to keep the interest of the user. In these cases, models using multi-armed bandit algorithms (Li et al., 2010, Louëdec et al., 2015) can be relevant. In this thesis, we also focus on improving the accuracy of recommendations and give little importance to novelty and diversity.

2.1.1 Recommender systems in a nutshell

Recommender systems can be seen as specific prediction models. Their common points are that they both use training data to learn a model capable to make predictions. The specificity of recommender systems lies in the training data, and also the output of the model. The training data are made of the items that a user has enjoyed so far, among the whole set of possible items. As a user generally considers a very small number of items compared to the whole set of items, this makes the data very sparse. Considering the output data, Machine learning models usually predict few values, while Recommender Systems associate a value to each item of the whole set of items, that is generally huge.

These two differences are important enough to require the development of methods specific to the recommendation problem, even though there are commonalities with classical machine learning methods[1]. From Figure 2.1, we can see the common points between machine learning and recommender models. A model whose parameters Θ are learnt with training data and that is then used to make predictions from input data. Learning the model means adjusting the parameters Θ using the training input and the output of the prediction function via a loss function that evaluates the veracity of the model. The figure also illustrates the specificities of recommender models: the type of input data, detailed in Subsection 2.1.2, the prediction models that can predict a large set of output, presented in Subsection 2.1.3, and the prediction tasks (the type of output considered), as well as the loss functions used to evaluate the difference between predicted and expected data, explained in Subsections 2.1.4.

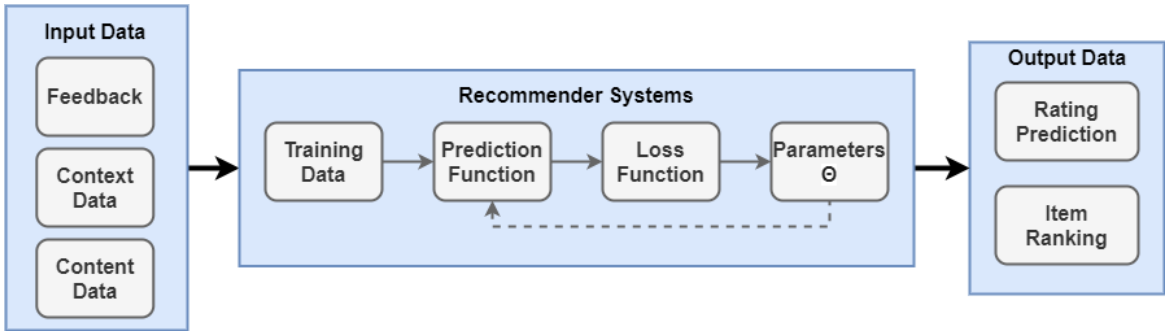

Figure 2.1: Representation of a recommender system.

2.1.2 Input data used by recommender systems

Nowadays, recommender systems have access to more and more data to understand users' tastes. The primary data available are the interactions between users and items, also known as feedback. There are two types of feedback:

- ***Explicit feedback:*** An explicit feedback is given by a user and usually takes the form of rating. It shows how much an item is appreciated by the user. The two most commonly forms are a score of 1 to 5 or a thumbs up/down mark. Explicit feedback provides detailed information on user preferences, but it is more difficult to collect compared to implicit one as users do not rate all the items with which they interact. Additionally, ratings can be biased due to the fact users do not rate items the same way (for example, some users can have an extreme rating method and will only award 5 points for what they like and 1 point for what they dislike);

- ***Implicit feedback:*** Implicit feedback (Oard et al., 1998) is retrieved automatically when the user watches, buys or clicks on an item. Implicit feedback is therefore very abundant because it does not require any effort from the user. It is also possible to transform explicit feedback into implicit feedback. The most common approach is to convert ratings into binary data. For instance, if we have a 1-to-5 score, we can convert

[1]Note that it is possible to have a recommender system based on non-parametric models such as neighborhood models. This type of models will be detailed later in this chapter.

all ratings above 3 to be positive feedback and other ratings to be negative/unobserved feedback. Another approach can be to convert all ratings as positive feedback.

Early recommender systems used explicit feedback (Koren et al., 2009, Ricci et al., 2011, Weimer et al., 2008). In recent years, the trend has reversed and more and more recommender systems use implicit feedback (Bayer et al., 2017, Hu et al., 2008, Kang and McAuley, 2018, Pan and Chen, 2013, Rendle et al., 2010, Sun et al., 2019, Tang and Wang, 2018, Yuan et al., 2020). It is important to note that the feedback, whether explicit or implicit, is ordered, which allows to exploit its sequential dynamics. In this thesis, we are interested in recommender system that use implicit feedback only. The primary characteristics of implicit feedback are:

- Only positive feedback is available ;
- Negative and missing feedback are mixed together ;
- Datasets with implicit feedback are very sparse (i.e. there is much more missing/negative feedback than positive one). In our experiments, the datasets used have a sparsity close to 99%. This has the advantage of reducing the complexity of some approaches, but also the disadvantage of providing less reliable recommendations. In all cases, this characteristic of datasets must be taken into account by recommendation methods.

Other sources of data are also used by recommender systems:

- Item features (Name, Category, Author, etc...) ;
- User features (Age, Gender, Jobs, etc...) ;
- Context of the recommendation (Location, Time, Social friends, weather, etc...).

To clarify the rest of the document, content information refers to item characteristics and context information refers to user features as well as the context of the recommendation. All of this information (Content/Context) can be difficult to collect. In addition, there can be a large disparity in the available information (i.e. some users have filled out their profiles completely while others have only partially completed their profile).

The data used by the recommendation system defines the problem addressed by it. We thus obtain a typology of the methods that we present in the following subsection.

2.1.3 Types of recommender systems

The commonly accepted typology of recommendation systems consists of grouping them into five classes: (1) Collaborative Filtering models, (2) Content/Context-based recommender systems (3) Knowledge-based recommender systems, (4) Demographic recommender systems and (5) Hybrid-based recommender systems. Their main differences are the way recommendations are generated and the way input data is processed. In the following, we briefly review knowledge-based recommender systems and demographic recommendation systems as they are not widely used due to their performance or their restricted use cases. We develop more widely the other three which gave rise to numerous contributions.

Knowledge-based recommender systems (Burke D., 2000) are used to recommend infrequently purchased items such as cars, travel, financial services, etc. In these cases, it is difficult to know the user's tastes and their preferences are likely to change between two interactions. For this type of system, it is the users who will explicitly give information about the item they are looking for. Then, the system will return the items according to the needs expressed by the user.

Demographic recommender systems (Krulwich, 1997, Pazzani, 1999) rely on user demographic information, such as age and country, to assign a class. Then, from a rule-based system, the recommendation is made based on the user's class. These methods are not very effective because using demographic information alone does not fully model the user tastes. Nevertheless, this information can be useful for creating hybrid models.

Collaborative Filtering models are one of the most common types of models used in recommendations. The idea of these models is to provide a recommendation based exclusively on user-item interactions (explicit or implicit feedback). Collaborative filtering models assume that if user $u1$ has the same tastes as other users for example $u2$ and $u3$, then user $u1$ is inclined to like items liked by $u2$ and $u3$. Historically, collaborative filtering models have been separated into two groups: Memory-based approaches and model-based approaches.

Memory-based methods, also known as neighborhood-based methods, were the first methods of collaborative filtering. They are generally separated into two types (Desrosiers and Karypis, 2011):

1. User-based Collaborative Filtering (User-based CF) (Herlocker et al., 1999) methods assume that similar users like the same items. User-based CF looks for users similar to a target user u using a similarity function based on an explicit or implicit feedback. Similar users or "neighbors" are users who have a similar rating pattern.

2. Item-based Collaborative Filtering (Item-based CF) (Sarwar et al., 2001) methods assume that users like items similar to items they have already liked. Item-based CF looks for items similar to a target item i using a similarity function based on explicit or implicit feedback. Two items are similar if they have been rated the same way by several users.

In Memory-based methods, the most commonly used similarity functions are cosine similarity, Pearson correlation and Jaccard coefficient. Once the nearest neighbors of a given item i have been identified (top-N neighbors), the score of i is calculated as the weighted sum of the similarity scores based on the neighbor ratings (in case of implicit feedback, the ratings can be replaced by the value 1). The great advantage of neighborhood-based methods is their simplicity (they are easy to understand and explain) and they can perform well as shown in (Kamehkhosh et al., 2017, Verstrepen and Goethals, 2014).

Model-based Collaborative Filtering is based on Machine Learning, Data mining and deep learning models to automatically learn user preferences. Model-based Collaborative Filtering has received a lot of attention over the past decades and many approaches have been proposed, such as Bayesian methods (Miyahara and Pazzani, 2000), restricted Boltzmann machines (Salakhutdinov et al., 2007), factorization machines (Rendle, 2012), matrix factorization (Koren and Bell, 2015, Koren et al., 2009), metric embedding model (Chen et al.,

2012a) and neural network(He et al., 2017b). Among these methods matrix factorization, metric embedding models, Neural networks are the most widely used and efficient methods.

The major problem with Collaborative Filtering methods is that it suffers from the cold-start problem. Item cold-start issue arises when an item does not have enough user interactions to be recommended, while user cold-start problem arises when a user has a number of feedback that is not enough to be eligible for a recommendation. We provide a detailed description of collaborative filtering models in Sections 2.2 and 2.3.

Content/Context-based recommender systems use item information (content information) such as name, author, category, keywords to make recommendations (Lops et al., 2011, Pazzani and Billsus, 2007). Unlike collaborative filtering models, content-based models only use user feedback and do not use feedback from other users to make a recommendation. This makes the model to focus on the user's interests and to recommend unknown items based on the categories of items the user has liked in the past. For example, if a user gives a positive feedback for a science fiction movie, the recommendation model is likely to recommend another science fiction movie because it has learnt that the user likes this category of movie. Unlike collaborative filtering methods, content-based models do not suffer from the cold-start item problem (i.e. the ability of a recommender system to recommend a new item while it has no interactions with users) because some users have probably already interacted with items similar to the new item. However, content-based models have some disadvantages:

- The provided recommendations lack diversity as they are always part of the same area of interest that the user likes (the model is centered on the user tastes only);
- In order to make relevant recommendations, the user must give sufficient feedback. These types of methods are strongly impacted by the user's cold-start problem (i.e. the ability of a recommender system to be able to recommend items to a new user).

On the other hand, context-based recommender systems use information related to user profiles as well as location, time, weather, or any other contextual information. For example, there are many works that exploit time information from previous actions to provide recommendations. This kind of methods is know as Time-Aware Recommender Systems (TARS) (Campos et al., 2014). Most TARS methods also use collaborative filtering techniques making them hybrid models.

Hybrid-based recommender systems (Burke, 2002, 2007) combine several types of models to improve performance and overcome weaknesses related to the type of recommender systems used. One of the most common combinations is to mix a content-based method with a collaborative filtering method. This reduces the cold start problem of collaborative filtering methods. For instance, CHAMELEON (de Souza Pereira Moreira, 2018), a deep Learning hybrid-based recommender system, uses the interaction between user and item, item content and user context to address the specific challenges of news recommendation. This architecture allows CHAMELEON to reduce the problem of cold-start. However, as for content/context-based recommender systems, hybrid-based recommender systems use content and context input data which can be difficult to collect.

2.1.4 Recommender systems problem formulations

The recommendation problem consists to learn the preferences and tastes of a user. It can therefore be seen as a learning problem from examples. From this angle, recommender systems try to predict the rate of an item. A more robust approach consists to no longer predict the rate of an item, but its relative ranking with respect to other items. We detail these two approaches in the following.

Prediction of the rating: It is common to define the recommendation problem as predicting a user's rating on an item. For example, Netflix launched a competition in 2009 to predict user ratings for movies (Koren, 2009b). The dataset was made of a $m \times n$ matrix, with m the number of users and n the number of items, which contained some user ratings on the items (i.e. explicit feedback), but also many missing values. The goal of the challenge was to predict the missing value via a recommender model. This can be seen as a matrix completion problem, since we have an incomplete matrix that is filled in by a learning algorithm (Aggarwal, 2016). However, this framework has two main drawbacks: (1) The rating predicted by a model would be in the best case the rating of how a user likes an item when interacting with it and not how he likes the item before the interaction (Yuan, 2018); (2) By optimizing the accuracy of the ratings, the model is trained to have good accuracy in rating items that will not be recommended, as most of them will not appreciated by the user (Guillou, 2016). This explains why, in most application cases, this problem formulation is not used and many approaches focus on optimizing a list of items the user wants to interact with, as explained below.

Item ranking: In real-world scenario, recommender systems return a list of N items to users. This list of items is made up of the Top-N items that have the best chance of being chosen or liked by the user. Hence, it is common to consider the recommendation problem as an item ranking problem, called Top-N recommendation problem. In this case, the prediction model builds the list of the Top-N items by taking the ones having the highest prediction value. More generally, models that tackle the item ranking problem are part of the Learning-To-Rank (LtR) models (Liu, 2011). Notice that, for this formulation, models can use implicit or explicit feedback as well as content and context features.

Let us now focus on Collaborative Filtering models – that is, models that only use implicit or explicit feedback – as this is the type of recommender systems we consider in this thesis. The objective is to compute a predicted score $\widehat{p}_{u,i}$ for the interaction between user u and item i. This score is obtained by the prediction function f with parameters Θ: $\widehat{p}_{u,i} = f(u, i \mid \Theta)$. There are three main approaches to determine the model parameters Θ by minimizing a loss function:

- ***Pointwise approaches*** consist to minimize a loss function between the predicted score and the target value for an entry. The two most commonly used loss functions are the weighted square loss function (Pan et al., 2008) and the logistic loss (Mikolov et al., 2013). However, as explained before, there are only few observed feedbacks. Two strategies are then possible: We can assume that all unobserved feedback is either a negative feedback – also known as AMAN (all missing as negative) – or an unknown feedback – also know as AMAU (all missing as unknown). Nevertheless, these two

strategies are a bit too extreme and impact the accuracy of recommendations. Another approach consists to balance the proportion of unobserved feedback treated as negative feedback (Pan et al., 2008). The model parameters are then adjusted using a Stochastic Gradient Descent (SGD) and negative sampling (i.e. uniformly sample a portion of the negative feedback that is use to evaluate the loss functions) for efficient optimization. However, pointwise approaches do not take into account the interdependence between items. As a result, the final position of many items in the list is ignored by the loss function and the model estimates are somewhat imprecise;

- ***Pairwise approaches*** rely on the assumption that a user prefers an observed feedback over an unobserved one. As for the pointwise approaches, the pairwises approaches aim to predict the interest of a user for all items. But it focus on the relative order of preferences between items. To that end, the loss function examines a pair of items and try to produce higher scores for observed feedback than unobserved ones: $i >_u j$ with i the observed feedback, also called positive item, and j the unobserved feedback, called negative item. The loss functions optimize the model directly on the item ranking unlike the pointwise functions that optimize an absolute value.

  One of the best-known functions is Bayesian Personalized Ranking (BPR) Rendle et al. (2009), which is formalizes as maximizing the posterior probability of the model parameters given the order relation between the positive item i and the negative item j. Similar to pointwise approaches, pairwise approaches must process a large number of items, making it difficult to achieve a full gradient. Moreover, using a full gradient descent for loss optimization leads to slow convergence. To cope with this problem, pairwise approaches also apply negative sampling, where a user u, a positive item i and a negative item j are uniformly sampled to form a training triple (u, i, j). This reduces the computational cost of the loss optimization. It is common to optimize the function using a Stochastic Gradient Descent (SGD), but it is also possible to use other optimization algorithms such as Adam optimizer (Kingma and Ba, 2014);

- ***Listwise approaches*** (Shi et al., 2012) use all the observed feedback and try to directly optimize ranking functions that are generally non-differential or non-continuous. For the moment this type of approach brings limited advantages compared to pairwise approaches.

In this thesis, we focus on pairwise approaches that directly optimizes the ranking of items. Indeed, this is a more realistic approach to tackle the problem of sequential recommendation. It is important to note that the main difference between pointwise, pairwise and listwise approaches comes from the loss function. Therefore, using the same prediction function can be optimized by pointwise, pairwise, or listwise approaches.

2.1.5 The long-term and short-term dynamics

The work carried out in this thesis focuses on the study of sequential recommendation that aims to predict the next item that will interest a user based on the sequence of items that he previously purchased/consumed. More generally, sequential recommendation models are part of sequence-aware recommendation presented in (Quadrana et al., 2018).

To achieve the best possible recommendation, sequential models aim to capture two aspects in the recommendation: (1) the user preferences – that is to say the general tastes of the user – also called long-term dynamics, and (2) the sequential dynamics – that is to say the context of users' activities – also called short-term dynamics. To understand the importance of these two aspects, let us consider a dummy example to illustrate the importance of the long and short-term dynamics: If a user buys a laptop, a sequential recommendation model may recommend a black laptop backpack, which may be relevant. This recommendation is based on the sequential dynamics which allows the model to recommend a "laptop backpack". The "black" color of the backpack is derived from the user preferences known by the model. In this dummy example only implicit data is used by the model.

In the following of this chapter, methods that only model user preferences are called "General recommendation" methods (presented in Section 2.2) and methods that only model sequential dynamics and methods that model long-term and short-term dynamics will be called "Sequential recommendation" (presented in Section 2.3).

There are several variants to this problem which lie outside the scope of this thesis. For instance, *temporal recommendation* (Ding and Li, 2005, Koren, 2009a, Wu et al., 2017, Xiong et al., 2010) takes into account the timestamps of users' actions to make recommendations based on a specific time (e.g. recommending a glass of wine in the evening but not in the morning). Also, we exclude *content/context-aware* (Chen et al., 2012b, Huang et al., 2018, Pasricha and McAuley, 2018, Rendle, 2012, Sanchez and Bellogín, 2020, Zhang et al., 2016), as these approaches require contextual information (timestamps, item categories, user features, etc...) which limits their use to specific data. Hence, in the remaining of this Chapter, we limit ourselves to methods that only take as input users' sequences to make recommendation, without considering additional information. In our case the actions are ordered but we do not use the timespan as a feature of our model.

There are other approaches, called session-based models (Wang et al., 2019) that are part of sequence-aware recommendation (Quadrana et al., 2018). Session-based models give more weight to recent events by construction: The user sequences is divided in sessions. Therefore, they focus on shorter user sequences (i.e., the sessions) fostering short term dynamics. Considering sessions makes it possible to focus the recommendation on similarities with recent items. Some of these methods, are widely used and are presented in Section 2.3.

Note that as mentioned in (Quadrana et al., 2018), sequence-aware recommender systems can be considered as special forms of context-aware recommender systems because they try to understand via the short-term dynamics the 'context' of users' activities. However to simplify the reading of this thesis, we have delimited context-aware recommender systems as models that use explicit contextual data such as time, location, or similar features.

2.2 General recommendation

A recommender system can be seen as two separate modules (Yuan, 2018): (1) a prediction function and (2) a loss function. In the previous section we presented the three main classes of loss functions (i.e. Pointwise, Pairwise, Listwise). In sections 2.2 and 2.3, we present in detail the different prediction functions that can be used for a collaborative recommender system.

Seminal recommendation methods aim at identifying user preferences using Collaborative Filtering techniques while ignoring the sequential dynamics. They are presented in this section, starting by matrix factorization – which is one of the most known and used methods –, followed by factorization of machines, neighborhood models – which despite their simplicity achieve good results –, and deep learning models.

2.2.1 Matrix Factorization

Matrix Factorization (MF) (Koren et al., 2009) is one of the most widely used methods in recommendation systems. It has gained a lot of popularity through the Netflix challenge (Koren, 2009b). MF-based models are part of the latent factor model and their purpose is to model the interactions between users and items. More formally, implicit feedback is transformed into a matrix, $\mathbf{D}_{\text{user}\times\text{item}} = (d_{u,i})$ where $d_{u,i} = 1$ if and only if u gave a feedback on i ($i \in I_{s_u}$). Matrix Factorization consists in decomposing this matrix into a product of two k-ranked matrices:

$$\mathbf{D}_{\text{user}\times\text{item}} \approx R \times Q$$

The prediction $\widehat{p}_{u,i}$, that user u chooses item i, is then estimated by the inner product of the corresponding vectors:

$$\widehat{p}_{u,i} \propto \langle \mathbf{R}_u, \mathbf{Q}_i \rangle = \sum_{x=1}^{k} \mathbf{R}_{u,x}\mathbf{Q}_{i,x}, \tag{2.1}$$

where $\mathbf{R}_u$ and $\mathbf{Q}_i$ are the latent vectors of dimension k associated to user u and item i. Note that the greater the value of $\widehat{p}_{u,i}$ the more the recommendation of item i is relevant for u. To fully understand what $\mathbf{R}_u$ and $\mathbf{Q}_i$ model, let us consider a concrete example: let I represent a set of movies. Then we can imagine that each dimension k models a genre of movies and each vector value of a latent vector of $\mathbf{Q}$ is the degree of affinity between a movies and a genre. On the other hand, each vector value of a latent vector of $\mathbf{R}$ models the affinity that a user has to the corresponding movie genre. It is therefore from the inner product of the affiliation of an item to a genre of movie ($\mathbf{Q}_i$) and the affinity of a user for a movie genre ($\mathbf{R}_u$), that we can estimate whether item i is relevant for user u. Note that this is an example and in practice it is very difficult to know what a dimension/feature of a latent vector corresponds to.

Early MF models generally used explicit feedback and to overcome the problem of rating bias (users with extreme rating behavior, popular items tend to be rated higher), bias terms can be added. This idea can also be apply to MF using implicit data:

$$\widehat{p}_{u,i} \propto \beta_i + \beta_u + \langle \mathbf{R}_u, \mathbf{Q}_i \rangle, \tag{2.2}$$

where β_i, β_u are biases respectively associated to item i and user u. In case of explicit feedback the average of ratings can be add as bias term.

SVD++ (Koren, 2008) enhances traditional matrix factorization models using explicit data by adding user information with implicit feedback. According to the authors of (Koren and Bell, 2015), even if independent implicit feedback is missing, this is not a problem for the method as it can capture a signal from rated items, regardless of their rating value. To that

end, SVD++ add a new set of latent vector $\mathbf{P}$ that correspond to the items of I. $\mathbf{P}$ is used to model users, based on the items they already have rated. SVD++ is defined as follows:

$$\widehat{p}_{u,i} \propto \beta_i + \beta_u + \beta + \langle \mathbf{R}_u + |I_{s_u}|^{-\frac{1}{2}} \sum_{j \in I_{s_u}} \mathbf{P}_j, \mathbf{Q}_i \rangle, \tag{2.3}$$

where I_{s_u} is the set of rated items by u and β is the average rating of all observed ratings. Unlike traditional matrix factorization models, each user is modeled by two components:

- $\mathbf{R}_u$ is learned from explicit feedback (it can be learned from implicit feedback);
- $|I_{s_u}|^{-\frac{1}{2}} \sum_{j \in I_{s_u}} \mathbf{P}_j$ is learned from implicit feedback.

It is also possible to add several sources of implicit feedback using several sets of latent vectors corresponding to items. For instance, the first kind of implicit feedback could come from clicks without purchase on items, and the second kind of implicit feedback could come from purchased items. This can be modeled as follows:

$$\widehat{p}_{u,i} \propto \beta_i + \beta_u + \beta + \langle \mathbf{R}_u + |N^1(u)|^{-\frac{1}{2}} \sum_{j \in N^1(u)} \mathbf{P}_j^{(1)} + |N^2(u)|^{-\frac{1}{2}} \sum_{j \in N^2(u)} \mathbf{P}_j^{(2)}, \mathbf{Q}_i \rangle, \tag{2.4}$$

where $N^1(u)$ and $N^2(u)$ are respectively the set of clicked items and the set of bought items by u. $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ are the two new latent matrices of items.

However, as usually users give implicit feedback on few items of I, the matrix $\mathbf{D}$ is sparse and these approaches suffer from loss of precision when the numbers of users and items grow. In order to overcome these problems, other approaches such as FISM (Kabbur et al., 2013) – which is an improvement of SLIM (Ning and Karypis, 2011) – decompose an implicit item-to-item similarity matrix in two k-ranked matrices $\mathbf{P}$ and $\mathbf{Q}$:

$$\widehat{p}_{u,i} \propto \beta_i + \beta_u + \frac{1}{|I_{s_u} \setminus \{i\}|^{\alpha}} \sum_{j \in I_{s_u} \setminus \{i\}} \langle \mathbf{P}_j, \mathbf{Q}_i \rangle, \tag{2.5}$$

where β_i, β_u are biases respectively associated to item i and user u, $\frac{1}{|I_{s_u} \setminus \{i\}|}$ normalized large sets of items, and α is used to control the degree of agreement between items. Hence, the more i is similar to items already chosen by user u, the more likely i will be a good recommendation for u. Taking into account these transitive relations between items makes it possible to increase the quality of the recommendation.

2.2.2 Factorization Machines

Factorization machines (FMs) (Rendle, 2010, 2012) are generic factorization approaches that combine the advantages of Support Vector Machines (SVM) with factorization models. In contrast to SVM predictors, using factorized parameters FMs can deal with very sparse data. The architecture of FMs allow them to be versatile, that is to say they can be used for ranking tasks, regression, or classification depending on the loss function. According to (Rendle, 2010), Factorization Machines can mimic most of the factorization models like Matrix Factorization (Koren et al., 2009, Salakhutdinov and Mnih, 2008, Srebro et al., 2005), SDV++ (Koren, 2008), PITF (Rendle and Schmidt-Thieme, 2010) and FPMC (Rendle et al.,

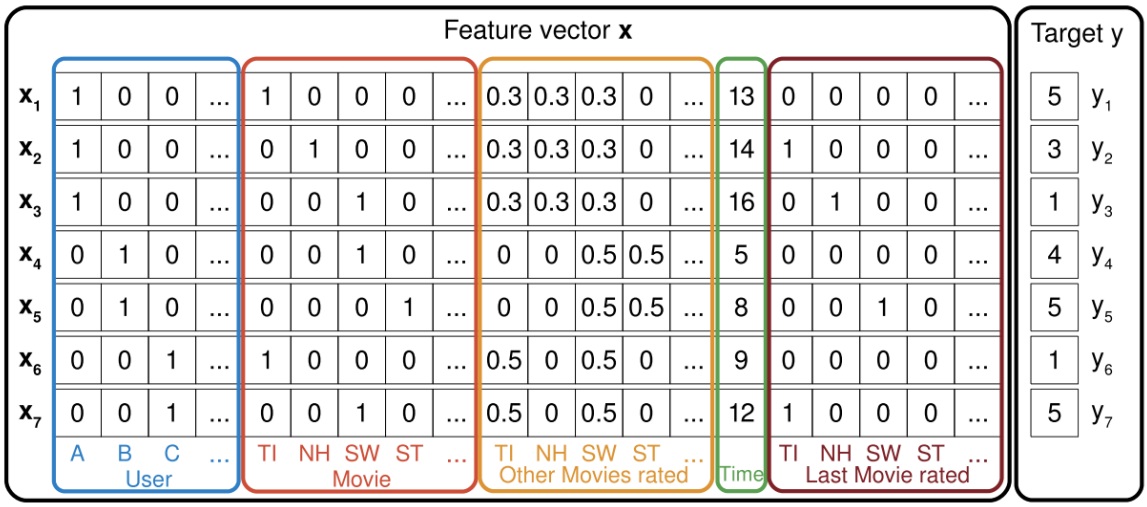

Figure 2.2: Representation of input data of FMs. This figure comes from (Rendle, 2010).

2010). As shown in Figure 2.2, all input variables are concatenated into a feature vector x. The blue and red columns represent variables for the active user and active item, respectively. The other columns are the real values of content and context data.

The model equation for a FM of degree $d = 2$ is the following:

$$\widehat{p}_{u,i} \propto w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{V}_i, \mathbf{V}_j \rangle x_i x_j, \tag{2.6}$$

where w_0 is the global bias ($w_0 \in \mathbb{R}$), w_i models the strength of the i-th variable ($w \in \mathbb{R}^n$) and $\langle \mathbf{V}_i, \mathbf{V}_j \rangle$ models the interaction between the two variables.

Recently, TransFM (Pasricha and McAuley, 2018) has adapted ideas of FMs into translation-based sequential recommendation [2]. However, to take full advantage of FMs or TransFMs, content and context information are needed. As shown in (Pasricha and McAuley, 2018), whit only implicit feedback (our scope), these methods barely outperforms Matrix Factorization techniques.

2.2.3 Neighborhood-based methods

One of the simplest Neighborhood-based recommendation methods is certainly the one used in (Hidasi et al., 2016a) called Item-KNN (IKNN). It only uses the last item of the sequence of user actions (s_u) to make the recommendation (i.e. the most similar items to the last item). In the case of implicit feedback, we have a matrix $\mathbf{D}_{\text{user} \times \text{item}} = (d_{u,i})$ with $d_{u,i} = 1 \Leftrightarrow i \in I_{s_u}$. The similarity between two items i and j, noted $sim(i, j)$, is usually determined by the cosine similarity measure. The length of the list of recommendation is control using the number k of neighbors. IKNN, as defined in (Hidasi et al., 2016a), can be extended as proposed by (Deshpande and Karypis, 2004) in order to consider all the items of the user feedback to

[2] We will give details about translation-based sequential recommendation in the next section when we introduce TransRec (He et al., 2017a).

influence the score of item i for user u:

$$\widehat{p}_{u,i} \propto \sum_{j \in I_{s_u}} sim(j,i)\mathbb{1}_j(i), \tag{2.7}$$

where $\mathbb{1}_j(i)$ is a function that return 1 if i is in the neighborhood of j and 0 otherwise.

Recently Session-based KNN (SKNN) has become popular in session-based recommendation because it improves GRU4REC (Jannach and Ludewig, 2017), a recurrent neural networks with Gated Recurrent Units for session recommendation. Unlike IKNN, which only uses the last item, here all the items of the current session[3] are used to find the k most similar sessions in the training data. First, given a session s, SKNN finds the k most similar sessions of s (N_s) using similarity measure such as the Jaccard index or the cosine similarity on binary vector. According to (Ludewig and Jannach, 2018), using $k = 500$ led to good performances on many datasets. Then, using the set of k most similar session N_s and the chosen similarity function $sim(s_1, s_2)$, the prediction score of item i for the current session s is defined as (Bonnin and Jannach, 2014):

$$\widehat{p}_{s,i} \propto \sum_{n \in N_s} sim(s,n)\mathbb{1}_n(i), \tag{2.8}$$

where $\mathbb{1}_n(i)$ is a function that return 1 if session n contains i and 0 otherwise.

Note that, in practice all the similarities are pre-computed and ordered during the training phase which allows to make quick recommendations. For more details on scalability consideration see (Ludewig and Jannach, 2018) which gives an implementation of Neighborhood-based recommendation methods.

Despite their simplicity, neighborhood-based methods (or memory-based methods) often perform equally or better than complex methods like deep neural networks as discussed in (Jannach and Ludewig, 2017, Ludewig and Jannach, 2018, Verstrepen and Goethals, 2014). Neighborhood-based methods are generally good baseline on many datasets.

2.2.4 Deep learning models

Recently, deep learning techniques have been introduced in many recommender systems. The advantage of neural networks compared to factorization models is that they are able to approximate continuous functions. The first model based on a neural networks architecture is certainly Restricted Boltzmann Machine (RBM) based recommender provided by Salakhutdinov et al. (Salakhutdinov et al., 2007). RBM targets rating prediction and its architecture is difficult to adapt to item ranking.

After RBM, another category of deep neural network called *Autoencoder* emerged with the work of (Sedhain et al., 2015). AutoRec uses user rating vectors or item rating vectors as input and try to reconstruct them in output Layers. Two variants are possible, (1) one based on item interactions called item-based AutoRec (I-AutoRec) and (2) one based on user actions called user-based AutoRec (U-AutoRec). I-AutoRec performs better than U-AutoRec due to the higher variance of user rating. However like RBM, I-AutoRec and U-AutoRec are designed for rating prediction using explicit feedback. A Collaborative Denoising AutoEncoder (CDAE) (Wu et al., 2016) has been proposed to tackle the problem of item

[3] In our case a user only have one session.

ranking using implicit feedback. As can be seen in the figure 2.3, CDAE uses one hidden layer as standard Denoising Autoencoder. The input layer has $|I| + 1$ nodes where the last node is specific for the user and the others nodes correspond to items. If the user has interacted with an item, the input node is equal to 1, otherwise it is equal to 0. The output layer has $|I|$ nodes for the reconstruction of the input data. The prediction value of item i for user u is defined by:

$$\widehat{p}_{u,i} \propto f(\mathbf{W}_i^{'T} h(\mathbf{W}^T \widetilde{y_u} + \mathbf{V}_u + \beta) + \beta_i), \tag{2.9}$$

where $\mathbf{W}^{'} \in \mathbb{R}^{|I| \times |k|}$ and $\mathbf{W} \in \mathbb{R}^{|I| \times |k|}$ are weight matrices, $\mathbf{V}_u \in \mathbb{R}^{|k|}$ are weight vectors for the user input node and β and β_i are bias vectors. $f(\blacksquare)$ and $g(\blacksquare)$ are mapping functions such as identity or sigmoid functions.

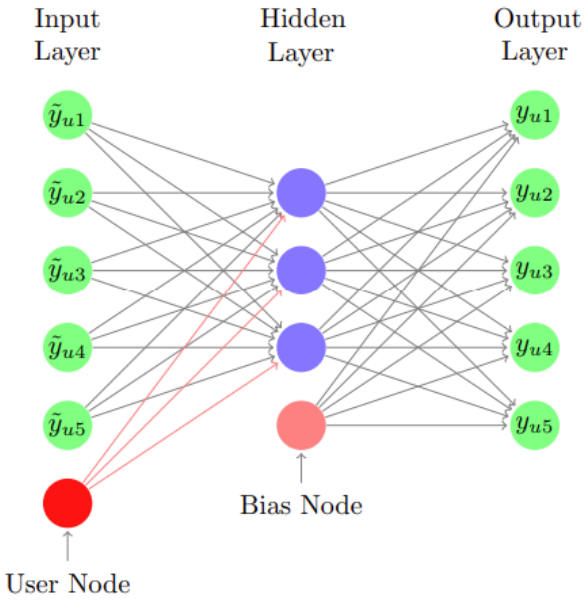

Figure 2.3: Illustration of CDAE for one user. This figure comes from (Wu et al., 2016).

Another line of work uses Multilayer Perceptron (MLP) (Hornik et al., 1989) in order to replace the matrix factorization. For instance, the Neural Collaborative Filtering (NCF) model called NeuMF (He et al., 2017b) provides a combination of a generalized matrix factorization (GMF) – which is a generalized and extended matrix factorization model – with a Multilayer Perceptron (MLP). According to the authors of (He et al., 2017b), to obtain better performance it is preferable for GMF and MLP to learn separate embeddings and concatenate their last hidden layer (see Figure 2.4) as follow :

$$\widehat{p}_{u,i} \propto \sigma(h^T \begin{bmatrix} \phi^{MLP} \\ \phi^{GMF} \end{bmatrix}), \tag{2.10}$$

where h^T is an activation function and ϕ^{GMF} and ϕ^{MLP} are respectively the GMF and the MLP models.

For more information, on RBM, Autoencoder and Neural Collaborative Filtering see (Zhang et al., 2019). Note that, we focus here on neural network models that do not take

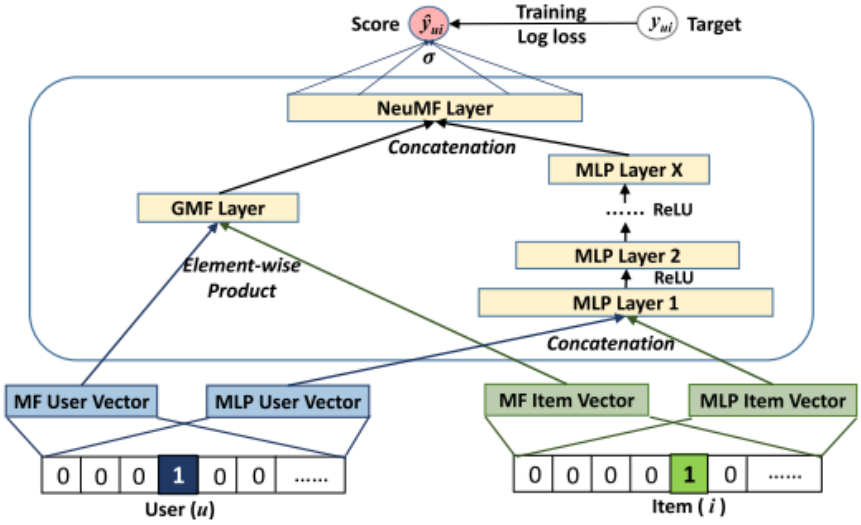

Figure 2.4: Architecture of NeuMF. This figure comes from (He et al., 2017b).

into account the sequential aspect of the data, as the other ones are presented in the next section.

2.3 Sequential recommendation

Sequential dynamics have been considered more recently as an important aspect to be taken into account by recommender systems. Represented by the order of the items in the user sequences, it can indeed be an important source of information for making relevant recommendations. In this section, we present the first models that took into account the sequential dynamics while omit the long-term dynamics, followed by sequence-aware extensions of Neighborhood-based methods. We then explain models that unify user preferences and sequential dynamics. Finally, we describe deep learning models.

2.3.1 Sequential recommendation based on Markov Chains

In order to take into account the sequential dynamics in recommender systems, many models use first order Markov Chains (Norris, 1998). It consists in using the probability $p(i \mid j)$ of predicting item i given the last item in the user sequence is j. From this, a simple recommender system can directly be derived based on first order Markov Chains (as presented in Ludewig and Jannach (2018)). Basically, this simple model is equivalent to counting the number of times item i is found after item j in all user sequences. More formally, the prediction score of item i given the last item j in a user sequence can be defined by the following maximum-likelihood estimation (MLE):

$$\widehat{p}_{i|j} \propto \frac{\sum_{u \in U} \sum_{x=1}^{|s_u|-1} \mathbb{1}_{EQ}(j, {s_u}^x) \mathbb{1}_{EQ}(i, {s_u}^{x+1})}{\sum_{u \in U} \sum_{x=1}^{|s_u|-1} \mathbb{1}_{EQ}(j, {s_u}^x)}, \tag{2.11}$$

where ${s_u}^x$ is the item at position x in sequence of user u and the function $\mathbb{1}_{EQ}(a, b)$ return 1 if a is equal to b and 0 otherwise. The numerator counts the number of times item i is found after item j in all user sequences, and the denominator counts the number of times item j appears in the sequences. Shani et al. (2005) propose to enhance the maximum-likelihood estimates (MLE) of the Markov chains transition using several heuristics (e.g. clustering and skipping). However, MLE easily fails due to the sparsity of feedback, and also because important properties, such as minimal variance and Gaussian distribution, are not met.

To overcome the problem of sparsity of the data, Rendle et al. (2010) propose the methtod Factorized Markov Chains (FMC) that decomposes the transition matrix $(p(i \mid j))_{ij}$ in a product of two k-ranked matrices $\mathbf{M}$ and $\mathbf{N}$. $\mathbf{M}_j$ and $\mathbf{N}_i$ represent the latent or embedding vectors of item j and i so that the probability of having item i after j is estimated by the inner product:

$$\widehat{p}_{i|j} \propto \langle \mathbf{M}_j, \mathbf{N}_i \rangle \tag{2.12}$$

It is also possible to consider high order Markov Chains with the same principle. However, FMC models use inner products which do not satisfy the triangle inequality property. For instance, if the transitions $A \rightarrow B$ and $B \rightarrow C$ have high probability, we can expect that the transition $A \rightarrow C$ is also of high probability. But FMC do not guarantee this because of the independent assumption between $\mathbf{M}_B$ and $\mathbf{N}_B$.

To guarantee the triangle inequality condition, Chen et al. (2012a) propose a Logistic Markov Embedding (LME) that maps each item in a latent Euclidean space. The probability $p(i \mid j)$ of predicting i given the last item j in the user sequence is estimated by the Euclidean distance between i and j:

$$\widehat{p}_{i|j} \propto -||P_j - P_i||_2^2, \tag{2.13}$$

with $||P_j - P_i||_2^2 = \sum_{x=1}^{k} (\mathbf{P}_{xj} - \mathbf{P}_{xi})^2$, and $\mathbf{P}_j$ and $\mathbf{P}_i$ are the latent vectors associated to items j and i of dimension k. Note that, we take the opposite of the Euclidean distance because it makes possible to maximize the prediction score $\widehat{p}_{i|j}$. In this model, each item is represented by a single vector, unlike in FMC model that uses two independent vectors for each item. Furthermore, the triangle inequality is now satisfied.

In this subsection, we presented the first models taking into account the sequential dynamics. Compared to methods that model only the long-term dynamics (user preferences), sequential recommendation models only based on Markov Chains are a slightly less efficient. It is by unifying the user preferences and the sequential dynamics that the best sequential recommendations are obtained.

2.3.2 Sequence-aware extensions for neighborhood-based methods

In the previous section, we presented two neighborhood-based methods (SKNN and IKNN) that do not take into account the order of the items when determining the nearest neighbors. As mentioned several times above, it is important to consider the sequential aspect of feedback. For this purpose, Ludewig and Jannach (2018) provide three extensions of SKNN to take into account the sequential dynamics:

1. ***Vector Multiplication Session-Based KNN (V-SKNN):*** The idea of this variant is to give more importance to recent items of a session when calculating similarity.

Instead of encoding the current session as a binary vector, V-SKNN uses a linear decay function to assigned a real value depending on the item's position in the user sequence. The more recent an item, the closer its value will be to 1. The prediction function is the same as SKNN, the modification is the similarity function;

2. ***Sequential Session-Based KNN (S-SKNN):*** It is the same idea as V-SKNN which is to give more importance to recent items in a session. However, this time the weight is applied directly to the prediction function as follows:

$$\widehat{p}_{s,i} \propto \sum_{n \in N_s} sim(s,n) w_n(s) \mathbb{1}_n(i), \tag{2.14}$$

where $\mathbb{1}_n(i)$ is the function that returns 1 if the session n contains i, and 0 otherwise. The function $\mathbb{1}_n(i)$ is weighted by $w_n(s) = \frac{x}{|s|}$, where $|s|$ is the number of items in the current session s and x is the position of the most recent item of s that also appears in the neighboring session n. For instance, if $|s| = 5$ and the third most recent item of s ($x = 3$) is the most recent item in the neighboring session n, the weight would be $w_n(s) = \frac{3}{|5|}$;

3. ***Sequential Filter Multiplication Session-Based KNN (SF-SKNN):*** This one also modify the prediction function but in a more restrictive way. The idea is to be able to recommend an item only if the sequence of items $\langle s^{|s|}, i \rangle$ (i.e. last item of the current session s followed by the candidate item exists at least once in all user sequences). The prediction function is defined as follows :

$$\widehat{p}_{s,i} \propto \sum_{n \in N_s} sim(s,n) \mathbb{1}_n(\langle s^{|s|}, i \rangle), \tag{2.15}$$

where $\mathbb{1}_n(\langle s^{|s|}, i \rangle)$ returns 1 if it exists a past session that contains the sequence $\langle s^{|s|}, i \rangle$ and 0 otherwise. Note that the current session s still need to contains i to returns 1.

2.3.3 Unifying user preferences and sequential dynamics

Current recommendation methods accommodate user preferences and sequential dynamics as it has been observed that it increases their performances. Factorized Personalized Markov Chains (FPMC) (Rendle et al., 2010) is one of the first method that uses both Matrix Factorization and first-order factorized Markov Chains in order to unify the long and short-term dynamics. The probability that user u chooses item i just after having taken item j is estimated by the sum of two inner products:

$$\widehat{p}_{u,i|j} \propto \langle \mathbf{R}_u, \mathbf{Q}_i \rangle + \langle \mathbf{M}_j, \mathbf{N}_i \rangle, \tag{2.16}$$

with $\mathbf{R}_u$, $\mathbf{Q}_i$ $\mathbf{M}_i$ and $\mathbf{M}_j$ the latent vectors associated respectively to user u, item i and j.

FPMC has been extended by Wang et al. (2015) with a Hierarchical Representation Model (HRM) that uses aggregation operations such as max or average pooling to model more complex interactions :

$$\widehat{p}_{u,i|j} \propto \langle aggregation(\mathbf{R}_u, \mathbf{Q}_j), \mathbf{Q}_i \rangle \tag{2.17}$$

More recently, FactOrized Sequential prediction with item SImilarity modeLs (Fossil) (He and McAuley, 2016) proposed the association of methods based on the similarity between items, like FISM (Kabbur et al., 2013), with Markov Chains of order L. Given the most L recent items, the probability that user u chooses item i as the next item is defined as follows:

$$\widehat{p}_{u,i|s_u{}^{[t-L,t[}} \propto \beta_i + \langle \Big(\frac{1}{|I_{s_u} \backslash \{i\}|^{\alpha}} \sum_{j \in I_{s_u} \backslash \{i\}} \mathbf{M}_j \Big) + \Big(\sum_{x=1}^{L} (\eta_k + \eta_k^u) \mathbf{M}_{s_u{}^{t-x}} \Big), \mathbf{N}_i \rangle \tag{2.18}$$

The first part which is the weighted sum of interacted items by the user corresponds to user preferences and the second part which is the weighted sum of the last L items corresponds to the sequential dynamics. t is the position (time step) of the last item in the user's sequence, β_i the bias term for the item i, η_k is a global weight shared by all the users, and η_k^u is a personalized weighting factor that controls the relative importance of the long and short-term dynamics of user u. Fossil outperforms FPMC especially by using L order Markov Chain and similarity-based methods like FISM to model user preferences. However, like FMC, Fossil, HRM and FPMC do not guarantee the triangle inequality.

On top of that, Personalized Ranking Metric Embedding (PRME) (Feng et al., 2015) enhances FPMC and Fossil by replacing the inner products with Euclidean distances. As argued in (Chen et al., 2012a, Feng et al., 2015), metric embedding model brings better generalization ability than Matrix Factorization to represent Markov chains because of the triangle inequality assumption. The probability that user u takes the item i after item j is estimated by the sum of the following Euclidean distances:

$$\widehat{p}_{u,i|j} \propto - \left(\alpha \cdot ||\mathbf{R}_u - \mathbf{Q}_i||_2^2 + (1-\alpha) \cdot ||\mathbf{M}_j - \mathbf{M}_i||_2^2 \right), \tag{2.19}$$

with α a weight that controls the long-term and short-term dynamics. The major drawback of PRME is that the user preferences and sequential dynamics are two separated components which can damage the performance because these two components are highly correlated.

Another recent work, Translation-based Recommendation (TransRec) (He et al., 2017a) proposed a novel metric embedding model that unifies user preferences and sequential dynamics into a "transition space". To achieve this, items are embedded as points P_i in a latent transition space and users are modeled as translation vectors T_u in the same space:

$$\widehat{p}_{u,i|j} \propto \beta_i - d(P_j + T_u, P_i), \tag{2.20}$$

with $d()$ a distance (L_1 or squared L_2), T_u a translation vector representing u, and P_i, P_j points in the transition space related to items i and j. The sequential dynamics is captured with first order Markov chains.

However, all of these models suffers from a lack of personalization on the short term part due to the fact that Markov chains have a fixed length regardless of the users and their considered items.

2.3.4 Deep learning models

Deep learning models learn multiple levels of representations of data using multiple layers. These methods have improved the state-of-the-art in many domains such as speech recognition, visual recognition, natural language processing and images processing (LeCun et al.,

2015). Recently, a growing research effort has been dedicated to the investigation of the use of deep learning models for sequential recommendation (Zhang et al., 2019). Here we present the most common Deep learning models used for sequential recommendation.

Recurrent Neural Network (RNN) (Goodfellow et al., 2016) are widely used to model sequential data as for speech recognition (Graves et al., 2013) or machine translation (Sutskever et al., 2014). In this context, Recurrent Neural Networks (RNN) have been widely used for session-based recommendation where users are not identified and their long-term dynamics is usually missing. Such networks use Long short-term Memory (LSTM) or Gated Recurrent Units (GRU) and have shown good results to model user sequential Dynamics (Devooght and Bersini, 2017, Hidasi et al., 2016a, Jannach and Ludewig, 2017). Hidasi et al. (2016a) proposed GRU4Rec which uses Gated Recurrent Unit (GRU) (Cho et al., 2014). GRU4Rec is tailored for session-based recommendation. The input of the network is the actual state of the current session, that is to say a binary vector of size $|I|$, the number of items, where only the active items (items in the current session) are equal to one, the other items being equal to zero. The output of the network gives for each item the probability of being the next item. Figure 2.5 shows the whole architecture of GRU4Rec. To accelerate the training phase, the authors propose a session-parallel mini-batches algorithm and a sampling method for output. GRU4Rec has been proposed with two pairwise loss function: Bayesian Personalized Ranking (Rendle et al., 2009) and TOP1.

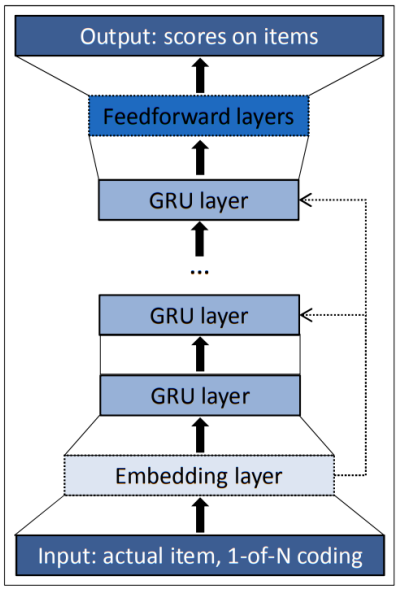

Figure 2.5: Illustration of the general architecture of GRU4Rec. This figure comes from (Hidasi et al., 2016a).

On top of GRU4Rec, Tan et al. (2016) propose three techniques to improve model performance : (1) Data augmentation with sequence preprocessing and dropout regularization, (2) Model pre-training tailored for temporal shifts in the data distribution and (3) Distillation with privileged information to learn on small datasets. Quadrana et al. (2017) propose a Hierarchical RNN (HRNN) which add GRU layers to model user information across several sessions. This allows HRNN to track the long-term dynamics more efficiently than GRU4Rec.

There are others RNN models (Beutel et al., 2018, Hidasi et al., 2016b, Smirnova and Vasile, 2017) that use side information, which means that they are outside the scope of this thesis.

Another line of work has investigated the use of Convolutional Neural Network (CNN) for recommendation. These methods use convolution layers and pooling operations. Traditionally, CNN are used to capture local features for image recognition (Karpathy et al., 2014) or for natural language processing (Kim, 2014). Convolutional Sequence Embedding Recommendation Model (CASER) (Tang and Wang, 2018) is the first CNN-based method that captures both user preferences and user sequential dynamics. CASER embeds L previously considered items as an $L \times d$ matrix $\mathbf{E}$ (d is the number of latent dimensions). This embedding matrix $\mathbf{E}$ is then processed as an "image". Using various convolution filters, CASER extracts sequential patterns as local features. Note that the "image" $\mathbf{E}$ is learnt simultaneously with all filters. CASER is made of 3 components :

1. ***Embedding Look-up:*** Given a user u and a time step t, the embedding look-up retrieves the L previous items and stack them to create the matrix $\mathbf{E}$ ($\mathbf{E} \in \mathbb{R}^{L \times d}$);

2. ***Convolutional Layers:*** The convolutional layers have two different filters: Horizontal filters capture union-level patterns (i.e. several ordered items that jointly influence the target item) and vertical filters that capture point-level sequential patterns (i.e. several previous items that influence individually the target item);

3. ***Fully-connected Layers:*** This component concatenates the horizontal and vertical convolutional layers in a fully-connected neural network layer to produce the convolutional sequence embedding. User preferences are captured using a user embedding vector which is concatenated in a final fully-connected neural network layer with the convolutional sequence embedding. The output of these layers gives for each item the probability for user u at time step t to interact with it.

Recently, Yuan et al. (2019) propose a 1D CNN model called NextItNet that uses 1D dilated convolution (Yu and Koltun, 2016) rather than standard 2D convolution like CASER. According to the authors, NextItNet outperforms CASER and GRU4Rec.

Attention-based models have shown their effectiveness to model sequential data in the case of machine translation (Bahdanau et al., 2015). The idea of the attention mechanism is that each input data does not influence each output of the model in the same way. One of the advantages of attention-based models is that they are more interpretable than other deep leaning architectures thanks to the attention weights. Some models have integrated attention mechanisms into recommender models (Chen et al., 2017, Li et al., 2017, Xiao et al., 2017). However, attention mechanisms were only an additional component (e.g. RNN+attention). Inspired by a purely attention-based sequential model for machine translation tasks named *Transformer* (Vaswani et al., 2017), Self-Attentive Sequential Recommendation (SASRec) (Kang and McAuley, 2018) is a new sequential recommendation model based on self-attention (Vaswani et al., 2017) that outperforms many advanced sequential models on sparse and dense datasets. SASRec is composed of 3 components :

1. ***Embedding Layer:*** From the sum of the item embedding matrix and a positional embedding matrix, the embedding layer creates an input embedding matrix $\mathbf{E} \in \mathbb{R}^{n \times d}$ with n the maximum sequence length and d the number of latent dimensions ;

2. ***Self-Attention Block:*** The input matrix **E** is converted into three matrices by linear projection and feed the attention layer which is defined as a scaled dot-production attention (Vaswani et al., 2017). After this, a point-wise two-layer feed-forward network is applied on the output of the attention layer to endow the model with non-linearity. It is also possible to stack several Self-Attention blocks;

3. ***Prediction Layer:*** After b Self-Attention blocks, SASRec uses Matrix Factorization to predict the probability of a user to choose an item.

Recently, Sun et al. (2019) proposed a new sequential recommendation model called BERT4Rec (Bidirectional Encoder Representations from Transformers for sequential Recommendation) that outperforms SASRec. BERT4Rec is inspired from BERT (Devlin et al., 2019) a multi-head self-attention model for text sequence modeling.

2.4 Evaluation of sequential recommendation

When designing a new recommendation model, it is important to properly evaluate it. There are two ways to test a recommender system, and these two ways are complementary. The first way is to conduct an empirical study (offline evaluation), where the goal is to compare the performance of a model to the state-of-the-art using scientific metrics. This offline evaluation is performed on several datasets and uses several metrics. All new sequential recommendation models require an empirical study. The second way is to perform an industrial study (online evaluation), where the goal is to evaluate recommender systems in real situation. Online evaluation is often carried out via A/B Testing. However this can be difficult to set up and takes time.

2.4.1 Empirical study

The goal of an empirical study is to compare recommender systems using several datasets and evaluation metrics. The datasets need to be carefully chosen. It is important to select datasets from different domains and with different levels of sparsity to better understand the strengths and weaknesses of the tested models. There is no universal evaluation protocol for recommender systems. Indeed, there are several specificities regarding the type/task of the recommender system. In the case of evaluation of sequential recommendation models, we present below the preprocessing of feedbacks, data splitting for training/testing and evaluation Metrics.

Data preprocessing: For sequential recommendation models that use implicit feedback, it is common to filter users and items to only keep those having enough interactions. One of the most common configurations is to only consider users and items that have at least 5 interactions. Note that in case of explicit feedback, ratings can be converted into implicit feedback.

Data splitting: The dataset partition strategy is a important settings for the evaluation protocol and have a considerable impact on the recommendation performance. The partition

strategy is applied on each user sequence and the most common are as follows (Quadrana et al., 2018) :

- *Time-based split* strategy consists in select a point in time to separate training data and test data. Each interaction prior to that point will be in the train set and each interaction after this point will be in the test set;
- *Percentage-based split* strategy consists in putting $x\%$ of sequence items in the train set, and the remaining $100 - x\%$ in test set. Usually $x = 80\%$. Furthermore, it is possible to complete this strategy with a 5-fold cross validation Yuan et al. (2016);
- *Fixed-size split* strategy consists in putting the k more recent items of each user sequence in the test set and the remaining item in the train set. The most commonly used size is 1, which means separating the user sequence in 2 parts (also known as leave-one-out evaluation): The most recent item is used for the test set and other items of the user sequence are used as train set.

Note that it is common to use a validation set in addition to the train and test set for a better selection of model hyperparameters.

Evaluation Metrics: The problem of sequential recommendation is directly related to ranking problems. This is why the metrics used to evaluate the performance of sequential recommendation models are based on ranking metrics. In addition to performance metrics there are other metrics to evaluate the diversity, novelty (Hurley and Zhang, 2011) or popularity of the recommendations provided by a recommender system. Note that, the ground-truth item for user u (g_u) is the item present in the test set for user u. In case of leave-one-out evaluation, the ground-truth item g_u is the last item of u. Hence, the performances of the sequential recommendation models can be assessed by the following metrics:

- ***Area Under the ROC Curve (AUC, Rendle et al. (2009)):*** This measure computes how high the ground-truth item of each user has been ranked in average:

$$AUC = \frac{1}{|U|} \sum_{u \in U} \frac{1}{|I \backslash I_{s_u}|} \sum_{j \in I \backslash I_{s_u}} \mathbb{1}(\widehat{p}_{u,g_u,t} > \widehat{p}_{u,j,t}), \tag{2.21}$$

  where the indicator function $\mathbb{1}(\widehat{p}_{u,g_u,t} > \widehat{p}_{u,j,t})$ returns 1 if the ground-truth item g_u (the positive item) is rank higher (i.e. have a higher prediction score) than the negative item j, 0 otherwise.

- ***Hit Rate at position X (HIT_X):*** The HIT_X metric returns the average number of times the ground-truth item is ranked in the top X items :

$$HIT\_X = \frac{1}{|U|} \sum_{u \in U} \mathbb{1}(R_{g_u} \leqslant R_X), \tag{2.22}$$

  where the indicator function $\mathbb{1}(R_{g_u} \leqslant R_X)$ return 1 if the ranking of the ground-truth item R_{g_u} is smaller than the Xth ranking R_X;

- ***Normalized Discounted Cumulative Gain at position X (NDCG_X):*** The NDCG_X is a position-aware metric which assigns larger weights to higher positions in order to evaluate the relevance of the recommended items :

$$NDCG\_X = \frac{DCG\_X}{IDCG\_X}, DCG\_X = \sum_{x=1}^{X} \frac{\mathbb{1}(R_x \leqslant R_X)}{log_2(x+1)}, \tag{2.23}$$

where R_x is the rank of the candidate item at position x. Here we replace form the original formulas $2^{rel_i} - 1$ by $\mathbb{1}(R_x \leqslant R_X)$ because we use a binary value to represents the relevance score (1 if it's relevance, 0 otherwise). IDCG_X is the ideal DCG calculate form the ground-truth. NDCG_X is usually reported as an average of all individual metric of all users.

- ***Precision_X (PREC_X) and Recall_X (REC_X):*** Precision and recall evaluate the relevance of the first X candidate items as follows :

$$PREC\_X = \frac{|L \cap \widehat{L_{1:X}}|}{X}, REC\_X = \frac{|L \cap \widehat{L_{1:X}}|}{|L|}, \tag{2.24}$$

where L is the list of ground-truth items and $\widehat{L_{1:X}}$ is the list of predicted items. PREC_X and REC_X are usually reported as an average of all individual metric of all users. Note that in case of leave-one-out split strategy HIT_X is equal to REC@X and proportional to PREC@X;

- ***Mean Reciprocal Rank (MRR, Shi et al. (2012)):*** The reciprocal rank is the multiplicative inverse of the rank of the first relevant item (according to the ground-truth item) given by a model for a user. The MRR is the average of reciprocal rank for all users :

$$MRR = \frac{1}{|U|} \sum_{u \in U} \frac{1}{rank_u}, \tag{2.25}$$

with $rank_u$ the rank of the first relevant item for the user u.

All these metrics are computing on the test set. To avoid heavy computation, in some articles (He et al., 2017b, Huang et al., 2018, Kang and McAuley, 2018), the authors follow the strategy of taking a sample of x negative items during the evaluation (also known as RelPlusN (Said and Bellogín, 2014)). In this thesis, we compute the metrics with all possible negative items (also known as TrainItems (Said and Bellogín, 2014)) to have an impartial assessment of all methods with exact measures instead of approximated ones. We also use the leave-one-out split strategy to create our test, validation and train set.

2.4.2 Industrial Study

To carry out an industrial study (or online evaluation), it is necessary to have direct access to a platform on which a recommendation system can be deployed. In this case, it is possible to perform evaluation on real users and have their interactions with the recommended items. The most common method used to evaluate a new recommendation model is A/B testing.

The purpose of A/B testing is to compare several models by randomly separating users into groups. For example, consider the case where a new model has to be compared to a currently used model on the platform. Users are randomly separated and each group is assigned to one of the two models. This makes possible to measure and compare the activity of the two groups, such as the number of clicks on the recommendations, the time spent on the platform, etc. It is therefore interesting to carry out an A/B test to validate the assumptions made during an empirical study. In order not to bias the results between the two groups of the A/B test, it is important to not implement other modifications on the platform (e.g. interface). An A/B test can last weeks or even months, depending on the platform's activity.

The use of multi-armed bandit can be a good alternative to A/B testing to find differences (number of clicks on the recommendations, the time spent on the platform, etc.) of performance between arms (i.e. here the recommendation models) (Scott, 2015). The advantage of multi-armed bandit compared to A/B testing is that it can adjust (i.e. gradually favoring the best model) while simultaneously exploring the performances of the two models and exploiting the best results. This generally saves time and resources.

It is also possible to carry out user surveys. The goal is to ask a part of the users to try the new recommendation system. This allows to collect their interactions and feedback. However, many users have to be included in the survey to obtain unbiased results.

In general, industrial studies are more complex to set up compared to empirical studies because they require time and a real platform environment. This is why in most of the works on recommender systems, only empirical studies are carried out. In this thesis we have been able to implement our recommendation model in a real platform and perform an industrial study.

2.5 Conclusion

In this chapter, we have introduced many models that address the problem of sequential recommendation using implicit feedback. However, on one hand, "traditional" sequential recommendation models (i.e. model without deep learning architecture) have two main drawbacks: (1) a lack of short-term personalization due to the fact that Markov chains have a fixed length regardless of the users and their considered items, and (2), if we exclude TransRec and Fossil, the other models are not fully unified as user preferences and sequential dynamic are two independent components. This damages the recommendation. On the other hand, recent approaches based on deep learning architecture considers varying sequence lengths in nonlinear models. Such models are much more difficult to interpret and their complex architecture requires a strong expertise during the implementation and the maintenance. In the next Chapter 3, we will present our proposed model that cope with the limitations of "traditional" sequential recommendation models without using complex architecture like deep learning, while having performance equal or better than these approaches.

Chapter 3

Sequential recommendation with metric models based on frequent sequences

Modeling user preferences (long-term dynamics) and user sequential dynamics (short-term dynamics) is of greatest importance to build efficient sequential recommender systems. The challenge lies in the successful combination of the whole user's history and his recent actions (sequential dynamics) to provide personalized recommendations. Existing methods capture the sequential dynamics of a user using fixed-order Markov chains (usually first order chains) regardless of the user, which limits both the impact of the past of the user on the recommendation and the ability to adapt its length to the user profile. In this chapter, we propose to use frequent sequences to identify the most relevant part of the user history for the recommendation. The most salient items are then used in a unified metric model that embeds items based on user preferences and sequential dynamics. Extensive experiments demonstrate that our method outperforms state-of-the-art of sequential recommendation models, especially on sparse datasets. We show that considering sequences of varying lengths improves the recommendations and we also emphasize that these sequences provide insight on the recommendation.

This chapter is based on Lonjarret et al. (2021), that has just been accepted for publication in Data Mining and Knowledge Discovery journal. Also, datasets and code are made available[1].

3.1 Introduction

Digital companies aim to build user loyalty and thus maximize the time spent by their users on their platforms. The more time a user spends on a platform, the more profitable it is for the company. As platforms have a large number of choices, it making difficult for a user to find relevant items. To cope this problem and build user loyalty, it has become essential to provide to users the most appropriate items based on their previous actions. This is

[1] `https://bit.ly/3gwZAOF`

what recommender systems is build for. In this chapter, we tackle the problem of sequential recommendation that aims to predict/recommend to a user the next item from implicit feedback. This is a challenging problem whose importance has been gradually recognized by researchers. Indeed, user preferences (i.e., the long-term dynamics) and user sequential dynamics (i.e., the short-term dynamics) need to be fruitfully combined to account for both personalization and sequential transitions. There are several variants to this problem which lie outside the scope of this problem. Hence, we restrain our study to methods that only take as input users' sequences to make recommendation, without considering additional information.

The first methods, which paved this research field, aim at capturing either user preferences or user sequential dynamics. User preferences have been identified using methods such as Matrix Factorization (MF) (Koren, 2009a, Koren and Bell, 2015) that uses inner products to decompose a compatibility matrix between users and items, or FISM (Kabbur et al., 2013) that splits a similarity matrix between items to better account for transitive relationships between items. Sequential dynamics has been caught by Markov Chains (MC) for the calculation of the conditional probability of appearance of an item according to a fixed number L of passed items. The next generation of recommender systems is based on models that combine both sequential dynamics and user preferences, like FPMC (Rendle et al., 2010) or FOSSIL (He and McAuley, 2016) which fuse Matrix Factorization and Markov Chains. PRME (Feng et al., 2015) improves FOSSIL and FPMC by replacing inner product with Euclidean distance. Such a metric embedding brings a better generalization, mostly due to the respect of the triangle inequality. However, it still suffers from a lack of personalization on the short term part, for example due to the fact that Markov chains have a fixed length regardless of the users and their considered items. Moreover, these models combine the user preferences and sequential dynamics using two embeddings – one for the user preferences and another one for the sequential dynamics – which can damage the quality of the model. More recently, and especially since the traces left by the users become longer, a growing research effort (Chen et al., 2018, Tang and Wang, 2018, Zhang et al., 2019) has been dedicated to the investigation of the use of deep learning technique for sequential recommendation. Yet, this kind of methods have a complex architecture which need large amounts of data and can have troubles on sparse datasets. These statements are confirmed by our experiments.

To cope with these limitations, we propose a new model **REBUS** (Recommendation Embedding Based on freqUent Sequences) that uses (1) frequent sequences to identify the part of user history that is the most relevant for recommendation and using these sequences to estimate Markov Chains of variable orders and (2) a unified metric embedding model based on user preferences and user sequential dynamics.

Figure 3.1 illustrates how **REBUS** works. It consists of embedding items into latent vectors and using these vectors to represent user preferences and user sequential dynamics:

- In Figure 3.1 (A), user preferences are represented by the weighted sum of all the items of the user history ;

- in Figure 3.1 (B) user sequential dynamics is represented by the weighted sum of the most representative recent items of the user history. These most representative items are those that match frequent sequences over the entire dataset. These frequent sequences are used as a proxy to know which items in the user history are important to capture sequential dynamics ;

- In Figure 3.1(c), **REBUS** recommend the item that is the nearest to the vector resulting from the weighted sum of user preferences and sequential dynamics.

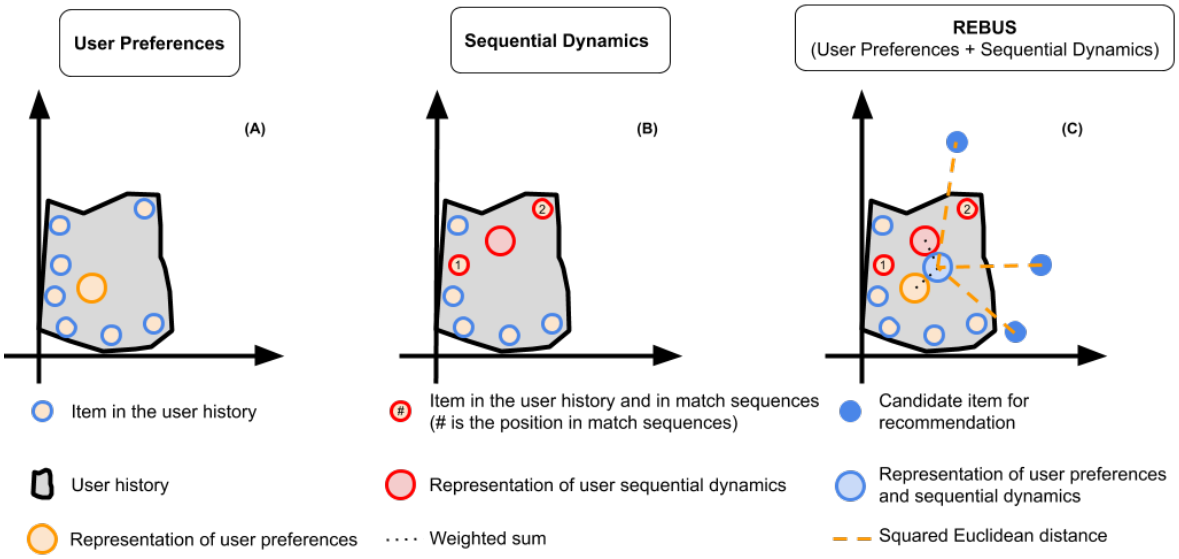

Figure 3.1: Overview of **REBUS** that accommodates user preferences and sequential dynamics through Euclidean distances. (a) **User preferences** are represented by the embedding of all the user's history items. (b) **Sequential Dynamics** is represented by the embedding of the user's items that match a frequent sequence. The temporal order of the items is taken into account with a temporal damping factor. (c) **REBUS** takes the nearest item of the weighted sum of user preferences and sequential dynamics for recommendation.

Our contributions are summarized as follows. We develop a new method, **REBUS**, that integrates and unifies metric embedding models to capture both user preferences and sequential dynamics. Especially, the use of personalized sequential patterns makes it possible to select the part of the recent user's history that is of most interest for the recommendation. In an empirical study over 12 datasets, we show that **REBUS** outperforms 9 state-of-the-art algorithms (that cover the different types introduced in Chapter 2) on sparse dataset and have nearly the same performance on dense datasets. Furthermore, we show that the sequences we use to model user sequential dynamics can provide additional insights on the recommendations. Finally, for reproducibility purposes, data and code are made available here: `https://bit.ly/3gwZAOF`.

The remaining of this chapter is organized as follows. Section 3.2 introduces the notations. **REBUS** model is defined in Section 3.3. We report an extensive empirical study in Section 3.4 and conclude in Section 3.5.

3.2 Notations

In this section, we introduce the notations we use throughout this chapter to define our proposal.

U and I are used to respectively denote the set of users and the set of items. The symbols u and i stand for individual users and items (i.e., $u \in U$ and $i \in I$). The trace left by a user u is the sequence of items $s_u = \langle i_1, \ldots, i_p \rangle$, with $i_\ell \in I$, $\ell = 1 \ldots p$, and i_p being the most recent item. Let $s_u^{[1,t]}$ be the substring of s_u starting at the 1^{st} and oldest item, and ending at the t^{th} and more recent one. We use I_{s_u} to denote items that appear in s_u and $I_{s_u^{[1,t]}}$ for those that belong to $s_u^{[1,t]}$. Notations used throughout this chapter are summarized and exemplified in the Table page ix. Notice that, this table contains all the notations of this thesis.

3.3 REBUS

Our proposed model, **REBUS** is a metric embedding model in which only items are projected. Their corresponding embedding vectors are influenced by both the preferences of the user and their sequential dynamics. The user preferences are wrapped in the model by constraining the latent vector P_i of an item that should be recommended to a user to be as close as possible to the average vector of the embedding of the items contained in the user history. In order to identify the part of the sequence that is most characteristic of a user, we consider frequent sequential patterns. These frequent sequences are both present in the history of several users (whose minimum number is specified by a threshold) while allowing to ignore certain items which are not sufficiently characteristic of their general behavior. Once the most important items are identified for a user, their order is taken into account in the model thanks to a damping factor based on the position of the item in the sequence. This allows sequences of different lengths to be used in a unified manner. The embedding vectors are then learned using the Bayesian personalized ranking optimization method (Rendle et al., 2009) on our model. These steps are detailed below.

3.3.1 Long-term metric-based model

To model user preferences, we follow the way paved by FISM (Kabbur et al., 2013) and FOSSIL (He and McAuley, 2016) while replacing inner product with Euclidean distance. The objective is to compute a latent vector P_i for each item i, so that the prediction for a user u to choose i varies in the opposite way to the distance between the sum of the items already chosen by u and the item i. Moreover, in order to not overweight items that appear in long transactions – items selected with many others – we normalize the user preferences by the inverse of the number of items in the sequence of user u:

$$\widehat{p}_{u,i} \propto -||\frac{1}{|I_{s_u} \setminus \{i\}|^\alpha} \sum_{j \in I_{s_u} \setminus \{i\}} P_j - P_i||_2^2.$$

As in FISM or FOSSIL, the hyperparameter α controls the degree of agreement between items: When $\alpha = 1$, the long-term part is equivalent to the barycenter of the latent vectors; The closer α is to 0, the more it is equivalent to the sum of latent vectors. In our experiments, we observed that the best values of α lie between 0.7 and 1.

In the learning phase, we estimate the prediction associated to item i and user u at position t, where i is the t^{th} item taken by u. Note that, for a user u, the first item is the most older one ($t = 1$) and the last taken item is the most recent one ($t = |s_u|$). It seems also rightful to restrict the set of items to be considered to those older than t ($I_{s_u^{[1,t[}}$) (even if

it is not the choice made by FISM and FOSSIL). This choice was confirmed empirically. In addition, we limit the number of items to be considered to the recent ones with MAX_LENGTH hyperparameter. It allows **REBUS** to be more flexible by controlling the temporal window that influences the user's preferences and to get rid of the old past. All in one, this leads to the following equation for estimating user long-term preferences:

$$\widehat{p}_{u,i,t} \propto - || \frac{1}{|I_{s_u^{[x,t[} \setminus \{i\}|^{\alpha}} \sum_{j \in I_{s_u^{[x,t[} \setminus \{i\}} P_j - P_i ||_2^2, \tag{3.1}$$

with $x = \max(t - \text{MAX\_LENGTH}, 1)$.

3.3.2 Short-term dynamics modeled by frequent patterns

As explained in Section 2.3.3, it has been shown that taking into account short-term individual dynamics improves the recommendation. However, existing approaches only consider a fixed, short and consecutive part of the history to make the recommendation. In contrast, **REBUS** takes into account parts of the user history which may be of different lengths for each user and also not necessarily consecutive. Finding the most adapted sequential pattern for a user u at a position t is accomplished in two steps:

1. Computing a set F of representative sequences of users' histories;
2. Identifying a sequence that personally represents a user u.

We have explored two scenarios. In the first scenario, we extract *frequent subsequences* from user's histories, and identify the subsequence that *perfectly matches* the history of a given user when making the recommendation. In the second scenario, we extract *frequent substrings* from user's histories and identify the substring that best characterizes a user using a string alignment algorithm that allows to skip some uninformative items. We implemented both scenarios and found that the quality of the results are very similar. However, the model based on frequent subsequences (i.e. Scenario 1) has a higher cost due to the size of the collection of frequent subsequences, that is much greater than the one of frequent substrings (i.e. Scenario 2). We thus detail below the second scenario.

Computing representative sequences from users' histories

A recommendation made for a user is a generalization of behaviors observed for many other users of the system. To capture the short-term dynamics of users, we want to identify substrings of interacted items that characterize the possible short-term dynamics in the system. That is to say, we want to get sets of items ordered in time that the users of the system are likely to consider in the same order. Substrings that can account for user sequential dynamics are the ones that appear in many user's histories. We identify them by extracting frequent substrings (Gusfield, 1997) that appear in at least MINCOUNT user sequences and are at most of size L. MINCOUNT and L are hyperparameters of our model. The obtained substrings constitute the set F. Three important points should be stressed here:

- The set F is computed once at the beginning of **REBUS** and then is used during the learning phase;

- The computation time of F takes less than a minute in our experiments and is therefore not a computational bottleneck;
- Each substring of a frequent substring is also frequent and thus belongs to F.

Deriving from F a personalized sequence for u

Once F is computed, the objective is to find which substring to use as personalized context for user u at position t. This substring, denoted $m_{s_u^{[1,t[}}$, must be included in the user's history while allowing the latter to contain additional items. To do that, we adapt the "*Exact matching with wildcards*" algorithm (Gusfield, 1997) to compute the longest substring among the substrings in F that end to the most recent item in $s_u^{[1,t[}$ that belongs to F. The function is presented in Algorithm 1. It consists to pick up the most recent item of $s_u^{[1,t[}$ that matches a substring of F. After that, it identifies the longest substring in F that ends with the previously found item and which is contained in the user sequence. If this process ends without any substring matching, ${s_u}^{t-1}$ is taken as personalized context for u at position t, where ${s_u}^{t-1}$ is the most recent item considered by u excluding s_u^t, the ground truth item. It allows our model to always consider sequential dynamics. It is important to notice that the personalized context for a given user relies on its last actions and therefore varies according to position t.

As an example, let us consider the set of frequent substrings $F = \{\langle 0\rangle, \langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 4\rangle, \langle 0,1\rangle, \langle 1,3\rangle, \langle 0,1,3\rangle, \langle 1,2\rangle, \langle 2,4\rangle, \langle 1,2,4\rangle\}$ and consider the user sequence $s_u^{[1,7[} = \langle 0,1,2,3,4,5\rangle$. The most recent item that can be exploited is 4 as item 5 does not appear in F. The longest sequence of F that ends with 4 and that only contains items of $s_u^{[1,7[}$ in the same order is $m_{s_u^{[1,7[}} = \langle 1,2,4\rangle$. Indeed, the substring $\langle 1,2,*,4\rangle$ matches $s_u^{[2,5]}$ with $*$ a wildcard. If the user sequence is $s_u^{[1,4[} = \langle 7,8,9\rangle$, none of the substrings of F matches $s_u^{[1,4[}$, and $m_{s_u^{[1,4[}} = \langle 9\rangle$.

Constraining P_i with short-term dynamics

The personalized sequence $m_{s_u^{[1,t[}}$ is then used to constrain the latent vector P_i to be as close as possible to the items contained in $m_{s_u^{[1,t[}}$. A damping factor η_r, depending on the item position r, is used to increase the importance of recent items of $m_{s_u^{[1,t[}}$. With $P_{m^r_{s_u^{[1,t[}}}$ be the vector corresponding to the item of $m_{s_u^{[1,t[}}$ at position r, we have:

$$\widehat{p}_{u,i,t} \propto -||\sum_{r=1}^{R} \eta_r P_{m^r_{s_u^{[1,t[}}} - P_i||_2^2. \tag{3.2}$$

where $R = |m_{s_u^{[1,t[}}|$ is the number of items in $m_{s_u^{[1,t[}}$. The value of η_r increases with r to give more weight to recent items. To normalize the short-term part of **REBUS**, we make η_r follow the softmax (normalized exponential) (Bishop, 2006) of the following linear function:

$$\eta_r = \frac{e^{\frac{r}{R}-1}}{\sum_{r=1}^{R} e^{\frac{r}{R}-1}}$$

Let us consider the same example as above where $m_{s_u^{[1,7[}} = \langle 1,2,4\rangle$ and $\langle 1,2,*,4\rangle$ matches $s_u^{[2,5]}$. Positions of items $\langle 1,2,4\rangle$ are respectively 1, 2 and 3 and $|m_{s_u^{[1,7[}}| = 3$ because we do

Algorithm 1: ExactMatchWithWildCard($s_u^{[1,t[}, F$)

Input: A user sequence $s_u^{[1,t[} = \langle {s_u}^1, \cdots, {s_u}^{t-1} \rangle$ and the set of frequent sequences F
Output: $m_{s_u^{[1,t[}}$, the longest substring among the substrings in F that end to the most recent item in $s_u^{[1,t[}$ that belongs to F, or ${s_u}^{t-1}$ if any.

$sequence \leftarrow \langle {s_u}^1, \cdots, {s_u}^{t-1} \rangle$
$path \leftarrow \langle\rangle$
for $i = t-1$ ***downto*** *1* **do**
  $item \leftarrow s_u^i$
  if $path = \langle\rangle$ **then**
    if $item \in F$ **then**
      $path \leftarrow \langle item \rangle$
  else
    if $\langle item \rangle \cdot path \in F$ **then**
      $path \leftarrow \langle item \rangle \cdot path$ ▷ The concatenation of strings $\langle item \rangle$ and $path$
if $path = \langle\rangle$ **then**
  $path \leftarrow \langle {s_u}^{t-1} \rangle$
return $path$ ▷ Here $m_{s_u^{[1,t[}} \leftarrow path$

not take into consideration wildcard item $*$. Thus $\eta_1 \approx 0.23$, $\eta_2 \approx 0.321$ and $\eta_3 \approx 0.448$. As expected the most recent item 4 has a greater importance compared to items 1 and 2.

3.3.3 The long-term and short-term metric embedding model

The embedding of items into a metric space has two main advantages. First, it brings better generalization as Euclidean distances preserve the triangle inequalities. Second, it makes it possible to fully unify the long and short-term dynamics (as expressed by equations 3.1 and 3.2) of each item into a single embedding vector resulting from the computation of one distance:

$$\widehat{p}_{u,i,t} \propto -||(\text{Long-term} + \text{Short-term}) - P_i||_2^2.$$

It is also usual to add a bias term β_i specific to each item, and a hyperparameter γ to weigh the importance between long-term and short-term dynamics:

$$\widehat{p}_{u,i,t} \propto -\Big(\beta_i + ||\big(\gamma \frac{1}{|I_{s_u^{[x,t[}} \setminus \{i\}|^\alpha} \sum_{j \in I_{s_u^{[x,t[}} \setminus \{i\}} P_j + (1-\gamma) \sum_{r=1}^{R} \eta_r P_{m^r_{s_u^{[1,t[}}}\big) - P_i||_2^2\Big) \tag{3.3}$$

3.3.4 Bayesian Personalized Ranking optimization criterion

The goal for a sequential recommender system is, for all users, to rank the ground-truth item higher than all other items. Bayesian Personalized Ranking (BPR) (Rendle et al., 2009) formalizes this problem as maximizing the posterior probability of the model parameters θ, given the order relation $>_{u,t}$ on items: $p(\theta \mid >_{u,t})$. Using Bayes' rule, the probability is proportional to $p(>_{u,t} \mid \theta)p(\theta)$. The goal is thus to identify the parameters that maximize

the likelihood of correctly ordering items. It is formally expressed as having $i >_{u,t} j$, which means that i is ranked higher than item j for user u at position t with $i = s_u^t$ the ground truth item. Assuming independence of items, their orders and users, this leads to estimate the model parameters by the maximum a posteriori probability (MAP):

$$\begin{aligned} \arg\max_{\theta} &= \ln \prod_{u \in U} \prod_{t=2}^{|s_u|} \prod_{j \neq s_u^t} p(s_u^t >_{u,t} j|\theta)\ p(\theta) \\ &= \sum_{u \in U} \sum_{t=2}^{|s_u|} \sum_{j \neq s_u^t} \ln p(s_u^t >_{u,t} j|\theta) + \ln p(\theta) \end{aligned} \tag{3.4}$$

The parameters of **REBUS** model are the embedded vectors P_i and the bias terms β_i for $i \in I$. $p(s_u^t >_{u,t} j|\theta)$ is the probability that the ground truth item s_u^t is correctly ranked with respect to j by the model, that is:

$$\begin{aligned} p(s_u^t >_{u,t} j \mid \theta) &= p(\widehat{p}_{u,s_u^t,t} >_{u,t} \widehat{p}_{u,j,t}|\theta) \\ &= p(\widehat{p}_{u,s_u^t,t} - \widehat{p}_{u,j,t} >_{u,t} 0|\theta) \end{aligned}$$

This quantity is approximated by $\sigma(\widehat{p}_{u,s_u^t,t} - \widehat{p}_{u,j,t})$, where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic sigmoid function. Taking as prior probability for θ a normal distribution with zero mean and $\lambda_\theta I$ as variance-covariance matrix, the criterion to optimize (equation 3.4) becomes:

$$\arg\max_{\theta} = \sum_{u \in U} \sum_{t=2}^{|s_u|} \sum_{j \neq s_u^t} \ln \sigma(\widehat{p}_{u,s_u^t,t} - \widehat{p}_{u,j,t}) - \lambda_\theta ||\theta||^2, \tag{3.5}$$

where λ_θ is a regularization hyperparameter.

3.3.5 Model training

REBUS learns the embedded vectors P_i and the bias terms β_i by maximizing equation 3.5. Hyperparameters – that is to say α, γ, L, minCount, regularization hyperparameters of the bias terms β_i and regularization hyperparameters λ_θ – are chosen using a grid search strategy. The learning rate as well as k, the length k of embedded vectors, are fixed by the analyst. The hyperparameter k must be chosen knowing that the larger it is, the more precise the item vectors and the more costly the computation of the model.

Item embedding P_i are randomly initialized using Xavier initialization (Glorot and Bengio, 2010) and the bias terms β_i are initialized to zero.

The parameters are learned using a variant of Stochastic Gradient Descent (SGD) called *Adam* optimizer (Kingma and Ba, 2014) with a batch size of 128 examples. For each batch, it consists in uniformly sampling a user u, a position t, that gives the positive item $i = s_u^t$, and a 'negative' item j.

3.3.6 Applying the model for recommendation

Once **REBUS** has been trained, it can be used to make recommendations. Considering the past actions of a user $s_u^{[1,t]}$, the most appropriate frequent sequence $m_{s_u^{[1,t]}}$ is found using

Algorithm 1 and used in Equation 3.3, with i a candidate item. The item with the highest $\widehat{p}_{u,i,t}$ value (or equivalently the smallest Euclidean distance) is recommended. The complexity for making recommendation for the user u is equal to $O(REBUS_{Comp} \times |I \backslash I_{s_u}| \times k)$ where $REBUS_{Comp}$ is the complexity of **REBUS** operations. Note that **REBUS** can recommend more than one item, e.g. the Top-N items associated to the N smallest Euclidean distances.

To illustrate how it works, let us consider the following example:

- $I = \{A, B, C, D, E, F\}, \qquad s_u = \langle A, B \rangle$
- $\widehat{p}_{u,C} = 0.9, \quad \widehat{p}_{u,D} = 0.6, \quad \widehat{p}_{u,E} = 0.1, \quad \widehat{p}_{u,F} = 0.7$

In this example, **REBUS** will recommend item C at the first position, followed closely by F and D. The last recommended item will be E. In this example E have a low recommendation score as it is never or rarely found in sequences containing items A and B.

3.3.7 Customization of REBUS

REBUS can be easily customized, allowing it to adapt even more precisely to the data. Directly from **REBUS**, two variants can be realized by taking the extreme values for the parameter γ:

- If $\gamma = 1$, only the long-term part of Equation 3.3 is considered. This variant is called hereafter **REBUS_UP**;
- If $\gamma = 0$, only the short-term part of Equation 3.3 is considered. This variant is called hereafter **REBUS_SD**;

Furthermore, we consider other variants of **REBUS**, named **REBUS**$_{XMC}$, where Markov chains with a fixed order X are used to model the short-term dynamics instead of the personalized sequence $m_{s_u^{[1,t[}}$. The damping factor η_r is also used to increase the importance of recent items. Notice that **REBUS**$_{XMC}$ is less personalized on the short-term part than **REBUS**, but it focuses more on the user's recent actions which can be an advantage on some datasets. The substitution of the personalized sequence by Markov chains with a fixed order X is straightforward and consists in replacing $m_{s_u^{[1,t[}}$ by $s_u^{[t-X,t[}$ in R in order to have $R = |s_u^{[t-X,t[}|$ and replace $m^r_{s_u^{[1,t[}}$ by $s_u^{(t-X-1)+r}$ in Equation 3.3. The final equation of **REBUS**$_{XMC}$ is:

$$\widehat{p}_{u,i,t} \propto -\Big(\beta_i + ||\big(\gamma \frac{1}{|I_{s_u^{[x,t[} \backslash \{i\}}|^\alpha} \sum_{j \in I_{s_u^{[x,t[} \backslash \{i\}}} P_j + (1-\gamma) \sum_{r=1}^{R} \eta_r P_{s_u^{(t-X-1)+r}}\big) - P_i||_2^2\Big) \tag{3.6}$$

As for **REBUS**, a variant can be directly created from **REBUS**$_{XMC}$ when $\gamma = 0$. In this case, only the short-term part of Equation 3.6 is considered. This variant is named **REBUS_SD**$_{XMC}$, where X represent the order of the Markov Chains.

We have shown here that **REBUS** can be simply declined in four variants, allowing to simply adapt to the specificities present in some datasets. Moreover, all these variants will allow us to analyze how the long-term and short-term dynamics individually impact **REBUS** and compare the personalized sequence against fixed-order Markov chains.

3.3.8 Discussion

As said before, one of the key characteristics of **REBUS** is to only embed items, like SASRec and FMC do. This characteristic brings two advantages compared to other models that embed items and users (like FPMC, PRME, TransRec and CASER). First, it has a smaller space complexity and second, it suffers less from the cold start problem. These two aspects are evaluated below.

Space Complexity: The learned parameters of **REBUS** are the item embeddings – each being of dimension k – and the items bias terms. This results in a number of parameters in $O(|I| \times k + |I|)$. Thus, the complexity of our model does not grow with the number of users, unlike other models that embed users and items.

Recommending items to new users: We can say that **REBUS** suffers less from the cold-start problem, since it can make recommendations to new users who have interacted only with a single item of the system. We show in the next section that **REBUS** has good performance compared to other models when it comes to the problem of cold start users.

3.4 Experiments

In this section, we present a thorough empirical study. We first begin by describing the 12 real-world datasets we consider, as well as the questions these experiments aim to answer. Then, we report an extensive comparison of **REBUS** with 9 state-of-the-art algorithms for sequential recommendation. Results demonstrate that **REBUS** outperforms state-of-the-algorithms according to several metrics in most of the cases. Finally, we provide a deeper analysis of **REBUS**, especially how frequent substrings are actually used in sequential recommendation, the impact of the user preferences with regard to the sequential dynamics and how **REBUS** can be tuned to give better results. For reproducibility purposes, the source code and the data are made available [2].

3.4.1 Datasets and aims

To evaluate the performance of **REBUS** on both sparse and dense datasets from different domains, we consider 5 well-known benchmarks and introduce a new dataset:
Amazon was introduced by (McAuley et al., 2015). It contains Amazon product reviews from May 1996 to July 2014 from several product categories. We have chosen to use the 3 following diversified categories: *Automotive, Office product* and *video games.*
MovieLens 1M[3] (Harper and Konstan, 2015) is a popular dataset including 1 million movie ratings from 6040 users between April 2000 and February 2003. We used the datasets pre-processed by selecting the most recent x ratings for each user, $x \in \{5, 10, 20, 30, 50\}$. These datasets allow us to study the performance of **REBUS** on the same dataset but with different levels on sparsity (i.e. MovieLens with the 5 most recent ratings will be more sparse than MovieLens with the 50 most recent ratings). -

[2] `https://bit.ly/3gwZAOF`
[3] `http://grouplens.org/datasets/movielens/1m/`

Epinions describes consumer reviews for the website Epinions from January 2001 to November 2013. This dataset was collected by the authors of (Zhao et al., 2014).
Foursquare depicts a large number of user check-ins on the Foursquare website from December 2011 to April 2012.
Adressa (Gulla et al., 2017) includes news articles (in Norwegian). The dataset was offered by Adresseavisen, a local newspaper company in Trondheim, Norway.
Visiativ[4] is a new dataset that is made of resource downloads from *myCADservices* platform from November 2014 to August 2018. This dataset is relevant to us because it is the case study of this thesis and from there, we can assess sequential recommendation since the downloaded documents are tutorials that users read to train themselves and improve their comprehension of some specific software.

For each dataset, ratings are converted into implicit feedback and we only consider users and items that have at least 5 interactions. The main characteristics of these datasets before and after preprocessing are reported in Tables 3.1 and 3.2. This preprocessing allows models to focus on users that have enough actions in order to make good recommendation. We can notice that for Epinions, Foursquare and Amazon datasets, there are a large number of users and items that are deleted by the preprocessing.

Table 3.1: Main characteristics of the datasets before preprocessing.

	Datasets	#Users	#Items	#Actions	#A/#U	#A/#I	Sparsity
Others	Epinions	117323	42447	193662	1.65	4.56	99.996%
	Foursquare	485381	83999	1021966	2.11	12.17	99.997%
	Adressa	607805	13522	2624554	4.32	194.1	99.968%
	Visiativ	5068	945	23573	4.65	24.94	99.508%
Amaz.	Ama-Auto	851432	320116	1373794	1.61	4.29	99.999%
	Ama-Office	909314	130006	1243186	1.37	9.56	99.999%
	Ama-Game	826767	50210	1324753	1.60	26.38	99.997%
MovieLens	ML-5	6040	2848	30175	5.00	10.60	99.825%
	ML-10	6040	3114	59610	9.87	19.14	99.683%
	ML-20	6040	3324	111059	18.39	33.41	99.447%
	ML-30	6040	3391	152160	25.19	44.87	99.257%
	ML-50	6040	3467	215676	35.71	62.21	98.970%

[4] `https://www.visiativ.com/en-us/`

Table 3.2: Main characteristics of the datasets after preprocessing (users and items that have at least 5 interactions).

	Datasets	#Users	#Items	#Actions	#A/#U	#A/#I	Sparsity
Others	Epinions	5015	8335	26932	5.37	3.23	99.94%
	Foursquare	43110	13335	306553	7.11	22.99	99.95%
	Adressa	141933	3257	1861901	13.12	571.66	99.60%
	Visiativ	1398	590	16417	11.74	27.83	98.01%
Amaz.	Ama-Auto	34316	40287	183573	5.35	4.56	99.99%
	Ama-Office	16716	22357	128070	7.66	5.73	99.97%
	Ama-Game	31013	23715	287107	9.26	12.11	99.96%
MovieLens	ML-5	6040	2848	30175	5.00	10.60	99.82%
	ML-10	6040	3114	59610	9.87	19.14	99.68%
	ML-20	6040	3324	111059	18.39	33.41	99.45%
	ML-30	6040	3391	152160	25.19	44.87	99.26%
	ML-50	6040	3467	215676	35.71	62.21	98.97%

To evaluate the performance and the limits of **REBUS**, we propose to answer the following questions:

- What are the performances of **REBUS** compared to those of state-of-the-art algorithms for sequential recommendation for sparse and dense datasets? What about **REBUS**'s performance in presence of cold-start users?
- Does **REBUS** take benefit from sequential behaviors in a better way than Markov chains of fixed order do?
- What is the most important component? The user preferences? The sequential dynamics or both?
- What about the recommendations? Does **REBUS** provide diverse recommendations?

3.4.2 Comparison methods

We compare **REBUS** to 9 state-of-the-art methods designed for both item and sequential recommendation:

- Popularity (**POP**), the naive baseline that ranks items according to their popularity (Aggarwal, 2016);
- Matrix Factorization with Bayesian Personalized Ranking (**BPR**) (Rendle et al., 2009), that recommends items by considering only user preferences with matrix factorization techniques;
- Factorized Markov Chains (**FMC**) (Rendle et al., 2010), that is based on the factorization of the item-to-item transition matrix;

- Factorized Personalized Markov Chains (**FPMC**) (Rendle et al., 2010), that considers both user preferences and user dynamics thanks to matrix factorization and first-order Markov chains;

- Personalized Ranking Metric Embedding (**PRME**) (Feng et al., 2015), that embeds user preferences and user dynamics into two Euclidean distances;

- Translation-based Recommendation (**TransRec**) (He et al., 2017a), that unifies user preferences and sequential dynamics with translations;

- Convolutional Sequence Embedding Recommendation (**CASER**) (Tang and Wang, 2018), a CNN-based method that captures both user preferences and user sequential dynamics;

- Self-Attentive Sequential Recommendation (**SASRec**) (Kang and McAuley, 2018), a self-attention based model that captures both user preferences and user sequential dynamics ;

- Session-based KNN (**S-KNN**) (Jannach and Ludewig, 2017), a nearest-neighbor-based approach designed for session-recommendation. S-KNN compares the current session with past sessions in training data. In our experiments, a user's sequence is assimilate to a session [5].

Table 3.3 provides a comparison of the above methods according to several criteria: whether they are personalized, sequentially aware, metric-based, integrated into a unified model, based on personalized order Markov chains or on a model that only embed items. Indications about the time needed to train the models are also reported in the last column. For fair comparisons, we return the time required for 25 epochs on Amazon Office. As we can see, the time needed for **REBUS** is comparable to the one of other methods. It includes the time needed to extract frequent substrings.

3.4.3 Experimental settings

We apply the same partition as (Bayer et al., 2017, He et al., 2017a,b, Kang and McAuley, 2018) for user sequences (also known as *leave-one-out* evaluation). Indeed, for each dataset, the user sequences are split into 3 parts:

1. The most recent item is used for the test. It is named *ground-truth item* and denoted g_u;

2. The second most recent item is used for the validation when learning the model;

3. Other items of the user sequence are used to train the models.

[5] Note that we did not make experiments with **GRU4Rec** as it has already been shown that it is outperformed by both **CASER** (Tang and Wang, 2018) and **SASRec** (Kang and McAuley, 2018), and sometimes also by **S-KNN** (Ludewig and Jannach, 2018).

Table 3.3: Overview of the different models. **P**: Personalized?, **S**: Sequentially-aware?, **M**: Metric-based?, **U**: Unified model?, **O**: personalized Order Markov chains?, **I**: embed only items?, **T**: Time (in seconds) for the Amazon Office dataset with 25 epochs.

Property	P	S	M	U	O	I	T
Pop	✗	✗	✗	✗	✗	✗	∼1s
BPR	✓	✗	✗	✗	✗	✗	∼21s
FMC	✗	✓	✗	✗	✗	✓	∼19s
FPMC	✓	✓	✗	✗	✗	✗	∼26s
PRME	✓	✓	✓	✗	✗	✗	∼25s
TransRec	✓	✓	✓	✓	✗	✗	∼27s
S-KNN	✓	✓	✗	✗	✗	✗	∼1s
CASER	✓	✓	✗	✓	✗	✗	∼75s
SASRec	✓	✓	✗	✓	✗	✓	∼20s
REBUS	✓	✓	✓	✓	✓	✓	∼30s

Contrary to (He et al., 2017b, Huang et al., 2018, Kang and McAuley, 2018), which follow the strategy of taking a sample of x negative items during the evaluation (i.e. items that a user has not interacted with), we compute the metrics with all possible negative items (also known as *TrainItems* (Said and Bellogín, 2014)) to have an impartial assessment of all methods with exact measures instead of approximated ones. The performances of the models are assessed by three widely used metrics for sequential recommendation (He et al., 2017a, Kang and McAuley, 2018, Rendle et al., 2009):

Area Under the ROC Curve (AUC):

$$AUC = \frac{1}{|U|} \sum_{u \in U} \frac{1}{|I \backslash I_{s_u}|} \sum_{j \in I \backslash I_{s_u}} \mathbb{1}(\widehat{p}_{u,g_u,t} > \widehat{p}_{u,j,t}),$$

where the indicator function $\mathbb{1}(b)$ returns 1 if its argument b is *True*, *0* otherwise. This measure calculates how high the ground-truth item of each user has been ranked in average.

Hit Rate at position X (HIT_X):

$$HIT\_X = \frac{1}{|U|} \sum_{u \in U} \mathbb{1}(R_{g_u} \leqslant R_X),$$

where R_{g_u} is the ranking of the ground-truth item and R_X is the Xth ranking. The HIT_X function returns the average number of times the ground-truth item is ranked in the top X items. We compute HIT_5, HIT_10, HIT_25 and HIT_50.

Normalized Discounted Cumulative Gain at position X (NDCG_X):

$$NDCG\_X = \frac{1}{|U|} \sum_{u \in U} \frac{\mathbb{1}(R_{g_u} \leqslant R_X)}{log_2(R_{g_u} + 1)}.$$

The NDCG_X is a position-aware metric which assigns larger weights to higher positions. We compute NDCG_5, NDCG_10, NDCG_25 and NDCG_50. Note that, as we use the *leave-one-out split strategy*, $INDCG\_X$ is always equal to 1 because we only have one ground-truth item.

We also consider two metrics to assess how diverse are the recommendations made by a model.

Popularity rate X (Pop_X): The POP_X is the proportion of predicted items that are similar to the X most popular items. We can note that the POP model (the most popular) should have a Pop_X value equal to 1. However, it is not always the case since already consumed items are not recommended to user.

Diversity rate X (Div_X): The DIV_X is the proportion of items that are predicted at least once to all users (the number of unique predicted items divided by the total number of items). Div_X is sometimes referred to as aggregate diversity (Adomavicius and Kwon, 2012).

REBUS is implemented using *TensorFlow* and the *Adam* optimizer (Kingma and Ba, 2014). In order to make a fair comparison, we implement BPR, FMC, FPMC, PRME, TransRec with the same architecture as **REBUS**. For CASER and SASRec, we use the code provided by the authors. And for S-KNN, we used the implementation of the authors of (Ludewig and Jannach, 2018) which we modified to have equivalent evaluation. To limit the numbers of combinations to explore with grid search, we fix the following parameters for all models: The learning rate is set to 0.001, the batch size to 128 for every dataset, except for Adressa dataset where we fixed the batch size to 1024 in order to reduce the computation time of the learning phase [6], the dimension of the learned latent vectors k is set to 10 and we stop the training if there is no improvement of the AUC validation for 250 epochs. We compute the AUC validation every 25 epochs. All regularization hyperparameters are taken in $\{0, 0.001, 0.01, 0.1, 1\}$. For models with other hyperparameters we have tried those given by the authors, except for CASER which requires too many hyperparameters: There are about 50000 possible combinations. Regarding specific hyperparameters of **REBUS**, $\alpha \in \{0.1, \ldots, 1.0\}$, $\gamma \in \{0.0, 0.1, \ldots, 1.0\}$ [7], MAX_LENGTH is fixed to 105, MINCOUNT has for default value 2, and L takes its value between 3 and 5 for small datasets (Visiativ, Epinions, ML-5) and between 8 and 15 for other datasets. Best hyperparameters for each dataset are reported in Appendix 7 Table 7.1 for BPR, FMC, FPMC, PRME, TransRec and **REBUS**, Table 7.2 for SASRec, Table 7.3 for CASER and Table 7.4 for S-KNN. The Best hyperparameters are also reported in repository (`https://bit.ly/3gwZAOF`) in CSV and Excel format to facilitate the reading of the data.

[6] Increasing the batch size speed up the learning phase as this makes possible to benefit even more from TensorFlow parallelization. Furthermore it has a limited effect on the performance of the model.

[7] Only for Foursquare dataset, we observed that it is better not to have γ (remove γ and $(1-\gamma)$ in Equation 3.3). It is noteworthy that recommendations may be different to the case where $\gamma = 0.5$ due to the bias terms β_i.

3.4.4 Performance study

The performances of the different methods on every dataset are reported in Table 3.4 and Table 3.5[8]. We can observe that **REBUS** outperforms other models on most of the datasets, having the best AUC value. **REBUS** also performs well on all other metrics, with an average rank between 2.83 to 3.75, an improvement between 1.35% to 6.94% over the best competitor, and the best HIT_25, HIT_50, NDCG_25 and NDCG_50 in overall. For HIT_5, HIT_10, NDCG_5 and NDCG_10, we respectively found improvements of 7.15%, 3.22%, 11.56% and 8.74% compared to the best competitor. However, these high percentages have to be nuanced because the obtained values are often low (i.e., in comparison to AUC) and the results vary a lot from one dataset to another.

In general, BPR-MF outperforms FMC. This confirms that user preferences have a more important role than the sequential dynamics in the recommendations of the next-item. As expected, SASRec performs very well on dense datasets (i.e., ML30, ML50 where $\#A/\#U > 20$). However, this model is outperformed by several others on sparse datasets. We can observe that **REBUS** also outperforms PRME and FPMC on sparse datasets. It gives evidence that having independent latent vectors to model user preferences and sequential dynamics is not an advantage for sparse datasets. S-KNN exhibits bad AUC performance because of its architecture : S-KNN cannot rank items that are not in the neighborhood of the user target. Besides, S-KNN reports fair performance on HIT_X and NDCG_X and outperforms all models on Amazon datasets – this confirms the observations made by the authors of (Ludewig and Jannach, 2018) – but it is outperformed by **REBUS** for all other datasets.

The improvement of **REBUS** according to each model is reported in Table 3.6. Considering all datasets, **REBUS** outperforms all the studied models. However, some models exhibits better performances when only focusing on MovieLens datasets (e.g., SASRec, TransRec, FPMC and FMC for which **REBUS** has negative improvements).

In Figure 3.3, we study the effect of the size of the latent vectors ($k \in \{10, 20, 30, 40, 50\}$) [9] on AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50. In most of the cases, **REBUS** has the best performances for each k value for AUC, HIT_25, HIT_50 and is one of the best models for NDCG_25 and NDCG_50. It is worth to mention that **REBUS** obtains better results when the size of the latent vectors increases, which is not always the case of other methods. For instance, SASRec does not progress when the number of latent dimensions increases for 3 of 4 datasets on AUC, HIT_25, HIT_50.

Eventually, we study the diversity of recommendations provided by each model. Figure 3.2 reports POP_X and DIV_X for each model. For POP_X, the higher the value, the more the model tends to recommend popular items. For DIV_X the higher the value, the more the model tends to recommend different items. These results give evidence that **REBUS**' recommendations are rarely based on the most popular items. However, only a subset of items (between 20% and 50%) is recommended to the users, which remains comparable to most of the other models.

[8] We only show HIT_25, HIT_50, NDCG_25 and NDCG_50 in these tables. HIT_5, HIT_10, NDCG_5 and NDCG_10 are reported in Appendix 7 Tables 7.5 and 7.6.

[9] To avoid running again the grid search, we took the best combinations of hyperparameters that we previously found for k = 10.

Table 3.4: AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 for the different models on Epinions, Foursquare, Adressa, Visiativ, Amazon-Automotive, Amazon-Office-Product and Amazon-Video-Games datasets. The last row, called *Improv. vs Best*, shows the improvement in percentage of our method compared to the other best model (best obtained results are in bold and best obtained results for concurrent models are underlined). The last column is the average performance on these 7 datasets, the average of all datasets is included in Table 3.5.

	Metric	**Epinions**	**Foursq**	**Adressa**	**Visiativ**	**Auto**	**Office**	**Games**	**Avg**
POP	AUC	0.4575	0.9169	0.9582	0.7864	0.5856	0.6412	0.7484	0.7277
	HIT25	2.25%	46.66%	19.39%	32.04%	2.54%	0.62%	3.50%	15.29%
	HIT50	3.42%	55.65%	30.89%	48.24%	3.75%	1.67%	5.16%	21.25%
	NDGC25	0.80%	18.92%	7.21%	12.87%	0.96%	0.15%	1.37%	6.04%
	NDGC50	1.03%	20.68%	9.40%	15.97%	1.19%	0.36%	1.69%	7.19%
FMC	AUC	0.5421	0.9508	0.9842	0.8354	0.6251	0.6771	0.8473	0.7803
	HIT25	1.67%	53.06%	64.23%	45.66%	2.57%	1.35%	10.64%	25.60%
	HIT50	2.63%	64.19%	77.99%	58.78%	3.78%	2.67%	15.71%	32.25%
	NDGC25	0.77%	26.13%	29.19%	23.24%	1.05%	0.52%	4.19%	12.15%
	NDGC50	0.95%	28.29%	31.84%	25.75%	1.28%	0.77%	5.16%	13.44%
BPR	AUC	0.5593	0.9474	0.9656	0.8270	0.6649	0.7072	0.8590	0.7901
	HIT25	2.88%	46.95%	25.83%	42.72%	2.90%	1.58%	7.82%	18.67%
	HIT50	4.49%	58.38%	40.16%	57.42%	4.59%	2.59%	12.42%	25.72%
	NDGC25	1.20%	20.44%	9.35%	17.13%	1.10%	0.63%	2.85%	7.53%
	NDGC50	1.51%	22.66%	12.10%	19.95%	1.43%	0.82%	3.73%	8.88%
FPMC	AUC	0.5536	0.9492	0.9848	0.8433	0.6482	0.6958	0.8777	0.7932
	HIT25	1.63%	56.46%	65.36%	46.67%	2.36%	1.60%	12.28%	26.62%
	HIT50	2.51%	65.38%	79.39%	60.36%	3.64%	2.53%	18.33%	33.16%
	NDGC25	0.70%	30.88%	30.14%	22.68%	0.88%	0.56%	4.71%	12.94%
	NDGC50	0.87%	32.61%	32.85%	25.35%	1.13%	0.74%	5.87%	14.20%
PRME	AUC	0.6071	0.9538	0.9849	0.8572	0.6749	0.7154	0.8759	0.8099
	HIT25	1.88%	55.64%	64.78%	45.09%	2.34%	2.78%	12.63%	26.45%
	HIT50	2.72%	67.23%	78.19%	59.43%	3.85%	4.87%	18.74%	33.58%
	NDGC25	0.85%	30.71%	30.33%	24.86%	0.85%	0.95%	4.89%	13.35%
	NDGC50	1.01%	32.97%	32.92%	27.61%	1.14%	1.34%	6.07%	14.72%
TransRec	AUC	0.6138	0.9619	0.9852	0.8647	0.6991	0.7320	0.8869	0.8205
	HIT25	1.98%	58.67%	65.70%	51.25%	3.98%	3.97%	10.76%	28.04%
	HIT50	3.14%	67.81%	78.81%	64.73%	5.99%	6.37%	16.51%	34.77%
	NDGC25	0.94%	33.13%	31.99%	**26.66%**	1.56%	1.38%	4.08%	14.25%
	NDGC50	1.16%	34.90%	34.52%	**29.26%**	1.95%	1.84%	5.18%	15.55%
S-KNN	AUC	0.0684	0.8330	0.9367	0.8369	0.1476	0.2938	0.6322	0.5355
	HIT25	2.51%	**63.90%**	27.77%	44.87%	**5.15%**	**5.36%**	**13.94%**	23.36%
	HIT50	3.70%	**70.75%**	40.33%	61.15%	**6.71%**	**7.32%**	**19.54%**	29.93%
	NDGC25	**1.32%**	45.92%	12.97%	21.83%	**2.49%**	**2.77%**	**5.82%**	13.30%
	NDGC50	1.55%	47.25%	15.36%	24.95%	**2.79%**	**3.15%**	**6.89%**	14.56%
CASER	AUC	0.6238	0.9259	0.9750	0.8544	0.6872	**0.7510**	0.8282	0.8065
	HIT25	2.44%	46.45%	60.65%	46.74%	2.63%	1.42%	6.45%	23.83%
	HIT50	3.88%	56.73%	74.86%	61.08%	3.64%	2.50%	10.23%	30.42%
	NDGC25	0.96%	18.39%	26.29%	21.35%	0.94%	0.45%	2.37%	10.11%
	NDGC50	1.24%	20.40%	29.04%	24.12%	1.14%	0.66%	3.10%	11.38%
SASRec	AUC	0.6251	0.9617	**0.9870**	0.8731	0.6861	0.7392	0.8812	0.8219
	HIT25	2.60%	55.04%	**67.30%**	51.76%	2.55%	3.65%	9.16%	27.44%
	HIT50	4.00%	64.97%	**82.15%**	65.38%	4.07%	6.12%	14.54%	34.46%
	NDGC25	1.03%	30.70%	29.23%	25.00%	0.95%	1.30%	3.56%	13.11%
	NDGC50	1.30%	32.62%	32.10%	27.61%	1.24%	1.77%	4.59%	14.46%
REBUS	AUC	**0.6524**	**0.9677**	0.9854	**0.8735**	**0.7184**	0.7507	**0.8934**	**0.8345**
	HIT25	**3.39%**	62.77%	66.73%	**53.33%**	4.24%	4.12%	9.30%	**29.13%**
	HIT50	**5.32%**	70.41%	79.77%	**67.17%**	6.20%	6.24%	14.61%	**35.68%**
	NDGC25	1.31%	**46.02%**	**32.00%**	24.20%	1.68%	1.52%	3.52%	**15.75%**
	NDGC50	**1.68%**	**47.49%**	**34.52%**	26.86%	2.05%	1.92%	4.54%	**17.01%**
Rank	AUC	1	1	2	1	1	2	1	1.29
of	HIT25	1	2	2	1	2	2	3	1.86
REBUS	HIT50	1	2	2	1	2	3	6	2.43
	NDGC25	2	1	1	4	2	2	7	2.71
	NDGC50	1	1	1	4	2	2	7	2.57
Improv.	AUC	4.37%	0.60%	-0.16%	0.05%	2.76%	-0.04%	0.73%	1.53%
vs Best	HIT25	17.71%	-1.77%	-0.85%	3.03%	-17.67%	-23.13%	-33.29%	3.89%
	HIT50	18.49%	-0.48%	-2.90%	2.74%	-7.60%	-14.75%	-25.23%	2.62%
	NDGC25	-0.76%	0.22%	0.03%	-9.23%	-32.53%	-45.13%	-39.52%	10.53%
	NDGC50	8.39%	0.51%	0.00%	-8.20%	-26.52%	-39.05%	-34.11%	9.39%

Table 3.5: AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 for the different models on ML-5, ML-10, ML-20, ML-30 and ML-50 datasets. The last row, called *Improv. vs Best*, shows the improvement in percentage of our method compared to the other best model (best obtained results are in bold and best obtained results for concurrent models are underlined). The last column is the average performance on the 12 datasets, including datasets of Table 3.4.

	Metric	**ML-5**	**ML-10**	**ML-20**	**ML-30**	**ML-50**	**Avg(ML)**	**Avg(All)**
POP	AUC	0.7352	0.7722	0.7919	0.7981	0.8032	0.7801	0.7496
	HIT25	7.48%	7.60%	7.57%	7.57%	7.72%	7.59%	12.08%
	HIT50	12.65%	12.93%	12.82%	12.72%	13.13%	12.85%	17.75%
	NDGC25	2.64%	2.75%	2.73%	2.66%	2.70%	2.70%	4.65%
	NDGC50	3.63%	3.76%	3.74%	3.64%	3.74%	3.70%	5.74%
FMC	AUC	0.7414	0.8054	0.8446	0.8560	0.8689	0.8233	0.7982
	HIT25	9.84%	13.84%	16.44%	16.74%	18.63%	15.10%	21.22%
	HIT50	15.25%	20.60%	24.95%	25.58%	27.72%	22.82%	28.32%
	NDGC25	3.57%	5.72%	6.62%	6.69%	7.36%	5.99%	9.59%
	NDGC50	4.61%	7.01%	8.25%	8.39%	9.11%	7.47%	10.95%
BPR	AUC	0.7623	0.8241	0.8517	0.8570	0.8629	0.8316	0.8074
	HIT25	9.65%	11.96%	11.49%	10.66%	11.24%	11.00%	15.47%
	HIT50	15.98%	19.74%	19.56%	18.74%	18.56%	18.52%	22.72%
	NDGC25	3.52%	4.16%	4.06%	3.68%	3.87%	3.86%	6.00%
	NDGC50	4.73%	5.65%	5.61%	5.22%	5.27%	5.30%	7.39%
FPMC	AUC	0.7309	0.8150	0.8629	0.8766	0.8891	0.8349	0.8106
	HIT25	7.82%	14.65%	**17.42%**	**18.65%**	**18.96%**	15.50%	21.99%
	HIT50	13.26%	22.90%	26.28%	27.60%	28.63%	23.74%	29.23%
	NDGC25	2.87%	5.57%	**6.63%**	**7.27%**	**7.41%**	5.95%	10.02%
	NDGC50	3.91%	7.15%	**8.33%**	**8.99%**	**9.26%**	7.53%	11.42%
PRME	AUC	0.7771	0.8302	0.8645	0.8765	0.8867	0.8470	0.8254
	HIT25	10.51%	15.58%	14.56%	16.19%	18.00%	14.97%	21.67%
	HIT50	16.94%	23.81%	24.39%	25.55%	27.14%	23.57%	29.41%
	NDGC25	3.78%	5.79%	5.16%	5.94%	6.62%	5.46%	10.06%
	NDGC50	5.01%	7.37%	7.04%	7.72%	8.37%	7.10%	11.55%
TransRec	AUC	0.7900	0.8477	0.8736	0.8816	0.8883	0.8563	0.8354
	HIT25	14.36%	**16.99%**	16.49%	16.00%	17.09%	16.18%	23.10%
	HIT50	22.45%	**25.60%**	**26.51%**	26.30%	27.87%	25.75%	31.01%
	NDGC25	**5.33%**	**6.30%**	5.96%	5.95%	6.35%	5.98%	10.80%
	NDGC50	6.88%	**7.95%**	7.89%	7.93%	8.42%	7.81%	12.32%
S-KNN	AUC	0.2778	0.7164	0.8115	0.8297	0.8421	0.6955	0.6022
	HIT25	9.46%	14.99%	12.85%	12.62%	11.97%	12.38%	18.78%
	HIT50	13.08%	22.62%	21.13%	20.50%	19.27%	19.32%	25.51%
	NDGC25	3.87%	5.55%	4.54%	4.48%	4.26%	4.54%	9.65%
	NDGC50	4.57%	7.01%	6.13%	5.99%	5.66%	5.87%	10.94%
CASER	AUC	0.7493	0.8118	0.8534	0.8650	0.8764	0.8312	0.8168
	HIT25	7.40%	11.36%	14.27%	15.95%	18.66%	13.53%	19.54%
	HIT50	13.58%	18.20%	22.62%	24.57%	28.18%	21.43%	26.67%
	NDGC25	2.63%	4.06%	5.35%	5.89%	7.07%	5.00%	7.98%
	NDGC50	3.82%	5.37%	6.95%	7.54%	8.90%	6.52%	9.36%
SASRec	AUC	0.8000	0.8537	0.8760	**0.8860**	**0.8957**	**0.8623**	0.8387
	HIT25	12.77%	15.91%	16.11%	18.26%	18.55%	**16.32%**	22.80%
	HIT50	20.83%	25.32%	25.96%	**28.20%**	**29.29%**	**25.92%**	30.90%
	NDGC25	4.53%	6.08%	5.81%	6.97%	7.09%	**6.10%**	10.19%
	NDGC50	6.08%	7.89%	7.70%	8.87%	9.15%	**7.94%**	11.74%
REBUS	AUC	**0.8031**	**0.8563**	**0.8772**	0.8826	0.8884	0.8615	**0.8458**
	HIT25	**14.42%**	15.27%	15.60%	15.80%	16.72%	15.56%	**23.48%**
	HIT50	**22.79%**	25.50%	26.01%	26.21%	26.91%	25.48%	**31.43%**
	NDGC25	5.32%	5.59%	5.60%	5.80%	6.03%	5.66%	**11.55%**
	NDGC50	**6.92%**	7.55%	7.59%	7.80%	7.98%	7.57%	**13.07%**
Rank of REBUS	AUC	1	1	1	2	3	1.60	1.42
	HIT25	1	4	5	7	7	4.80	3.08
	HIT50	1	2	3	4	7	3.40	2.83
	NDGC25	2	5	5	7	7	5.20	3.75
	NDGC50	1	3	5	5	7	4.20	3.25
Improv. vs Best	AUC	0.39%	0.30%	0.14%	-0.38%	-0.82%	-0.09%	**0.85%**
	HIT25	0.42%	-10.12%	-10.45%	-15.28%	-11.81%	-4.66%	**1.65%**
	HIT50	1.51%	-0.39%	-1.89%	-7.06%	-8.13%	-1.70%	**1.35%**
	NDGC25	-0.19%	-11.27%	-15.54%	-20.22%	-18.62%	-7.21%	**6.94%**
	NDGC50	0.58%	-5.03%	-8.88%	-13.24%	-13.82%	-4.66%	**6.09%**

Table 3.6: Average improvement of **REBUS** compared to each model on others datasets (e.g., Epinions, Foursquare, Adressa, Visiativ and Amazon datasets), ML datasets (e.g., ML-5, ML-10, ML-20, ML-30 and ML-50) and all datasets.

	Metric	SASRec	TransRec	PRME	CASER	FPMC	BPR	FMC	POP	S-KNN
Avg Others	AUC	1.53%	1.71%	3.04%	3.47%	5.21%	5.62%	6.95%	14.68%	55.84%
	HIT5	13.98%	7.15%	17.98%	55.76%	15.28%	116.18%	22.41%	169.83%	20.41%
	HIT10	7.25%	3.22%	12.58%	34.88%	9.62%	79.79%	13.66%	119.54%	21.85%
	HIT25	6.16%	3.89%	10.13%	22.24%	9.43%	56.03%	13.79%	90.52%	24.70%
	HIT50	3.54%	2.62%	6.25%	17.29%	7.60%	38.72%	10.64%	67.91%	19.21%
	NDGC5	23.31%	11.56%	21.48%	77.11%	22.75%	153.87%	30.78%	229.32%	16.15%
	NDGC10	17.84%	8.74%	17.98%	59.40%	18.27%	121.25%	24.09%	179.30%	17.40%
	NDGC25	20.14%	10.53%	17.98%	55.79%	21.72%	109.16%	29.63%	160.76%	18.42%
	NDGC50	17.63%	9.39%	15.56%	49.47%	19.79%	91.55%	26.56%	136.58%	16.83%
Avg ML	AUC	-0.09%	0.61%	1.71%	3.65%	3.19%	3.60%	4.64%	10.43%	23.87%
	HIT5	-10.74%	-7.49%	0.00%	11.92%	-13.43%	53.74%	-17.56%	113.86%	25.58%
	HIT10	-6.75%	-5.38%	-0.26%	11.21%	-7.19%	48.85%	-9.15%	95.45%	26.47%
	HIT25	-4.66%	-3.83%	3.94%	15.00%	0.39%	41.45%	3.05%	105.01%	25.69%
	HIT50	-1.70%	-1.05%	8.10%	18.90%	7.33%	37.58%	11.66%	98.29%	31.88%
	NDGC5	-12.13%	-8.22%	2.68%	10.74%	-14.92%	57.65%	-20.00%	123.33%	23.50%
	NDGC10	-9.38%	-6.68%	1.34%	10.23%	-10.66%	53.25%	-14.51%	107.14%	24.42%
	NDGC25	-7.21%	-5.35%	3.66%	13.20%	-4.87%	46.63%	-5.51%	109.63%	24.67%
	NDGC50	-4.66%	-3.07%	6.62%	16.10%	0.53%	42.83%	1.34%	104.59%	28.96%
Avg All	AUC	0.85%	1.24%	2.47%	3.55%	4.34%	4.76%	5.96%	12.83%	40.45%
	HIT5	20.33%	10.48%	22.18%	68.47%	23.22%	134.12%	35.03%	184.29%	19.34%
	HIT10	11.74%	5.73%	16.61%	43.00%	15.07%	90.24%	21.79%	126.63%	20.59%
	HIT25	2.98%	1.65%	8.35%	20.16%	6.78%	51.78%	10.65%	94.37%	25.03%
	HIT50	1.72%	1.35%	6.87%	17.85%	7.53%	38.34%	10.98%	77.07%	23.21%
	NDGC5	31.67%	15.51%	25.15%	95.49%	32.10%	181.02%	45.22%	256.01%	15.07%
	NDGC10	25.16%	12.45%	21.94%	74.68%	26.31%	142.58%	36.21%	200.44%	16.06%
	NDGC25	13.35%	6.94%	14.81%	44.74%	15.27%	92.50%	20.44%	148.39%	19.69%
	NDGC50	11.33%	6.09%	13.16%	39.64%	14.45%	76.86%	19.36%	127.70%	19.47%

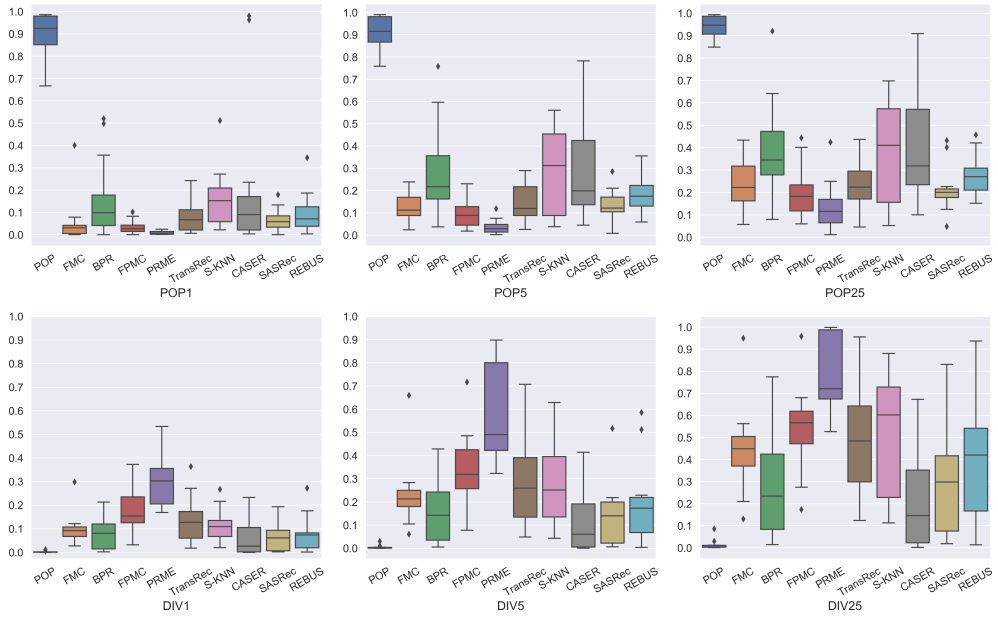

Figure 3.2: Performance of POP_X (first row) and DIV_X (second row) with $X \in \{1, 5, 25\}$ for each model. Values are averaged over all datasets.

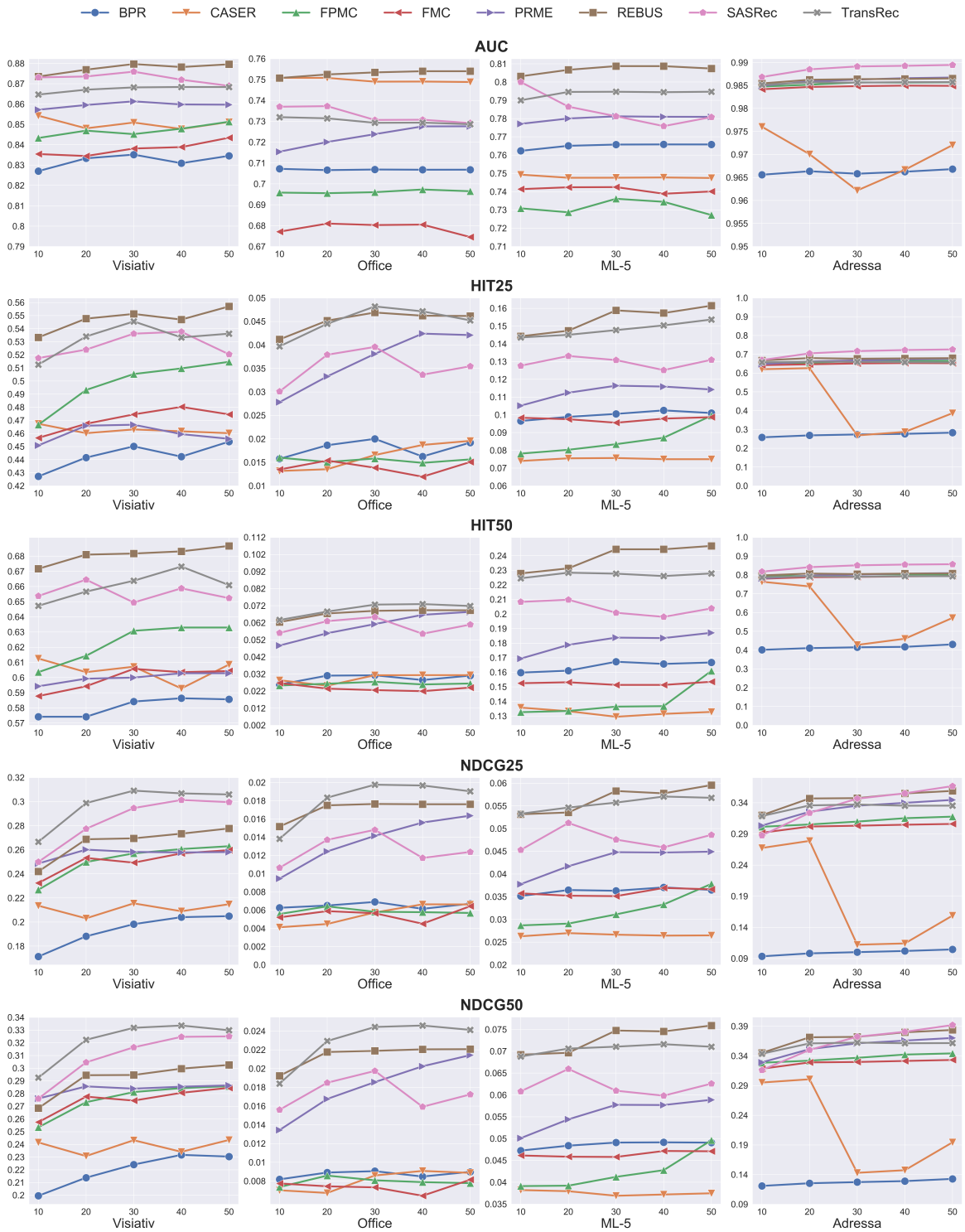

Figure 3.3: Influence on AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 (in ordinate) of the dimension k (in abscissa) of the latent vectors ($k = \{10, 20, 30, 40, 50\}$).

3.4.5 Cold-start user study

We now investigate how **REBUS** behaves when dealing with cold-start users, i.e., users whose sequence of actions is too short to take benefit from. We compare **REBUS** to solely 4 methods – POP, FMC, SASRec and S-KNN – since the other ones are not applicable for cold-start user recommendation. Only models that just embed items can make recommendations in this context. To evaluate the performance of **REBUS** and its competitors, we have chosen 7 datasets: Epinion, Foursquare, Adressa, Visiativ, Amazon-Automotive, Amazon-Office and Amazon-VideoGames. We then split each dataset into two parts:

1. One that contains users and items with at least 5 interactions. This is the same filter we used previously since we want to train the model with the same configuration and have the same hyperparameters;

2. A second one that contains cold-start users, that is to say users who do not appear in the first part but who interact with items that appear in the first part. Here, the users have a short history (between 1 and 4 items). This part is not used to train the models but only to evaluate their performance on the cold-start users.

For each dataset, the first parts are used as done in the previous studies. The second parts are split as follows:

1. The most recent item is used for the test;

2. The other items of the user sequence are used to predict the test item.

The performances of the different methods on every dataset are reported in Table 3.7 [10]. We can observe that **REBUS** outperforms other models except S-KNN on most of the datasets, having the best AUC and HIT_50 values. S-KNN returns the best score for HIT_25 and NDCG_X. This can be explained by the fact that user sequences in this testbed are similar to those used in session-based recommendations (short sequences). In this context, S-KNN is well adapted for cold-start users. We report in Figure 3.4 some additional indicators (i.e., POP_X and DIV_X) of the recommendations made for cold-start users. Results are similar to those obtained in Figure 3.2. S-KNN provides more diverse recommendations than other models. This may explain it is the best model for HIT_25 and NDCG_X. This study demonstrates that **REBUS** outperforms other models for the cold-start user problem for some performance metrics (i.e., AUC and HIT_50). For others metrics, **REBUS** is outperformed by only S-KNN. These good performances are mainly due to the embedding we use to model the data which is applicable on short user sequences.

[10] We only show AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 in the Table. HIT_5, HIT_10, NDCG_5 and NDCG_10 are shown in Appendix 7 Table 7.7.

Table 3.7: AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 for the different models that do not suffer of the problem of cold-start users. The 3 last rows, called *Improv. vs Best*, *Improv. vs SASRec* and *Improv. vs S-KNN*, shows the improvement in percentage of our method compared to the best model, SASRec and S-KNN (best obtained results are in bold and best obtained results for concurrent models are underlined).

	Metric	Epinions	Foursq	Adressa	Visiativ	Auto	Office	Games	Avg
POP	AUC	0.5741	0.8862	0.9689	0.7810	0.6400	0.6933	0.7702	0.7591
	HIT25	3.62%	41.38%	43.41%	31.21%	2.78%	0.46%	3.98%	18.12%
	HIT50	5.41%	51.93%	59.32%	46.08%	4.06%	1.69%	5.89%	24.91%
	NDGC25	1.41%	15.82%	18.55%	12.98%	1.09%	0.12%	1.55%	7.36%
	NDGC50	1.76%	17.88%	21.59%	15.81%	1.34%	0.36%	1.91%	8.66%
FMC	AUC	0.5829	0.9326	0.9776	0.8316	0.6644	0.7020	0.8529	0.7920
	HIT25	2.28%	48.89%	59.37%	45.54%	3.32%	1.32%	10.88%	24.51%
	HIT50	3.61%	60.15%	72.33%	57.97%	4.84%	2.72%	15.63%	31.04%
	NDGC25	0.91%	22.98%	27.35%	23.84%	1.50%	0.54%	4.27%	11.63%
	NDGC50	1.16%	25.16%	29.84%	26.22%	1.79%	0.81%	5.19%	12.88%
S-KNN	AUC	0.0495	0.7012	0.9515	0.8048	0.1138	0.1297	0.4103	0.4515
	HIT25	3.09%	**54.48%**	58.82%	51.01%	**6.91%**	**7.20%**	**14.83%**	**28.05%**
	HIT50	3.90%	61.65%	70.61%	61.18%	**8.21%**	**8.62%**	**18.93%**	33.30%
	NDGC25	**1.74%**	**34.67%**	**34.60%**	**27.89%**	**3.68%**	**4.15%**	**6.91%**	**16.24%**
	NDGC50	1.90%	**36.06%**	**36.87%**	**29.84%**	**3.93%**	**4.43%**	**7.70%**	**17.25%**
SASRec	AUC	**0.6504**	0.9407	0.9806	**0.8655**	0.6893	**0.7340**	0.8698	0.8186
	HIT25	3.36%	47.57%	61.29%	49.11%	2.85%	2.96%	9.26%	25.20%
	HIT50	5.07%	58.44%	74.98%	61.59%	4.20%	4.84%	14.12%	31.89%
	NDGC25	1.29%	23.12%	29.14%	24.56%	1.12%	1.05%	3.66%	11.99%
	NDGC50	1.61%	25.22%	31.77%	26.97%	1.38%	1.41%	4.60%	13.28%
REBUS	AUC	0.6253	**0.9537**	**0.9815**	0.8612	**0.7267**	0.7309	**0.8747**	**0.8220**
	HIT25	**3.94%**	53.75%	**63.42%**	**51.61%**	4.57%	5.09%	10.46%	27.55%
	HIT50	**6.01%**	**64.47%**	**75.65%**	**62.78%**	6.76%	7.65%	15.88%	**34.17%**
	NDGC25	1.58%	30.21%	33.22%	26.23%	1.79%	2.07%	4.04%	14.16%
	NDGC50	**1.97%**	32.28%	35.58%	28.39%	2.21%	2.56%	5.08%	15.44%
Rank of REBUS	AUC	2	1	1	2	1	2	1	1.43
	HIT25	1	2	1	1	2	2	3	1.71
	HIT50	1	1	1	1	2	2	2	1.43
	NDGC25	2	2	2	2	2	2	3	2.14
	NDGC50	1	2	2	2	2	2	3	2.00
Improv. VS Best	AUC	-3.86%	1.38%	0.09%	-0.50%	5.43%	-0.42%	0.56%	0.42%
	HIT25	8.84%	-1.34%	3.48%	1.18%	-33.86%	-29.31%	-29.47%	-1.78%
	HIT50	11.09%	4.57%	0.89%	1.93%	-17.66%	-11.25%	-16.11%	2.61%
	NDGC25	-9.20%	-12.86%	-3.99%	-5.95%	-51.36%	-50.12%	-41.53%	-12.81%
	NDGC50	3.68%	-10.48%	-3.50%	-4.86%	-43.77%	-42.21%	-34.03%	-10.49%
Improv. VS SASRec	AUC	-3.86%	1.38%	0.09%	-0.50%	5.43%	-0.42%	0.56%	0.42%
	HIT25	17.26%	12.99%	3.48%	5.09%	60.35%	71.96%	12.96%	9.33%
	HIT50	18.54%	10.32%	0.89%	1.93%	60.95%	58.06%	12.46%	7.15%
	NDGC25	22.48%	30.67%	14.00%	6.80%	59.82%	97.14%	10.38%	18.10%
	NDGC50	22.36%	27.99%	11.99%	5.27%	60.14%	81.56%	10.43%	16.27%
Improv. VS S-KNN	AUC	1163%	36.01%	3.15%	7.01%	538.58%	463.53%	113.19%	82.06%
	HIT25	27.51%	-1.34%	7.82%	1.18%	-33.86%	-29.31%	-29.47%	-1.78%
	HIT50	54.10%	4.57%	7.14%	2.62%	-17.66%	-11.25%	-16.11%	2.61%
	NDGC25	-9.20%	-12.86%	-3.99%	-5.95%	-51.36%	-50.12%	-41.53%	-12.81%
	NDGC50	3.68%	-10.48%	-3.50%	-4.86%	-43.77%	-42.21%	-34.03%	-10.49%

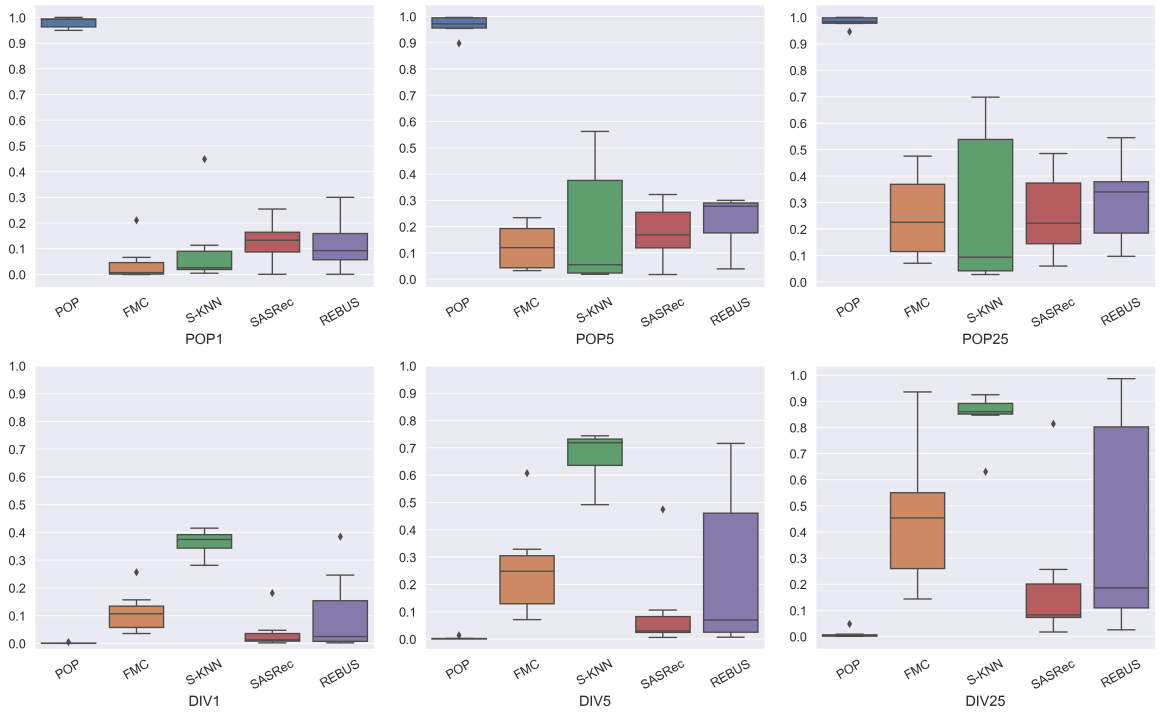

Figure 3.4: Performance of POP_X (first row) and DIV_X (second row) with $X \in \{1, 5, 25\}$ for the different models that do not suffer of the problem of cold-start users. Values are averaged over all datasets.

3.4.6 Study of the impact of user preferences and sequential dynamics in the recommendation

We have demonstrated that **REBUS** outperforms the state-of-the-art algorithms in most of the configurations (i.e., datasets and metrics). We now study how our model effectively behave. Especially, we investigate the impact of both user preferences and sequential dynamics on recommendation. To this end, we consider independently these two components. Furthermore, we also examine fixed-order Markov chains in our model instead of frequent sequences to capture the sequential dynamics. **REBUS_UP** refers to the long term part, **REBUS_SD** refers to the short term part using personalized context via frequent sequences, XMC means that the model uses Markov chains with a fixed-order X. The performances of the different configurations on each dataset are reported in Tables 3.8 and 3.9[11].

As expected, we can observe that the configurations of our model that combine user preferences and sequential dynamics (i.e. **REBUS** and **REBUS**$_{XMC}$) outperform other configurations – that only model either user preferences or the sequential dynamics – (1) for all metrics on sparse datasets and (2) only for AUC on dense datasets. Indeed, surprisingly, **REBUS_SD**$_{XMC}$ (when $X > 1$) outperforms on dense datasets all models tested in our experimental study, SASRec included, for HIT_X and NDGC_X (see Table 3.5). Despite the fact that **REBUS_SD**$_{XMC}$ (when $X > 1$) outperforms all non-combining configurations

[11] We only show AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 in these tables. HIT_5, HIT_10, NDCG_5 and NDCG_10 are reported in Appendix 7 Tables 7.8 and 7.9.

Table 3.8: AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 for **REBUS** and his variants on Epinions, Foursquare, Adressa, Visiativ, Amazon-Automotive, Amazon-Office-Product and Amazon-Video-Games datasets (best obtained results are in bold). The last column is the average performance on these 7 datasets, the average of all datasets is included in Table 3.9.

	Metric	Epinions	Foursq	Adressa	Visiativ	Auto	Office	Games	Avg
REBUS_UP	AUC	0.6495	0.9661	0.9828	0.8521	0.7147	0.7484	0.8860	0.8285
	HIT25	3.21%	61.16%	63.48%	43.80%	4.08%	3.88%	8.83%	26.92%
	HIT50	5.00%	69.20%	76.93%	59.64%	6.02%	6.06%	14.14%	33.86%
	NDGC25	1.27%	42.07%	31.80%	18.29%	1.62%	1.53%	3.28%	14.26%
	NDGC50	1.61%	43.62%	34.40%	21.34%	1.99%	1.94%	4.30%	15.60%
REBUS_SD$_{1MC}$	AUC	0.6257	0.9624	0.9846	0.8591	0.6855	0.7184	0.8710	0.8153
	HIT25	2.65%	55.12%	64.26%	50.97%	3.80%	4.08%	9.16%	27.15%
	HIT50	3.84%	65.09%	77.89%	63.58%	5.55%	6.74%	13.74%	33.78%
	NDGC25	1.19%	28.93%	30.23%	**27.11%**	1.47%	1.52%	**3.59%**	13.44%
	NDGC50	1.42%	30.85%	32.87%	**29.55%**	1.81%	2.03%	4.47%	14.71%
REBUS_SD$_{2MC}$	AUC	0.6428	0.9660	0.9852	0.8720	0.7132	0.7440	0.8872	0.8301
	HIT25	3.00%	55.26%	64.75%	**53.55%**	4.11%	4.40%	9.01%	27.73%
	HIT50	5.30%	65.07%	78.66%	66.45%	5.96%	6.93%	13.95%	34.62%
	NDGC25	1.29%	30.11%	30.67%	25.82%	1.61%	1.59%	3.47%	13.51%
	NDGC50	1.73%	32.01%	33.36%	28.31%	1.96%	2.08%	4.42%	14.84%
REBUS_SD$_{3MC}$	AUC	0.6485	0.9663	0.9850	0.8725	**0.7192**	0.7536	0.8892	0.8335
	HIT25	3.35%	55.21%	64.56%	52.90%	4.15%	**4.60%**	8.68%	27.64%
	HIT50	5.30%	64.92%	78.17%	66.31%	6.15%	**7.31%**	13.57%	34.53%
	NDGC25	1.34%	30.91%	30.36%	24.23%	1.60%	**1.66%**	3.36%	13.35%
	NDGC50	1.71%	32.79%	32.99%	26.81%	1.99%	**2.18%**	4.30%	14.68%
REBUS_SD	AUC	0.6114	0.9653	0.9849	0.8610	0.6904	0.7232	0.8796	0.8166
	HIT25	3.02%	55.37%	64.78%	51.25%	3.89%	4.11%	9.09%	27.36%
	HIT50	4.25%	65.20%	78.41%	63.80%	5.84%	6.69%	13.78%	34.00%
	NDGC25	1.20%	30.72%	30.47%	26.35%	1.51%	1.49%	3.53%	13.61%
	NDGC50	1.43%	32.62%	33.10%	28.78%	1.89%	1.99%	4.43%	14.89%
REBUS$_{1MC}$	AUC	0.6584	**0.9682**	0.9854	0.8751	0.7190	**0.7542**	0.8935	0.8363
	HIT25	**3.67%**	**63.21%**	66.45%	53.48%	4.11%	4.27%	9.28%	**29.21%**
	HIT50	5.18%	**70.81%**	79.71%	**67.60%**	5.99%	6.44%	**14.73%**	**35.78%**
	NDGC25	**1.46%**	**46.16%**	31.77%	24.59%	1.62%	1.55%	3.53%	**15.81%**
	NDGC50	**1.75%**	**47.63%**	34.33%	27.31%	1.99%	1.97%	**4.57%**	**17.08%**
REBUS$_{2MC}$	AUC	**0.6660**	0.9676	**0.9855**	**0.8754**	0.7189	0.7537	**0.8936**	**0.8372**
	HIT25	3.25%	62.76%	66.61%	52.40%	4.16%	4.15%	9.17%	28.93%
	HIT50	**5.56%**	70.26%	79.68%	66.16%	6.07%	6.51%	14.55%	35.54%
	NDGC25	1.31%	45.92%	31.92%	23.58%	1.62%	1.60%	3.46%	15.63%
	NDGC50	**1.75%**	47.37%	34.44%	26.21%	1.99%	2.05%	4.49%	16.90%
REBUS$_{3MC}$	AUC	0.6624	0.9669	0.9849	0.8729	0.7179	0.7528	0.8923	0.8357
	HIT25	3.49%	62.92%	64.60%	50.39%	4.13%	4.19%	8.82%	28.36%
	HIT50	5.44%	70.24%	78.53%	64.95%	6.01%	6.32%	14.06%	35.08%
	NDGC25	1.35%	46.14%	30.36%	22.57%	1.62%	1.61%	3.31%	15.28%
	NDGC50	1.73%	47.55%	33.05%	25.36%	1.98%	2.02%	4.32%	16.57%
REBUS	AUC	0.6524	0.9677	0.9854	0.8735	0.7184	0.7507	0.8934	0.8345
	HIT25	3.39%	62.77%	**66.73%**	53.33%	**4.24%**	4.12%	**9.30%**	29.13%
	HIT50	5.32%	70.40%	**79.77%**	67.17%	**6.20%**	6.24%	14.61%	35.67%
	NDGC25	1.31%	46.04%	**32.00%**	24.20%	**1.68%**	1.52%	3.52%	15.75%
	NDGC50	1.68%	47.52%	**34.52%**	26.86%	**2.05%**	1.92%	4.54%	17.01%

Table 3.9: AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 for **REBUS** and his variants on ML-5, ML-10, ML-20, ML-30 and ML-50 datasets (best obtained results are in bold). The last column is the average performance on the 12 datasets, including datasets of Table 3.8.

	Metric	ML-5	ML-10	ML-20	ML-30	ML-50	Avg(ML)	Avg(All)
REBUS_UP	AUC	0.7941	0.8428	0.8603	0.8631	0.8671	0.8455	0.8356
	HIT25	12.95%	12.49%	11.97%	11.61%	11.39%	12.08%	20.74%
	HIT50	21.51%	20.72%	19.99%	19.47%	19.49%	20.24%	28.18%
	NDGC25	4.80%	4.40%	4.10%	4.06%	3.88%	4.25%	10.09%
	NDGC50	6.43%	5.97%	5.64%	5.56%	5.43%	5.81%	11.52%
REBUS_SD$_{1MC}$	AUC	0.7848	0.8207	0.8556	0.8657	0.8725	0.8399	0.8255
	HIT25	13.45%	15.66%	17.83%	18.08%	18.20%	16.65%	22.77%
	HIT50	20.83%	23.22%	26.38%	27.14%	27.39%	24.99%	30.12%
	NDGC25	5.10%	6.18%	6.74%	6.99%	7.03%	6.41%	10.51%
	NDGC50	6.52%	7.63%	8.39%	8.73%	8.80%	8.01%	11.92%
REBUS_SD$_{2MC}$	AUC	0.7954	0.8468	0.8718	0.8773	0.8850	0.8553	0.8406
	HIT25	14.06%	**18.15%**	19.32%	**19.71%**	**19.75%**	**18.20%**	**23.76%**
	HIT50	22.07%	26.89%	29.03%	**30.02%**	30.29%	27.66%	31.72%
	NDGC25	5.17%	**6.75%**	7.28%	**7.74%**	**7.49%**	**6.88%**	10.75%
	NDGC50	6.71%	8.43%	9.14%	**9.71%**	**9.50%**	**8.70%**	12.28%
REBUS_SD$_{3MC}$	AUC	0.7969	0.8531	0.8763	0.8804	0.8879	0.8589	0.8441
	HIT25	13.78%	17.34%	**19.56%**	19.44%	19.47%	17.92%	23.59%
	HIT50	21.87%	**27.57%**	**29.92%**	29.86%	**30.44%**	**27.93%**	**31.78%**
	NDGC25	5.03%	6.48%	**7.35%**	7.51%	7.28%	6.73%	10.59%
	NDGC50	6.58%	**8.44%**	**9.34%**	9.51%	9.39%	8.65%	12.17%
REBUS_SD	AUC	0.7758	0.8282	0.8572	0.8677	0.8756	0.8409	0.8267
	HIT25	13.96%	16.05%	17.50%	18.26%	18.43%	16.84%	22.98%
	HIT50	21.29%	23.88%	26.73%	27.77%	27.79%	25.49%	30.45%
	NDGC25	5.16%	6.13%	6.49%	7.03%	6.83%	6.33%	10.58%
	NDGC50	6.57%	7.64%	8.26%	8.86%	8.62%	7.99%	12.02%
REBUS$_{1MC}$	AUC	0.8023	0.8555	0.8770	0.8829	0.8883	0.8612	0.8466
	HIT25	**14.82%**	15.17%	15.80%	16.58%	16.87%	15.85%	23.64%
	HIT50	22.67%	24.66%	26.61%	27.21%	27.34%	25.70%	31.58%
	NDGC25	**5.44%**	5.48%	5.68%	6.08%	6.21%	5.78%	**11.63%**
	NDGC50	**6.95%**	7.30%	7.76%	8.12%	8.21%	7.67%	**13.16%**
REBUS$_{2MC}$	AUC	0.8012	0.8557	0.8770	**0.8837**	**0.8905**	**0.8616**	**0.8474**
	HIT25	14.46%	14.85%	15.35%	16.48%	15.91%	15.41%	23.30%
	HIT50	22.31%	24.82%	25.53%	27.17%	26.78%	25.32%	31.28%
	NDGC25	5.24%	5.38%	5.45%	6.04%	5.83%	5.59%	11.45%
	NDGC50	6.74%	7.29%	7.40%	8.09%	7.92%	7.49%	12.98%
REBUS$_{3MC}$	AUC	0.8000	0.8538	0.8764	0.8825	0.8894	0.8604	0.8460
	HIT25	14.03%	14.41%	14.82%	15.78%	15.63%	14.93%	22.77%
	HIT50	21.96%	24.06%	25.30%	26.31%	25.62%	24.65%	30.73%
	NDGC25	5.08%	5.25%	5.29%	5.67%	5.53%	5.36%	11.15%
	NDGC50	6.61%	7.10%	7.29%	7.69%	7.45%	7.23%	12.68%
REBUS	AUC	**0.8031**	**0.8563**	**0.8772**	0.8826	0.8884	0.8615	0.8458
	HIT25	14.42%	15.27%	15.60%	15.80%	16.72%	15.56%	23.48%
	HIT50	**22.79%**	25.50%	26.01%	26.21%	26.91%	25.48%	31.43%
	NDGC25	5.32%	5.59%	5.60%	5.80%	6.03%	5.66%	11.55%
	NDGC50	6.92%	7.55%	7.59%	7.80%	7.98%	7.57%	13.08%

(that is to say **REBUS_SD**$_{1MC}$, **REBUS_SD** and **REBUS_UP**), there is no improvement when XMC is used in **REBUS** (i.e. **REBUS**$_{XMC}$ with $X > 1$). In overall, **REBUS_SD** provides better performances than **REBUS_SD**$_{1MC}$, which proves that there is an interest in using personalized sequences to model sequential dynamics. To conclude, the most consistent configurations are **REBUS**$_{1MC}$ and **REBUS**.

3.4.7 Study of the used user preferences window

We study the effect of the hyperparameter MAX_LENGTH. It allows **REBUS** to be more flexible by controlling the temporal window that influences the user's preferences and to get rid of the old past: MAX_LENGTH < 15 means that only the recent actions of the user are considered, whereas greater values of MAX_LENGTH enable older past actions to impact the representation of the user preferences. As MAX_LENGTH only affects the user preferences, it also influences **REBUS**$_{XMC}$.

In Figure 3.5, we analyze the impact of MAX_LENGTH (MAX_LENGTH $\in \{5, 10, 15, 25, 35, 50, 75, 100, 200\}$) on AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50. We can observe different cases:

1. In most of the cases, all metrics follow nearly the same behavior. There are some exceptions, for instance on ML-50 dataset when $5 \leqslant$ MAX_LENGTH $\leqslant 15$ the AUC increases while the HIT_25, HIT_50, NDCG_25 and NDCG_50 decrease. However, when MAX_LENGTH > 15, all metrics follow nearly the same behavior;

2. The best performance appears when **REBUS** consider high values of MAX_LENGTH (MAX_LENGTH > 30) (e.g., Adressa dataset, $\gamma = 0.3$) ;

3. The best performance appears when **REBUS** consider low values of MAX_LENGTH (MAX_LENGTH $\leqslant 15$) (e.g., ML-50 dataset) ;

4. There is no real trend that appears when we change MAX_LENGTH (e.g., Office and Visiativ dataset).

Also, the green ellipse in Figure 3.5 is the fixed MAX_LENGTH used for **REBUS** in Section 3.4.4 but we can see that if we change MAX_LENGTH for **REBUS** and **REBUS**$_{XMC}$ we can sometimes obtain better performance on AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 than those reported on Tables 3.4 and 3.5.

We can conclude that the choice of MAX_LENGTH has an influence on the performance of **REBUS**. With only few tests (4 or 5 trials), a solution close to the best one can be quickly found.

3.4.8 Study of the used personalized sequences

We study the characteristics of the sequences used by **REBUS** for recommendation. The first five columns (from (A) to (E)) of Table 3.10 [12] report the percentage of sequences $m_{s_u^{[1,t[}}$ that are equivalent to: (A) no matched items, (B) a first order Markov chain. Column (A) and (B) are both equivalent to a first order Markov chain but we split it into two different

[12]The two numbers after the name of the dataset are respectively the value of MINCOUNT and L.

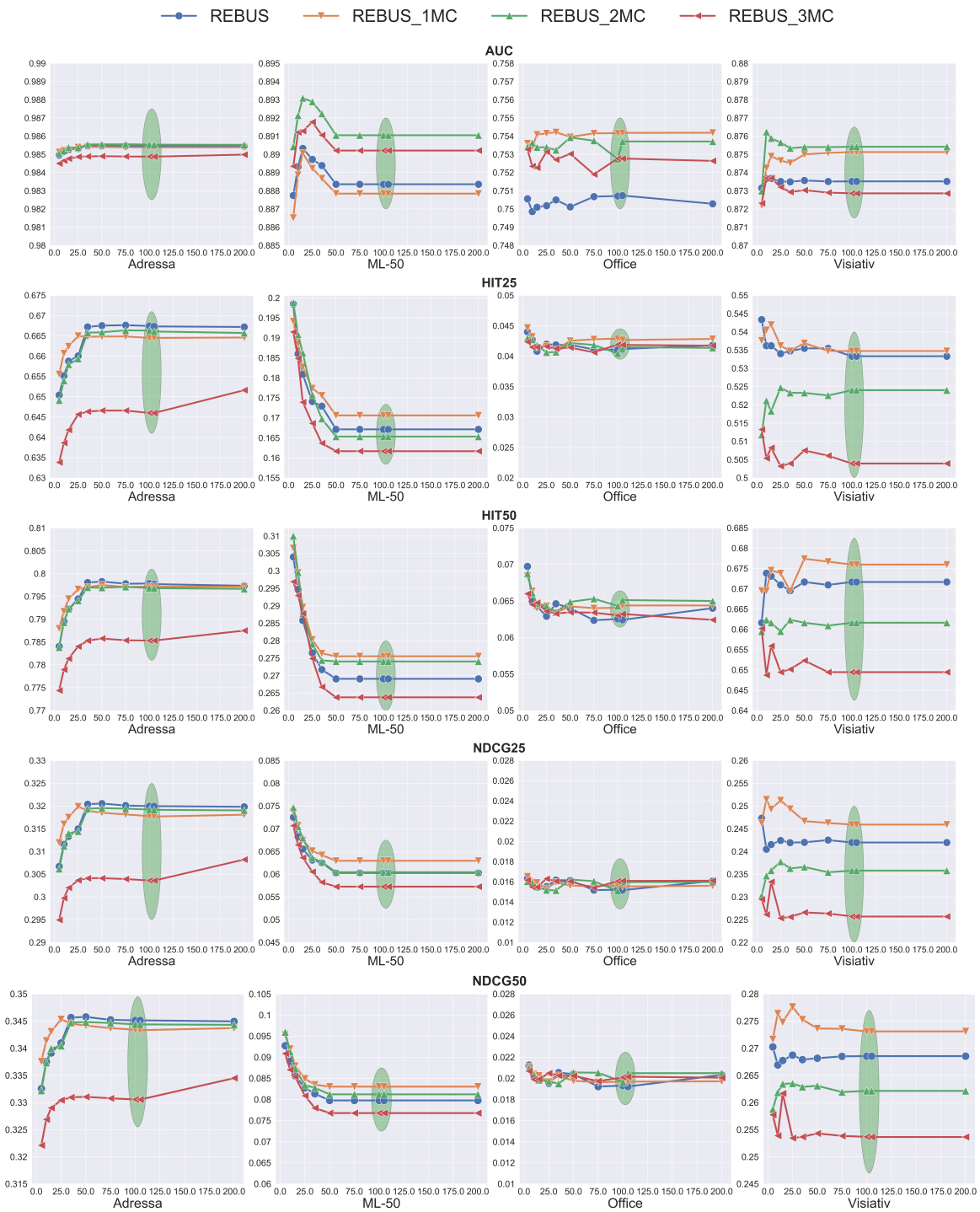

Figure 3.5: Influence on AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 (in ordinate) of MAX_LENGTH (in abscissa)for **REBUS** $\mathbf{REBUS}_{1MC}$, $\mathbf{REBUS}_{2MC}$ and $\mathbf{REBUS}_{3MC}$. The green ellipse is the fixed MAX_LENGTH use in 3.4.4

columns to point out the fact that it is very uncommon that no item from F matches $s_u^{[1,t[}$, (C) a first order Markov chain but not with the most recent item (D) a L order Markov chain ($L > 1$). In column (E) we report the percentage of sequences $m_{s_u^{[1,t[}}$ that cannot be modeled by Markov chains (sequences that are not strings). The column (A) confirms that it is very uncommon that none of the sequences of F match the sequence $s_u^{[1,t[}$. **REBUS** often takes into account the most recent item within the user's sequence, but there are some cases for which older items are used. This cannot be obtained by first order Markov chains. Furthermore, depending on the dataset, between 10% to 40% (Columns (C) + (E)) of the sequences are different from Markov Chains. The added value of our approach can be assessed by (C), (D) and (E).

The two last columns[13] of Table 3.10 reports the mean of $m_{s_u^{[1,t[}}$ sizes (F) and the mean of their occupation in s_u (G), i.e., difference between the last position and the first position in $m_{s_u^{[1,t[}}$ (here we count the wildcards). In most cases, short sequences are exploited to model the sequential dynamics. Nevertheless, we have observed that there are different sizes ranging from 1 to 15. The mean of the occupation of the sequences is rather low. **REBUS** often uses *compact* sequences. This can explain the good results of short Markov chains to capture sequential dynamics in **REBUS**. We can also observe that **REBUS** sometimes uses non-consecutive items with an arbitrary number of spaces between items. Such sequences cannot be obtained by fixed order Markov chains. This is why frequent sequences outperforms Markov chain when focusing only on the sequential dynamics (see Table 3.8).

Table 3.10: The first five columns show the percentage of substrings $m_{s_u^{[1,t[}}$ that are equivalent to: (A) no matched items, (B) a first order Markov chain, (C) a first order Markov chain but not with the most recent item, (D) a L order Markov chain ($L > 1$). (E) shows the percentage of substrings $m_{s_u^{[1,t[}}$ that cannot be modeled by Markov chains (substrings that are mapped in a non-consecutive way on the user history). The two last columns show the mean of $m_{s_u^{[1,t[}}$ sizes (F) and the mean of their occupation length (here the wildcards are taken into account) in the user sequence (G).

Dataset	**(A)** no match	**(B)** MC_1	**(C)** MC_1_old	**(D)** MC_L	**(E)** Seq.	**(F)** Size	**(G)** Occup.
Epinions_2_3	3.39%	62.21%	34.06%	0.16%	0.18%	1.01	1.01
Foursquare_2_15	0.06%	26.15%	1.39%	43.85%	28.54%	2.36	2.93
Adressa_2_2	0.00%	1.66%	0.02%	93.05%	5.26%	1.98	2.07
Visiativ_2_10	0.00%	40.27%	0.21%	28.97%	30.54%	1.71	2.91
Ama-Auto_2_8	2.12%	72.82%	22.48%	1.31%	1.27%	1.03	1.04
Ama-Office_2_8	1.19%	72.86%	20.79%	2.69%	2.48%	1.05	1.10
Ama-Games_2_10	0.06%	80.74%	5.57%	6.13%	7.50%	1.14	1.38
ML-5_2_3	0.00%	87.05%	7.50%	3.08 %	2.37%	1.06	1.08
ML-10_2_8	0.00%	79.35%	1.94%	8.06 %	10.65%	1.20	1.48
ML-20_2_8	0.00%	59.11%	0.81%	13.41%	26.67%	1.42	2.90
ML-30_2_10	0.00%	47.37%	0.51%	16.71%	35.41%	1.55	4.21
ML-50_2_10	0.00%	35.33%	0.17%	20.94%	43.56%	1.68	5.96

[13]Sequences that do not match any substring of F are omitted for columns (F) and (G).

3.4.9 Relative importance of user preferences on sequential dynamics

The user preferences modeling part is very important to provide accurate recommendations. Models that only focus on user preferences (e.g., BPR-MF, **REBUS_UP**) outperform those that only consider the sequential dynamics (e.g., FMC, **REBUS_SD**, **REBUS_SD**$_{1MC}$) on the AUC metric. However, there are also models that only focus on sequential dynamics and outperform those that only consider the user preferences on HIT_X or NDGC_X. The trade off between user preferences and sequential dynamics is very important to make **REBUS** the most robust and accurate as possible for all kind of datasets and metrics. The γ hyperparameter makes it possible to weight the importance given to each part of the model: $\gamma = 0.5$ means that the two parts have the same importance, whereas the greater the γ, the greater the importance of the user preferences in the recommendations. The best values of γ determined by grid search ($\gamma \in \{0.0, 0.1, \ldots, 1.0\}$) are reported in Appendix 7 Table 7.1. For most of datasets, γ is between 0.3 and 0.7. This means that both user preferences and sequential dynamics are important, but one may dominate the other to get accurate recommendations.

In Figure 3.6, we analyze the impact of γ on AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50. We can observe different cases:

1. γ does not really have an impact on AUC performance of **REBUS** (e.g., Adressa dataset);

2. The best performance on AUC appears when **REBUS** focused on the sequential dynamics (e.g., ML-50 dataset, $\gamma = 0.3$);

3. The best performance on AUC appears when **REBUS** is focus on the user preferences (e.g., Office dataset, $\gamma = 0.7$);

4. The best performance on AUC is reached by **REBUS** when the user preferences and the sequential dynamics have the same importance (e.g., Visiativ).

Notice that for Office dataset, HIT_25, HIT_50, NDCG_25 and NDCG_50 do not follow the same behavior as AUC: a greater γ increases the AUC performance but decreases the HIT_25, HIT_50, NDCG_25 and NDCG_50 performances.

Finally, the green ellipse in Figure 3.6 is the best γ identified for **REBUS** in Section 3.4.4 but we can see as in the previous experiment with MAX_LENGTH that if we change γ for **REBUS**$_{XMC}$ we can sometimes obtain better performance on AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 than those reported on Tables 3.4 and 3.5.

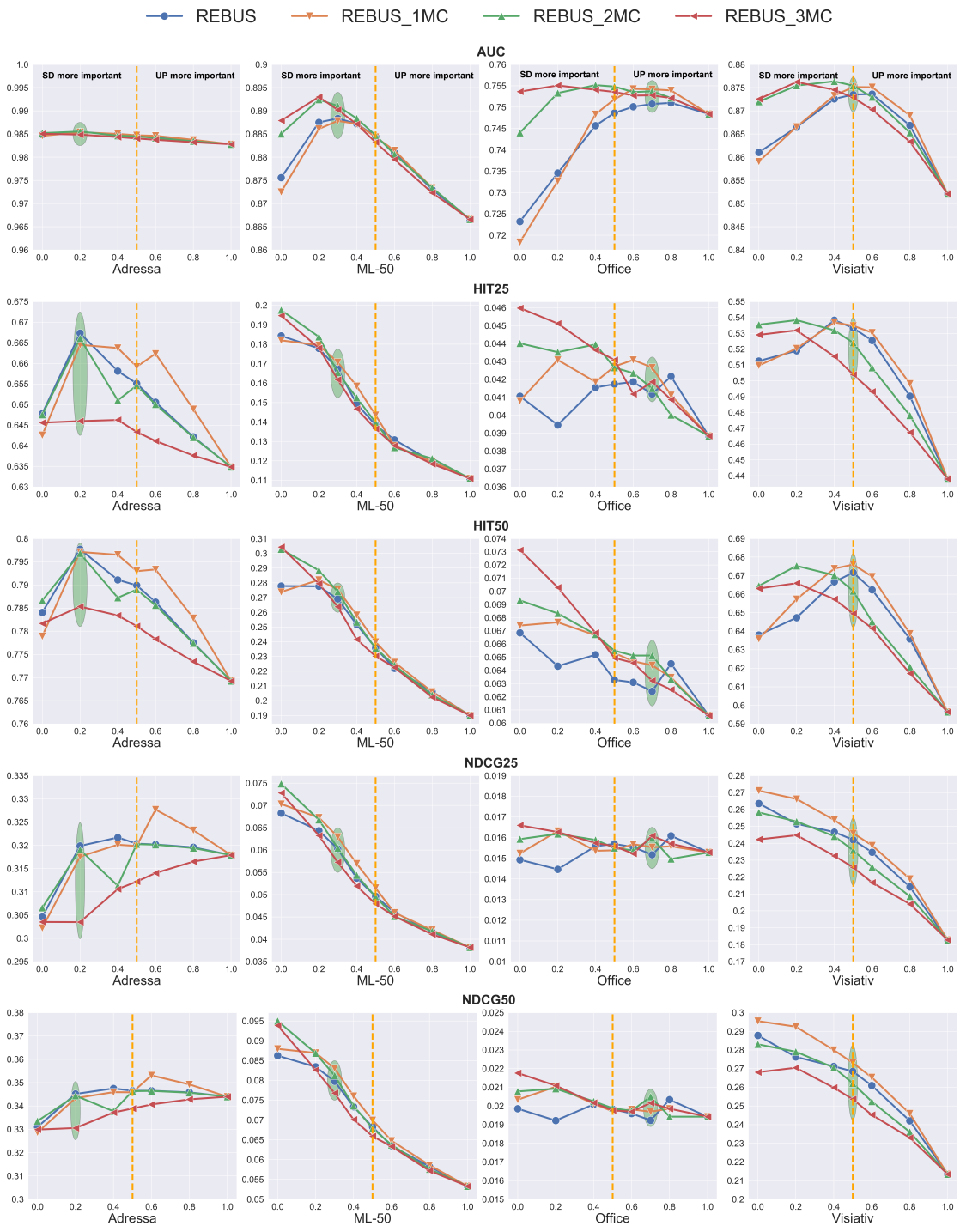

Figure 3.6: Influence on AUC, HIT_25, HIT_50, NDCG_25 and NDCG_50 (in ordinate) of γ (in abscissa) for **REBUS** **REBUS**$_{1MC}$, **REBUS**$_{2MC}$ and **REBUS**$_{3MC}$.The green ellipse is the best γ identified in 3.4.4.

3.4.10 Examples of recommendations

Figure 3.7 shows some recommendations made by **REBUS** on Amazon-Video Games. As can be seen on these examples, **REBUS** captures the sequential dynamics and recommends items which are similar to the ground truth. For instance, for the first user, **REBUS** recommends *Final Fantasy X-2* because the user sequentially bought the previous editions of the *Final Fantasy* games (the red squared items). The ground truth is a similar game: *The Legend of Dragoon*. On this example, the sequential dynamics is captured with a 4-item substring mapped with a discontinuity (a wildcard) on the user sequence. Some recommendations are based on smaller and more compact sequences, e.g., the two last recommendations use only the most recent item to capture the sequential dynamics. Based on the last user action – *Pokemon Mystery Dungeon on Nintendo 3DS* and *Zumba Fitness on Nintendo Wii* – **REBUS** respectively recommends *The Legend of Zelda: Ocarina of Time on Nintendo 3DS* (instead of *Nintendo 3DS XL*) and a *Nintendo Wii*.

These examples of recommendations illustrate how REBUS can provide insight to explain recommendations. Indeed, thanks to $m_{s_u^{[1,t[}}$ (The red squares in Figure 3.7) we know what influenced **REBUS** for the sequential dynamics. However, this is not enough to explain the whole recommendation. Indeed $m_{s_u^{[1,t[}}$ does not explain the user preferences part of the recommendation and the trade-off between user preferences and sequential dynamics.

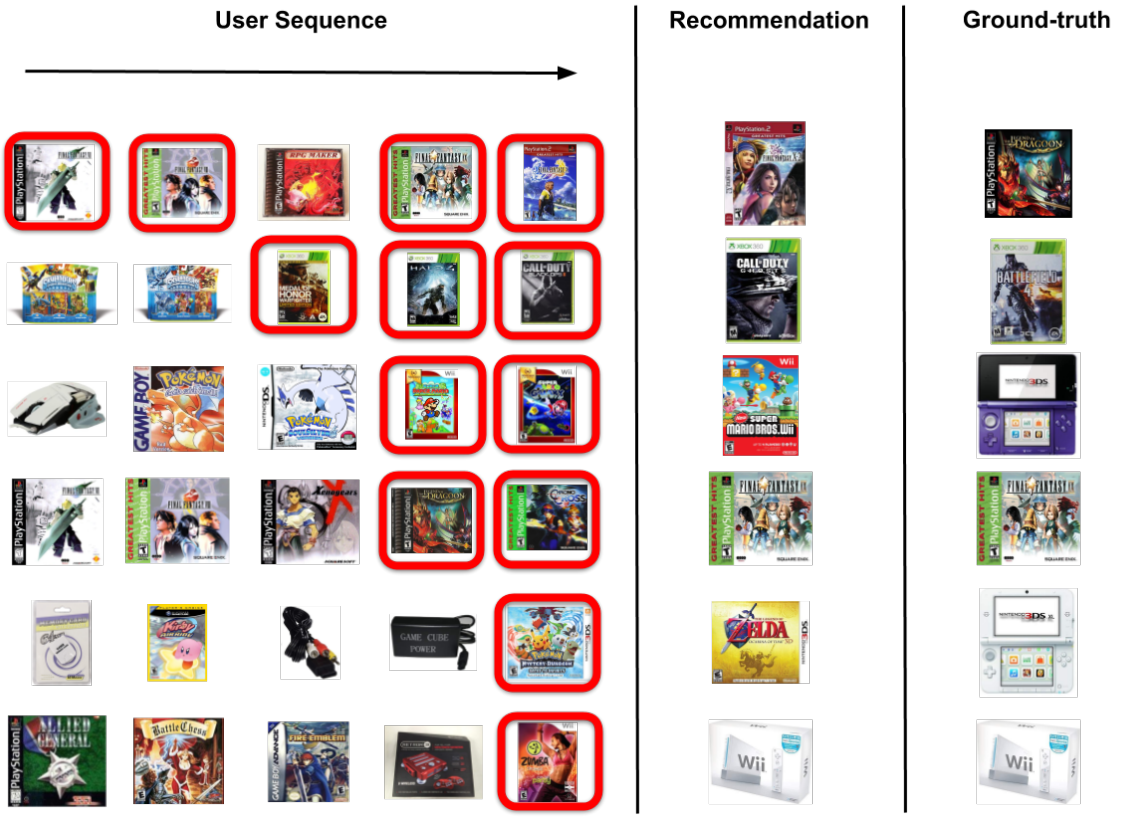

Figure 3.7: Recommendations for a random sample of users by **REBUS** on Amazon Video Games. The red squares are items in $m_{s_u^{[1,t[}}$.

3.5 Conclusion

In this chapter, we introduced **REBUS**, a new metric embedding model that only project items and addresses the problem of sequential recommendation. It uses frequent substrings as a proxy to learn conditional distributions of item appearance in the sequential history of the user in order to provide personalized order Markov chains and thus capture sequential dynamics. Our model learns a representation of the user preferences and the sequential dynamics in the same Euclidean space. We have carried out thorough experiments on multiple real-world datasets. These experiments demonstrate that our sequential recommendation model **REBUS** provides the most accurate recommendations for most of the datasets. Indeed **REBUS** outperforms other methods in sparse data. When considering dense data, our model remains competitive and we have seen that the performances of **REBUS** can even be increased with the use of shorter users' histories to model the sequential dynamics. For instance, **REBUS_SD** – that uses Markov chains of length 2 and the damping factor to model the sequential dynamics – outperforms the state-of-the-art models on dense data (e.g., ML-50 or ML-30). In addition, we can use the substrings that support the recommendations to understand the mechanisms at work in this process. This empirical study also gives evidence that **REBUS** has good performance with respect to the cold-start user problem, as it outperforms most of state-of-the-art models. Actually, the item embedding is powerful even if the history of the user is very short (e.g., 1 or 2 items). In Chapter 5, thanks to Visiativ, we have been able to implement and test the performance of **REBUS** in *myCADservices* platform.

This work opens up several avenues for further research such as the investigation of more sophisticated pattern mining techniques to both better model the user preferences and the sequential dynamics. Moreover an improvement could be to learned γ during the learning phase.

We have shown that **REBUS** can provide some insight to explain a recommendation based on the personalized sequences which capture the sequential dynamics. However, this is not enough to explain the whole recommendation in particular the user preferences part of the recommendation. In Chapter 4, we propose a method, based on subgroup discovery with different pattern languages (i. e., itemsets and sequences), that provides interpretable explanations of the recommendations for **REBUS** but also for other models. Such an approach is useful for comparing different models and explaining the rationale for recommendations.

Chapter 4

Explaining the Recommendation of any Model

Explainable AI has received a lot of attention over the past decade, with the proposal of many methods explaining black box classifiers such as neural networks. Despite the ubiquity of recommender systems in the digital world, only few research has attempted to explain their functioning, whereas it raises problems such as ethical issues. Indeed, recommender systems direct to a large extent user choices and their impact is important as they give access to only a small part of the range of items (e. g., products and/or services), as the submerged part of the iceberg. Consequently, they limit access to other resources. The potentially negative effects of these systems have been pointed out as phenomena like echo chambers and winner-take-all effects, because the internal logic of these systems is to likely enclose the consumer in a "déjà vu" loop. Therefore, it is crucial to provide explanations of such recommender systems and to identify the user data that led the system to make a specific recommendation. This makes it possible to evaluate recommender systems not only regarding their efficiency (i. e., their capability to recommend an item that was actually chosen by the user), but also w.r.t. the diversity, relevance and timeliness of the active data used to make the recommendation. In this chapter, we propose a deep analysis of 7 state-of-the-art models learnt on 9 datasets based on the identification of the items or the sequences of items actively used by the models. The proposed method, which is based on subgroup discovery with different pattern languages (i. e., itemsets and sequences), provides interpretable explanations of the recommendations – useful to compare different models and explain the reasons behind the recommendation to the user.

This chapter has been published In 2020 IEEE International Conference on Data Science and Advanced Analytics (DSAA) (Lonjarret et al., 2020). Also for reproducibility purposes, all the datasets, source code, and experimental results are made available in a repository[1].

4.1 Introduction

A recommender system offers personalized recommendations that are based on the history of users in the system and their respective resemblance to the histories of other users. Per-

[1] `https://bit.ly/2DD0H0f`

sonalizing a suggestion then consists of filtering the content in order to keep only the most relevant items for a given user. Since such processes are typically opaque, being able to explain a recommendation should increase the user's confidence and trust in the system (Pu and Chen, 2006). This question has already been considered, for classification problems (e. g., (Ribeiro et al., 2016)), but also in the scope of recommender systems, e. g., (Tintarev and Masthoff, 2011) to some extent. However, these approaches did not extend to a deep analysis of complex recommendation models nor to methods providing model-agnostic explanations. This is the task we tackle in this chapter, restricting our analysis to recommender systems that only use the user's history in the system (the users' sequences), without taking into account external information such as user profile data or item content (i.e. collaborative filtering model).

Recommender systems are used to retrieve information and products that are most relevant to a user. Existing techniques and methods are numerous and – even if we can understand them from a theoretical point of view – it is usually difficult to understand a particular recommendation. However, being able to identify the "foundations" of a recommendation – i. e., answering the "why" question using an *explanation* – would help to increase the confidence that one can have in it (Tintarev and Masthoff, 2011), and also to assess and compare different methods that may only differ slightly.

The term explanation has been widely investigated in different disciplines, such as cognitive science, artificial intelligence, linguistics, philosophy of science, etc. Essentially, explanations support human beings in their decision-making (Schank, 1986), in order to enhance understandability and transparency, and thus increasing trust in a system. This is also the way we understand explanation in the context of recommender systems. According to Tintarev and Masthoff (2007), we can distinguish between different goals of an explanation, i. e., transparency, validity, trustworthiness, persuasiveness, effectiveness, efficiency, satisfaction, relevance, comprehensibility and education. Here, we mainly relate to validity with respect to a recommendation and its effectiveness, i. e., helping a user to discover her preferences in order to increase the transparency and trustworthiness of a recommender system.

As we have seen in the previous chapters, it is generally accepted that a recommendation can be based on a "global" characterization of the user, which generally relates to her "user preferences" in the literature. It can also depend on recent user activity, and/or sequential relations between a set of items (like watching episodes of a series in order). In our thesis, we call this "sequential dynamics".

Thus, we propose a method for explaining recommender systems with the objective to answer the following questions, which we address in this chapter:

1. Which actions in the past are behind the recommendation?

2. Is the order of the actions (i. e., their sequence) important for the recommendation?

3. In the case that the order of actions is important, which sequences of actions do support the recommendation?

4. Furthermore, can we interpret a model globally by identifying a typology of possible recommendations?

5. Finally, how can we compare the "explanations" of several models?

The proposed method (see Figure 4.1 for an overview) exploits the neighborhood of the user's sequence to study the variability of the recommendation made by the model:

1. Given a user sequence s_u, it generates sequences of items close to s_u. Two different neighborhoods are considered:
   a) $N_1(s_u)$ consists of all subsets of the items contained in the user sequence, where the items are ordered as in the user sequence. With this neighborhood, the goal is to identify items that strongly impact the recommendation;
   b) The second neighborhood $N_2(s_u)$ focuses on the impact of the item order on the recommendation. To this end, $N_2(s_u)$ consists of perturbed sequences, resulting from the permutation of two consecutive items in the user sequence – a random number of times (i. e., given s_u an arbitrary number of permutations is applied).
2. The recommender model $\mathcal{R}$ is then used to compute the recommendations based on the new sequences. We then apply subgroup discovery (Atzmueller, 2015, Wrobel, 1997) to identify the past actions that play an important role in the recommendation of the initial item i^*. Subgroup discovery is part of the family of rule-based models which are widely accepted as the most interpretable ones (Fürnkranz et al., 2020). We then have two options:
   a) On the sequences of N_1, the subgroup discovery approach we propose identifies user actions that lead the model to recommend $i^\star$ in one of the top positions. To that end, we use an itemset description language and apply the SD-Map algorithm (Lemmerich et al., 2016);
   b) For the sequences N_2, we first check whether the order between past actions matters via a simple distributional test. We use sequences as the description language and the algorithm SeqScout (Mathonat et al., 2019) to identify the patterns that lead to the recommendation of $i^\star$ at the first position.
3. For both, the best subgroup allows us to identify conditions on the user's actions maximizing the score of i^*.

Our contributions are summarized as follows:

1. We present an approach for explaining recommendations of specific models of recommender systems exploiting the recommendation history and where a perturbation on the original user sequence has an effect on the provided recommendation. For instance, we cannot explain a matrix factorization approach, as **BPR-MF** Rendle et al. (2009), that only models user preferences because once the model has been trained a perturbation on the original user sequence will have no effect on the recommendation. This methodological approach for generating specific sets of recommendation histories is itself model-agnostic and thus broadly applicable.

2. Instantiating this approach, we present an explanation method utilizing subgroup discovery (Wrobel, 1997) for identifying active data used for recommendation.

3. Finally, we present experiments performing a deep analysis of 5 state-of-the-art models and our model **REBUS**, together with one of its variant **REBUS_UP** – known to only consider user preferences while others also integrate user dynamics in their decisions – learnt on 9 datasets, including Visiativ dataset. We specifically target this dataset for a case study, while we include other datasets to be used as benchmarks in order to

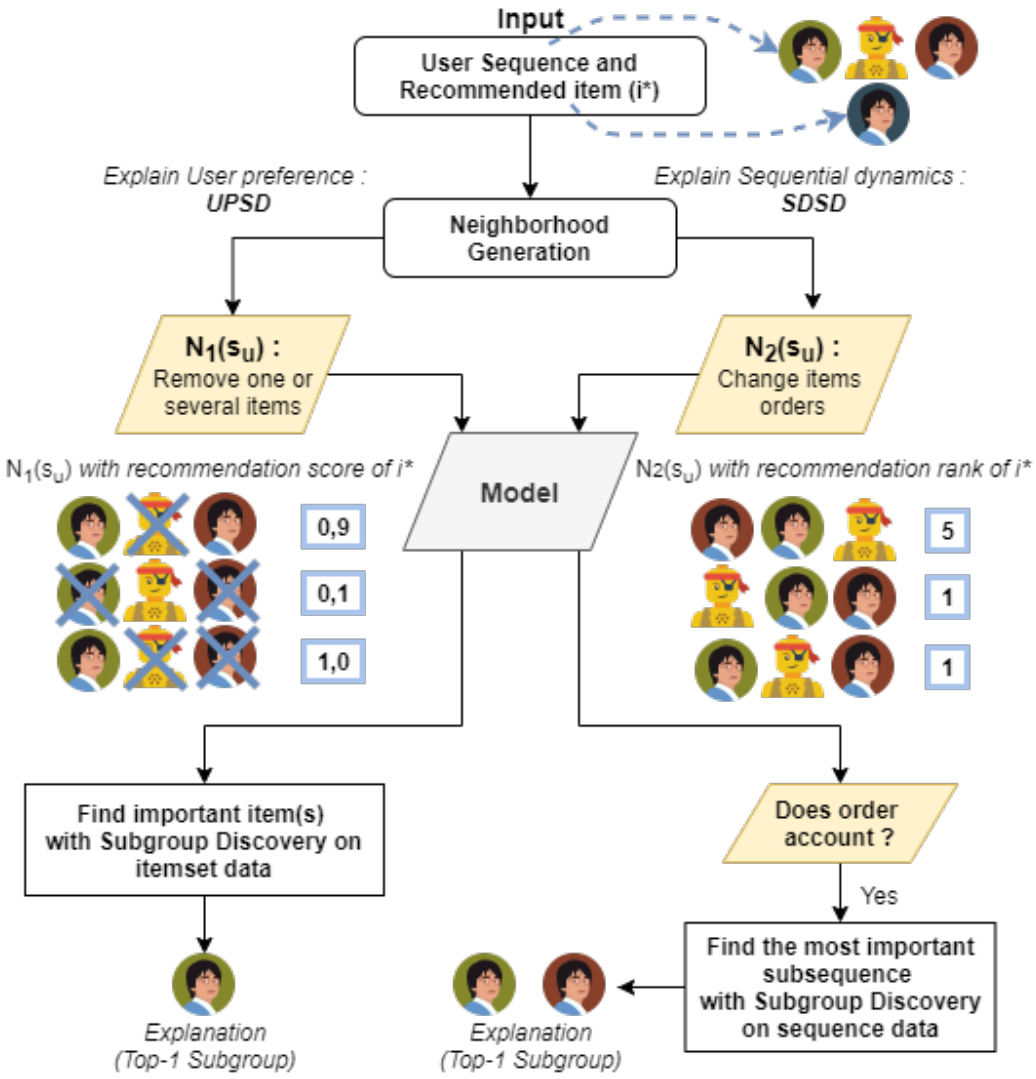

Figure 4.1: Overview of how the explanation is created given a user sequence and the corresponding recommended item i^*.

show the applicability of the proposed approach and method. We provide an extensive discussion of our results in context.

The rest of the chapter is structured as follows: Section 4.2 introduces the notations. we review the literature in Section 4.3. After that, Section 4.4 presents the applied method. Next, Section 4.5 describes our experiments and discusses their results. Finally, Section 4.6 concludes with a summary and interesting directions for future work.

4.2 Notations

In this section, we introduce the notations we use throughout this chapter to define our proposal. Let us consider a set of users U: Each user $u \in U$ is described by a history of traces

of her interactions with the system s_u – a sequence of actions. Interactions can be diverse: Listening to music, watching a movie, buying a product, reading a document, etc. We denote products or services made available by the system as a set of items I. A user interacts with an item if she takes it. Her interactions are ordered in time. Then, the *user history* of u is defined as $s_u = \langle i_1, \ldots, i_k \rangle$, $i_1, \ldots, i_k \in I$ and i_j being an action prior to i_k, for $j < k$. We use I_{s_u} to denote items that appear in s_u.

A recommender system $\mathcal{R}$ takes as input a user history s_u and produces *recommendations*: $\mathcal{R}(s_u) = \{(i, r_i^{s_u}),\ i \in I\}$. For each item $i \in I$ a value $r_i^{s_u} \in \mathbb{R}$ is associated, such that the higher this value, the more the respective item is recommended to user u. The numerical value $r_i^{s_u}$ of item i can only be interpreted relative to the values $r_j^{s_u}, j \neq i$ associated with other items, yielding the ordered recommendations: $\mathcal{R}_{order}(s_u) = \langle i_1, \ldots, i_{|I|} \rangle$ with $r_{i_k}^{s_u} \geqslant r_{i_\ell}^{s_u}\ \forall k < \ell,\ k, \ell \in \{1 \ldots |I|\}$.

We propose to represent an explanation by a pattern, given a description language, reflecting the items having an important influence on the ranking of i^* which is the Top-1 recommended item (the item whose recommendation has to be explained).

Notations used throughout this chapter are summarized and exemplified in the Table page ix. Notice that, this table contains all the notations of this thesis.

4.3 Related work

4.3.1 Model Explanation

Recently, the concept of transparent and explainable models has gained a strong focus and momentum in the data mining and machine learning community, specifically regarding model-agnostic explanations e. g., (Biran and Cotton, 2017, Guidotti et al., 2019, Li and Huan, 2017, Ribeiro et al., 2016). In this chapter, we specifically focus on such model-agnostic methods, utilizing subgroup discovery techniques, as interpretable and explainable patterns are known to significantly improved model assessment and evaluation (Atzmueller and Roth-Berghofer, 2010, Fürnkranz et al., 2020).

Several methods focus on specific model types, e. g., tree-based models (Tolomei et al., 2017) or local pattern-based approaches(Bloemheuvel et al., 2019, Duivesteijn and Thaele, 2014), e. g., for enhancing explanation or for getting a better understanding of where a classifier does not work. While the methods sketched above focus on specific modeling methods, there are several approaches for model agnostic explanation methods, e. g., (Ribeiro et al., 2016, 2018). In particular, general directions are given by methods considering counter factual explanation, e. g., (Mandel, 2007). Furthermore, other general methods consider data perturbation and randomization techniques as the basis for the investigation of black box models, e. g., (Henelius et al., 2014). Our proposed approach also considers perturbation techniques, where we specialize these for the particular case of recommender systems. In particular, we apply subgroup discovery methods for providing post-hoc explanations which provides the following benefits: (1) The approach can be applied to black-box recommendation model where a perturbation on the original user sequence has an effect on the provided recommendation – making it model-agnostic; (2) subgroup discovery is a prominent method for obtaining descriptive patterns (rules) which are assessed as one of the most interpretable models (Fürnkranz et al., 2020).

4.3.2 On Explanation in Recommender Systems

Explanation and *explanation-aware* approaches have been widely investigated in different disciplines, e. g., in artificial intelligence, data science, etc. e. g., (Roth-Berghofer et al., 2007, Schank, 1986, Wick and Thompson, 1992). Roth-Berghofer and Cassens (2005) outline the combination of goals and kinds of explanations, in the context of case-based reasoning. Sørmo et al. (2005) suggest a set of explanation goals addressing transparency, justification, relevance, conceptualization, and learning. Explanation goals specifically help to focus on user needs and expectations towards explanations. They aim to understand what and when the system has to be able to explain (something). For recommender systems, different approaches for providing explanations have been studied, e. g., (McSherry, 2005, Tintarev and Masthoff, 2011) targeting mainly content-based, collaborative filtering, and case-based approaches since explanation-awareness is an important factor for supporting the user, e. g., (Tintarev and Masthoff, 2011). Here, explanation is mainly integrated into the respective method. In contrast, for (black box) machine learning methods to be used for recommendation, explanations have been largely neglected. Our chapter investigates and tackles this problem, proposing a framework and method for model-agnostic explanations only utilizing the recommendation history.

4.4 Identifying the active data used for recommendation

We propose a model-agnostic [2] approach for local explanation in the context of recommender systems. Here, we consider a specific user, and analyze the recommendations according to her user history (i. e., her previous system interactions). Similar to (Ribeiro et al., 2016), we explore the neighborhood of these recommendations. Specifically, our objective is to isolate the items from the user's history which lead to a particular recommendation or favor an item. First, we generate profiles close to that of the studied user and calculate what the model associates as recommendations to these profiles. Second, we analyze them in order to isolate the groups or sequences of items for which the item initially recommended exhibits a high rank or a high score in a recommendation using subgroup discovery.

4.4.1 Neighborhood Generation

To identify the active items used for the recommendation, we explore two neighborhoods of user sequence s_u:

1. To focus on the impact of a group of items on the recommendation, we generate neighborhood histories $N_1(s_u)$ by removing one or several items from s_u. The possible number of such sequences is equal to $2^{|s_u|}$. If the recommendation is mainly based on a group of items, then removing these items from the respective sequence would have a great impact on the recommendation;
2. To assess the importance of the item order on the user history s_u we consider another neighborhood $N_2(s_u)$ consisting of sequences where only the order changes. Starting with the observed sequence s_u, we shuffle it by randomly picking an item of the sequence and swapping it with one of its neighbors. This process is repeated n times where n

[2] With the limitation that the model must be able to provide different score/rank for user u and item i if the input sequence is modified.

is chosen randomly in the interval $[5, 25]$. Thus, the parameter n is used to control the amount of disturbance incurred by the sequence. At the end of this process, the perturbed sequence is added to the neighborhood. We produce K such perturbed sequences to form the neighborhood[3]. The function is presented in Algorithm 2.

Algorithm 2: N_2_Generation(s_u, K)

Input: A user sequence s_u and the number of desired perturbed sequences K
Output: $N_2(s_u)$, Neighborhood of s_u made of sequences where only the items order of s_u changes.

```
nbActions ← |s_u|
N_2 ← [s_u]
for k = 0 to K do
    pertubSeq ← s_u
    N ← random number between 5 and 25
    for n = 0 to N do
        item ← random item in pertubSeq
        pertubSeq ← SwapItemWithNeighbors(pertubSeq,item)   ▷ Swap the random
          selected item with one of his neighbors and retrun the new perturbed
          sequence
    N_2 ← N_2 · pertubSeq                                   ▷ Append pertubSeq to N_2
return N_2
```

4.4.2 Subgroup Discovery for Analyzing Recommendations

We propose to apply subgroup discovery (Wrobel, 1997) to identify the *active data* used for recommendation – essentially, to provide an overview on the relationships between a *target variable* (or target attribute) and *explaining variables*, i. e., a subset of s_u in our context. In contrast to analyzing a recommendation sequence as a whole, typically only items at the top of the ranking are relevant. Therefore, we target a particular item $i^\star$, i. e., the top-1 recommended item. The target attribute is the numerical attribute $(r^s_{i^\star})_{s \in N}$ for sequences s taken in a set N.

We are interested in two types of explanations: (1) What are the items that "trigger" the recommendation of a particular item? (2) Is the order of those items important? Therefore, we consider two different subgroup description languages.

User Preferences via Subgroup Discovery (UPSD) The subgroup description language $\mathcal{I}$ used for active item identification is defined as all possible sets of items from I: $\mathcal{I} = 2^{|i|}$. A descriptive pattern $d \in \mathcal{I}$ *covers* a user history s_u iff $d \subseteq I_{s_u}$, the set of items that appear in s_u. Thus, considering a given set of user sequences N, the subgroup of N described by d, i. e., the *pattern cover*, is made of the sequences of N that contain all the items of d: $SG(d) = \{s \in N \mid d \subseteq I_s\}$ where I_s is the set of items that appear in the sequence s.

For characterizing the items of the sequence s_u that lead to the recommendation of the item $i^\star$, we consider subgroups of $N = N_1(s_u)$ whose associated target values are large com-

[3]NB: we remove duplicated perturbed sequences in a post-processing step.

pared to the remaining sequences of $N_1(s_u)$. More formally, we are looking for descriptions d such that the average value $\overline{r_{i^\star}(d)}$ of values $r^s_{i^\star}$, $s \in SG(d)$ is large. As proposed in (Lemmerich et al., 2016), there exists several quality measures that consider subgroups with respect to numeric target variables. In our case, we aim to find subgroups that are relatively large while also having large values for a numerical target attribute. This then corresponds to having large deviations compared to the mean of the target attribute computed on the whole dataset. For subgroup discovery, we apply the SD-Map algorithm instantiated for numeric target variables (Lemmerich et al., 2016), which discovers the top-k subgroups according to a given quality function, taking into account specific user sequences. In our context, we apply the *qmean* function (inspired by the simple binomial quality function for numerical target attributes, c. f., (Lemmerich et al., 2016)). This quality function considers the deviation of the mean of the target attribute in the subgroup and the total dataset (given by the set of neighboring sequences for a user sequence). So, for a user u and her user sequences s_u, with $N = N_1(s_u)$, we compute

$$qmean(d, i^\star, N) = \sqrt{|SG(d)|} \cdot (\overline{r_{i^\star}(d)} - \overline{r_{i^\star}(\varnothing)}) \,,$$

where $\overline{r_{i^\star}(\varnothing)}$ is the mean of $r_{i^\star}$ over N and the pattern cover $SG(d)$ is computed on N accordingly. We also assessed further adaptations of the quality functions proposed in (Lemmerich et al., 2016) like the Piatetsky-Shapiro adaptation for numeric targets; the results were similar, while the *qmean* function resulted in slightly better subgroups in our application context concerning the deviations of the target attribute. Hence, for **UPSD** explanations, we consider the top-1 pattern that maximizes *qmean* function.

Sequential Dynamics via Subgroup Discovery (SDSD) To identify the potential order effect on the recommendation, we consider the subgroup description language $\mathcal{S}$ made of all sequences of items of I without repetition: $\mathcal{S} = \Pi_{k=0}^{|I|} 2^{|i|-k}$. A pattern $d \in \mathcal{S}$ covers a sequence s, $d \sqsubseteq s$, iff d appears in s in the same order: Let $d = \langle i_1, \ldots, i_k \rangle$ and $s = \langle s_1, \ldots, s_\ell \rangle$, there exists indices $j_1 < \cdots < j_k$ such that $s_{j_h} = i_h, \forall h = 1 \ldots k$. The subgroup of a set of user sequences N described by d is the set of sequences selected by d: $SG(d) = \{s \in N \mid d \sqsubseteq s\}$. We can assume that there is an effect of the item order on the recommendation if the first recommended item changes for some sequences of $N_2(s_u)$ (i. e., the variance of the rank of $i^\star$ is different from 0).

In this case, we can look for the sub-sequence that favors the target item $i^\star$. To the best of our knowledge, there does not exist any subgroup discovery approach considering sequences with numerical target. Therefore, we propose to use SeqScout (Mathonat et al., 2019) designed to identify discriminating sequences when the target is categorical. For this, SeqScout evaluates the quality of a pattern d with the precision measure that is the proportion of sequences selected by d that lead to the recommendation of $i^\star$ at the first position:

$$Prec(SG(d)) = \frac{\{s \in SG(d) \mid r^s_{i^\star} \geqslant r^s_j,\ \forall j \neq i^\star\}}{\{s \in SG(d) \mid \exists j \neq i^\star,\ r^s_j > r^s_{i^\star}\}}$$

To prevent obtaining very precise but small subgroups, SeqScout provides a measure inspired by the weighted relative accuracy (WRAcc) measure, e. g., (Lemmerich et al., 2016):

$$WRAcc(SG(d)) = \frac{|SG(d)|}{|N_2(s_u)|}(Prec(SG(d)) - Prec(\varnothing)),$$

where $Prec(\varnothing)$ is the precision measure over $N = N_2(s_u)$, since $SG(d)$ is computed relative to N as discussed above. SeqScout is a sampling approach that searches for a set of patterns that locally optimize WRAcc, following an exploration-exploitation trade-off. At this end, we only consider the top-1 pattern with respect to WRAcc.

4.4.3 Running example

As an example, we report in Table 4.1 the neighborhood generation for $N_1(s_u)$ and $N_2(s_u)$ with $s_u = \langle 4, 223, 373 \rangle$ and the target item $i^\star = 374$ for user u. The numbers $4, 223, 373$ and 374 identify items. As there are 3 items in s_u, there are 7 sequences in $N_1(s_u)$ – $2^3 - 1$ because we remove the sequence without items – and 6 possible sequences for $N_2(s_u)$. For each perturbed sequence, we computed the score of $i^\star$ for $N_1(s_u)$ and the rank of $i^\star$ for $N_2(s_u)$.

Table 4.2 shows the generation of explanations by **UPSD** and **SDSD**, using the neighborhoods $N_1(s_u)$ and $N_2(s_u)$ of Table 4.1. The best explanations of **UPSD** and **SDSD** are respectively $\{373\}$ and $\langle 223, 373 \rangle$. These results seem relevant. Indeed, for $N_1(s_u)$, we can see that the best scores come when $\{373\}$ is present. For $N_2(s_u)$, we can observe that $i^\star$ is ranked first when $\{373\}$ is the last item. This is why $\langle 223, 373 \rangle$ and $\langle 4, 373 \rangle$ have the same WRAcc value.

Table 4.1: Example of neighborhood generation for $N_1(s_u)$ and $N_2(s_u)$ with $s_u = \langle 4, 223, 373 \rangle$ and the target item $i^\star = 374$.

	$\mathbf{N_1(s_u)}$			$\mathbf{N_2(s_u)}$	
#	Perturbed sequence	Reco. Score	#	Perturbed sequence	Reco. Rank
1	$\langle 4, 223, 373 \rangle$	0.99	1	$\langle 4, 223, 373 \rangle$	1
2	$\langle \varnothing, \varnothing, 373 \rangle$	0.84	2	$\langle 373, 223, 4 \rangle$	23
3	$\langle \varnothing, 223, \varnothing \rangle$	-3.73	3	$\langle 4, 373, 223 \rangle$	62
4	$\langle \varnothing, 223, 373 \rangle$	0.88	4	$\langle 223, 4, 373 \rangle$	1
5	$\langle 4, \varnothing, \varnothing \rangle$	-2.75	5	$\langle 223, 373, 4 \rangle$	23
6	$\langle 4, \varnothing, 373 \rangle$	1.18	6	$\langle 373, 4, 223 \rangle$	62
7	$\langle 4, 223, \varnothing \rangle$	-2.93			

Table 4.2: Example of generation of explanation by **UPSD** and **SDSD** with $N_1(s_u)$ and $N_2(s_u)$ of Table 4.1.

	UPSD			**SDSD**	
#	Explanations	qmean	#	Explanations	WRAcc
1	$\{373\}$	3.52	1	$\langle 223, 373 \rangle$	0.17
2	$\{4, 373\}$	2.65	2	$\langle 4, 373 \rangle$	0.17
3	$\{223, 373\}$	2.44	3	$\langle 4, 223, 373 \rangle$	0.11
4	$\{4, 223, 373\}$	1.78	4	$\langle 223, 4, 373 \rangle$	0.11

4.4.4 Complexity

The complexity of our methods is due to the complexity related to the neighborhood generations and the one due to the computation of the explanations on them. The prediction of the score or rank of $i^\star$ by the recommender system has an important impact on the complexity of the neighborhood generations. Each recommender system has its own computational complexity, denoted RS_{Comp}. Thus, computing $N_1(s_u)$ has a complexity in $O(RS_{Comp} \times 2^{|s_u|})$, while the complexity of $N_2(s_u)$ is in $O(RS_{Comp} \times |I \backslash I_{s_u}| \times K)$, where K is number of generated sequences. We limit the complexity related to the computation of N_1 and N_2 by only considering users with a history size $|s_u|$ between 2 and 15. Considering the computation of the explanations based on $N_1(s_u)$, we use SD-Map (Atzmueller and Lemmerich, 2009) instantiated for numeric target attributes. As discussed in (Atzmueller and Lemmerich, 2009) the complexity of SD-Map grows exponentially with the number of items. However, with the efficient pruning approaches implemented by SD-Map (Atzmueller and Lemmerich, 2009, Lemmerich et al., 2016) the runtime is typically significantly reduced in practical applications. The explanations based on $N_2(s_u)$ are obtained thanks to SeqScout that has a time-budget parameter which allows to tune the runtime easily given a suitable time-budget for controlling the algorithm's complexity.

4.5 Experiments

In this section, we report on our experimental study using several datasets and recommender systems. For reproducibility purposes, all the datasets, the source code, and the experimental results are made available in a repository[4].

4.5.1 Datasets

To evaluate our framework on both sparse and dense datasets from different domains, we have selected 9 datasets from the ones considered in Chapter 3:
Amazon was introduced by (McAuley et al., 2015). It contains Amazon product reviews from May 1996 to July 2014 from several product categories. We have chosen to use the 3 following diversified categories: *Automotive, Office product* and *video games*.
MovieLens 1M[5] (Harper and Konstan, 2015) is a popular dataset including 1 million movie ratings from 6040 users between April 2000 and February 2003. We pre-processed the dataset by selecting the most recent x ratings for each user, $x \in \{5, 10, 20, 50\}$. The resulting datasets allow us to consider different levels on sparsity (i.e. MovieLens with the 10 most recent ratings will be more sparse than MovieLens with the 50 most recent ratings).
Visiativ[6] is a real-world dataset that gathers the downloads of computer-aided design resources on the *myCADservices* platform of Visiativ from November 2014 to August 2018.

For each dataset, we pre-process the data as explained in Chapter 3: Ratings are converted into implicit feedback and we only consider users and items that have at least 5 interactions. The main characteristics of these datasets before and after preprocessing are reported in Table 3.1 and Table 3.2 of Chapter 3. Also, as explained earlier, we constrain the size of $|s_u|$

[4] https://bit.ly/2DD0H0f
[5] http://grouplens.org/datasets/movielens/1m/
[6] https://www.visiativ.com/en-us/

between 2 and 15 in order to limit the computation time of N_1 and N_2. Table 4.3 shows the dataset characteristics after we apply the constrain on $|s_u|$.

Table 4.3: Main characteristics of the datasets after we constrain the size of $|s_u|$ between 2 and 15 : # U is the number of users for which an explanation is produced; $Avg(N_1)$ is the average of the sizes of the neighborhoods N_1; $Avg(N_2)$ is the same for N_2.

	Datasets	#U	$Avg(N_1)$	$Avg(N_2)$
Amaz.	Auto	28666	442	659
	Office	15109	830	971
	Video	28283	1034	1307
ML	ML-5	6039	16	25
	ML-10	6039	493	3422
	ML-20	1091	7801	5016
	ML-50	1091	7801	5016
	Epinions	4219	434	698
	Visiativ	1111	1808	1900

4.5.2 Considered models

We consider 7 different models designed for sequential recommendation:

- Factorized Markov Chains (**MC**) (Rendle et al., 2010), models sequential dynamics by factorizing the item transition matrix;
- Factorized Personalized Markov Chains (**FPMC**) (Rendle et al., 2010) models both user preferences and sequential dynamics using matrix factorization and first-order Markov chains;
- Personalized Ranking Metric Embedding (**PRME**) (Feng et al., 2015), embeds user preferences and sequential dynamics into two Euclidean distances;
- Self-Attentive Sequential Recommendation (**SASRec**) (Kang and McAuley, 2018), is a self-attention based model that captures both user preferences and sequential dynamics;
- Finally, Translation-based Recommendation (**TransRec**) (He et al., 2017a), is a model unifying user preferences and sequential dynamics with translations;
- Recommendation Embedding Based on freqUent Sequences (**REBUS**) (Lonjarret et al., 2021), our model presented in Chapter 3. It is an item embedding model that also uses frequent substrings as a proxy to learn conditional distributions of items in users' histories;
- **REBUS_UP**, is a variant of **REBUS** that does not model the sequential dynamics, but only the user preferences.

To avoid running a grid search to find the best combinations of hyperparameters, we use those previously found in Section 3.4.4 of Chapter 3. We have seen that these combinations make it possible to have good performance on the selected datasets. As a reminder, we summarize in Table 4.4 the performances of these models (the full table is available in Section 3.4.4 of Chapter 3).

Figure 4.2 illustrates the diversity of the recommendations provided by each model. We compute the percentage of items recommended at least once – i. e., the number of unique

Table 4.4: Performance of the models on 9 datasets – AUC, HIT10 and HIT50. Top scores for each dataset/metric in bold.

Models	Metric	Epinions	Visiativ	Auto	Office	Video	ML-5	ML-10	ML-20	ML-50
FPMC	AUC	0.5536	0.8433	0.6482	0.6958	0.8777	0.7309	0.815	0.8629	0.8891
	HIT10	0.0084	0.3125	0.0116	0.0071	**0.0692**	0.0411	0.0752	**0.0947**	**0.1035**
	HIT50	0.0251	0.6036	0.0364	0.0253	0.1833	0.1326	0.229	0.2628	0.2863
MC	AUC	0.5421	0.8354	0.6251	0.6771	0.8473	0.7414	0.8054	0.8446	0.8689
	HIT10	0.0102	0.3197	0.0146	0.0076	0.0598	0.0546	0.0821	0.0937	0.1012
	HIT50	0.0263	0.5878	0.0378	0.0267	0.1571	0.1525	0.206	0.2495	0.2772
PRME	AUC	0.6071	0.8572	0.6749	0.7154	0.8759	0.7771	0.8302	0.8645	0.8867
	HIT10	0.01	0.3269	0.0116	0.0118	0.0683	0.053	0.083	0.0729	0.0944
	HIT50	0.0272	0.5943	0.0385	0.0487	**0.1874**	0.1694	0.238	0.2439	0.2714
SASRec	AUC	0.6251	0.8731	0.6861	0.7392	0.8812	0.8	0.8537	0.876	**0.8957**
	HIT10	0.0156	0.3563	0.0124	0.0167	0.0502	0.0593	0.0816	0.0792	0.0965
	HIT50	0.04	0.6538	0.0407	0.0612	0.1454	0.2083	0.2532	0.2596	**0.2929**
TransRec	AUC	0.6138	0.8647	0.6991	0.732	0.8869	0.79	0.8477	0.8736	0.8883
	HIT10	0.0126	**0.3663**	0.0227	0.0183	0.0564	**0.0758**	**0.0876**	0.083	0.0841
	HIT50	0.0314	0.6473	0.0599	**0.0637**	0.1651	0.2245	**0.256**	**0.2651**	0.2787
REBUS	AUC	**0.6524**	**0.8735**	**0.7184**	**0.7507**	**0.8934**	**0.8031**	**0.8563**	**0.8772**	0.8884
	HIT10	**0.0174**	0.3534	**0.0247**	**0.0214**	0.0482	0.0734	0.0753	0.0767	0.0825
	HIT50	**0.0532**	**0.6717**	**0.062**	0.0624	0.1461	**0.2279**	0.255	0.2601	0.2691
REBUS_UP	AUC	0.6495	0.8521	0.7147	0.7484	0.886	0.7941	0.8428	0.8603	0.8671
	HIT10	0.016	0.271	0.0234	0.0211	0.0445	0.0654	0.0578	0.053	0.0502
	HIT50	0.05	0.5964	0.0602	0.0606	0.1414	0.2151	0.2072	0.1999	0.1949

recommended items divided by the number of item $|I|$ in each dataset. We can observe that the models **REBUS**, **REBUS_UP**, **SASRec** and **TransRec** often recommend the same item, compared to other models, while having good AUC or HIT values. As we will see below, this characteristic has an impact on the identified subgroups and their ability to explain a model.

4.5.3 Aims

The experiments aim to assess the capacity of our method to provide relevant explanations about recommendations provided by any model, and to bring local explanations that give insights about the models themselves. Taking this into account for the experimental design this empirical study aims to answer the following questions:

1. Can the explanation be interpreted? What are the elements responsible for the recommendation? Is the user's sequential dynamics taken into account? *Answers to these questions will make it possible to assess if the local explanations are relevant or not.*
2. Are local explanations generalizable?
3. To what extent are the models based on their local explanations?
4. Can we build an interpretable global model based on local explanations?

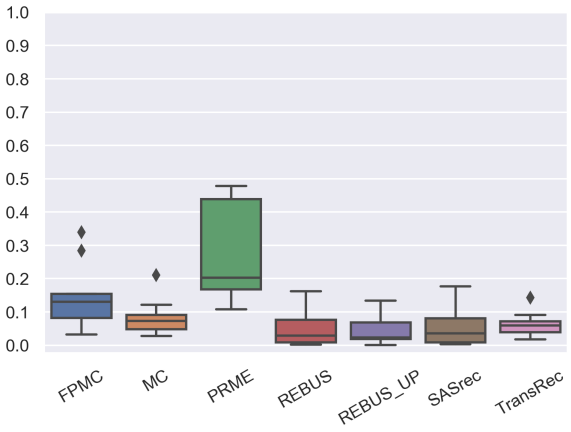

Figure 4.2: Percentage of items that are recommended at least once for each model. Values are averaged over all datasets

4.5.4 Applying UPSD and SDSD

To explain the recommendations made by the models on the different datasets, we compute the top-1 subgroup with **UPSD** and **SDSD** for all users such that $2 \leqslant |s_u| \leqslant 15$. For instance, considering user u,

1. **UPSD** generates the set of all possible neighbors $N_1(s_u)$ – i. e., all subsets of I_{s_u} – and for each sequence s in $N_1(s_u)$ and each model, it computes the recommendation score of $i^\star$.

2. **SDSD** generates the neighborhood $N_2(s_u)$ with $K = 10000$, as explained in Section 4.4.1. To evaluate whether the item order has an effect on the recommendation, it computes the rank of the target item $i^\star$ for each model.

Table 4.3 gives the average sizes of N_1 and N_2 for each dataset. The objective of the two methods is to evaluate the robustness of the recommendation for user u of the item $i^\star$, the first item recommended to u based on the sequence s_u.

4.5.5 Local Explanations

Examples of Local Explanations: Figure 4.3 reports some examples of explanations of recommendations provided by **REBUS** and **SASRec**. These local explanations are the results of **UPSD** and **SDSD**, and reflect the most decisive items in the recommendation (**UPSD**) as well as the sub-sequence of items that is characteristic for the recommended item (**SDSD**). In the first row, **REBUS** and **SASRec** make a relevant yet different recommendation, as highlighted by **UPSD** and **SDSD**, on products related to the Nintendo Wii. On the second row, **REBUS** and **SASRec** make a more questionable recommendation: They both recommend item related to Sony PlayStation while the last 3 purchased items of the user are related to Computer. Moreover, we can see that **REBUS** and **SASRec** recommend these items in other contexts (as in row 3 for **REBUS** and row 5 for **SASrec**), but here the recommendation is relevant as shown by the local explanations. To sum-up, we can

see that local explanations found by **UPSD** are very close for both models, whereas there are more differences on the local explanations found by **SDSD**.

Figure 4.3: Explanation examples in the Video dataset. An orange (resp. blue) line points to the important items for **REBUS** (resp. **SASRec**). If the item is important for both models, then the line is green. The lines above an item mean that the item is important according to the **SDSD** method, while the lines below mean that the item is important according to the **UPSD** method.

Figure 4.4 reports some local explanations for the **REBUS** and **SASRec** models for the Visiativ dataset. These examples involve users' navigation through Computer-aided design (CAD) resources (i. e., documentation, macro, library) of the *myCADservices* platform. For the first sequence, the local explanations for both models are similar except the first item provided by **SDSD** method. The recommendation provided by **SASRec** is relevant because it suggests an item related to the PIPING components. Notice that this recommendation is correct. **REBUS** provides a popular item that is relevant but less personalized. In the second sequence, **REBUS** and **SASRec** provide two different, yet relevant, recommendations. **SASRec** identifies materials, and color as active items to recommend an item related to texture. **REBUS** build its recommendation on materials, and outdoor images to recommend a new persona image library. For the third sequence, the recommendations are the same as for the second one because the sequences are almost similar. Notice that, **SASRec** does not identify any particular order on the items that may explain the recommendation. For **REBUS**, **SDSD** sheds light on the two last items of the user sequence. This demonstrates that a same (recommended) item can be explained in different ways even if the user sequences are almost similar.

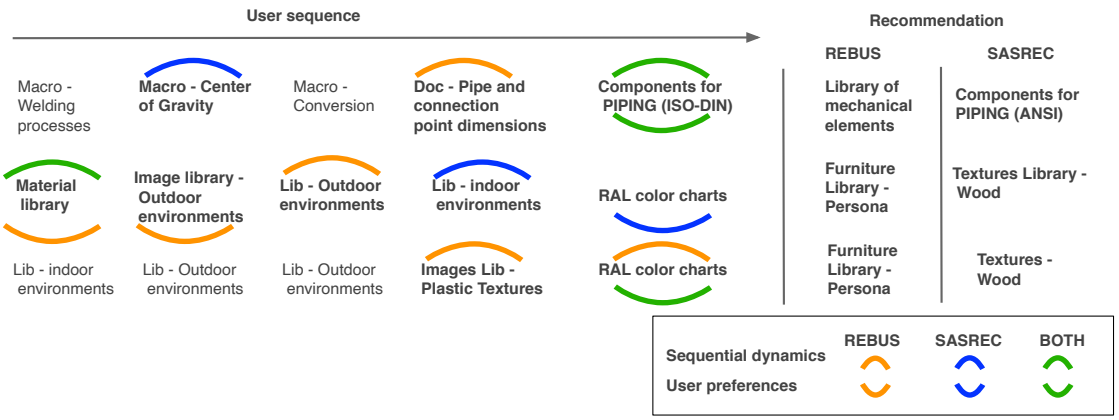

Figure 4.4: Examples of recommendation explanations in Visiativ dataset. Orange (resp. blue) line points to the important items for **REBUS** (resp. **SASRec**). If the item is important for both models, then the line is green. The lines above the item mean that the item is important according to the **SDSD** method, while the lines below the item mean that the item is important according to the **UPSD** method.

Analysis of Local Explanations: Table 4.5 reports the proportion of user sequences for which an explanation is given by **UPSD** and **SDSD**.

In most cases, **UPSD** identifies items related to user preferences, with the exception of **REBUS_UP** for which there is no explanatory item in 7% of the user sequences. The distribution of the number of items in the explanations provided by **UPSD** is given in Figure 4.5. Explanations are often made up of a single item for most of the models, with the exception of **REBUS**, **REBUS_UP** and **SASRec** for which longer explanations are given (i. e., with 2 or 3 items in the descriptions of the subgroups). These results were expected because **FPMC**, **MC**, **PRME** and **TransRec** are build with one-order Markov chains. Hence, only a perturbation on the last item can have an effect on the recommendation score.

Table 4.5: Proportions of users who get a local explanations from **UPSD** and **SDSD**. Proportions are averaged over all the datasets.

	FPMC	**MC**	**PRME**	**REBUS**	**REBUS_UP**	**SASrec**	**TransRec**
UPSD	1.00	1.00	1.00	1.00	0.93	1.00	1.00
SDSD	0.85	0.97	0.96	0.69	0.00	0.80	0.93

For the sequential dynamics, we can report slightly different observations. For the **REBUS_UP** model, the **SDSD** method does not provide any explanations regardless of the dataset (see Table 4.5). Thus, it seems that the order between the items does not influence the recommendation made by the model, which is indeed the case by construction of the method (Lonjarret et al., 2021). Some models take sequential dynamics into account for some datasets (e. g., **SASRec** for the ML-50 dataset see Fig 4.6), while the order of the items taken by the user clearly has no impact for other datasets (e. g., **REBUS** for Auto where some users have no explanation). Box plots of the explanation size for sequential dynamics are shown in Figure 4.6. We observe that the descriptions which explain the sequential dynamics of the users are longer compared to those w.r.t. the user preferences. However, it is important to

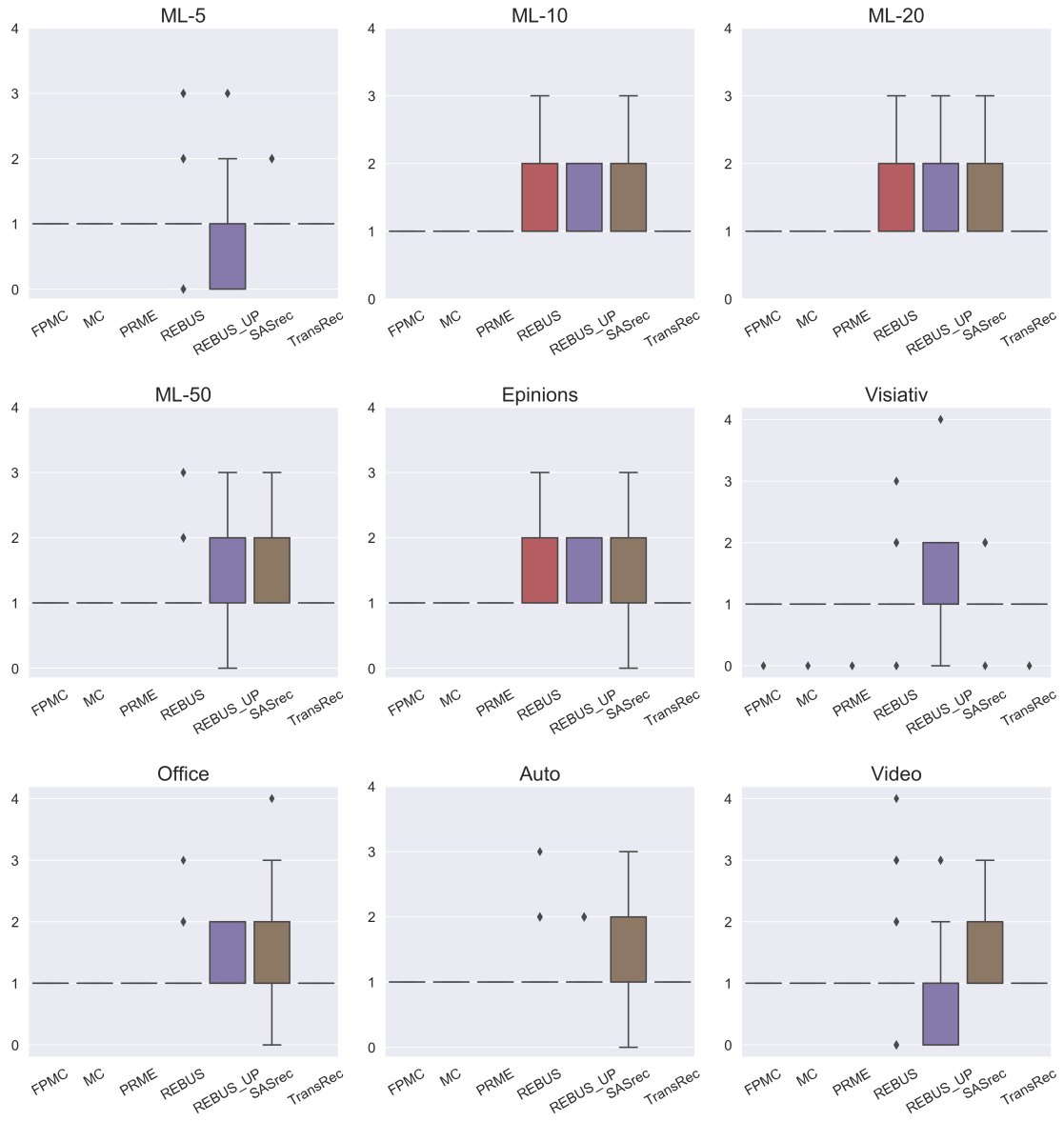

Figure 4.5: Number of items in the explanations given by **UPSD**.

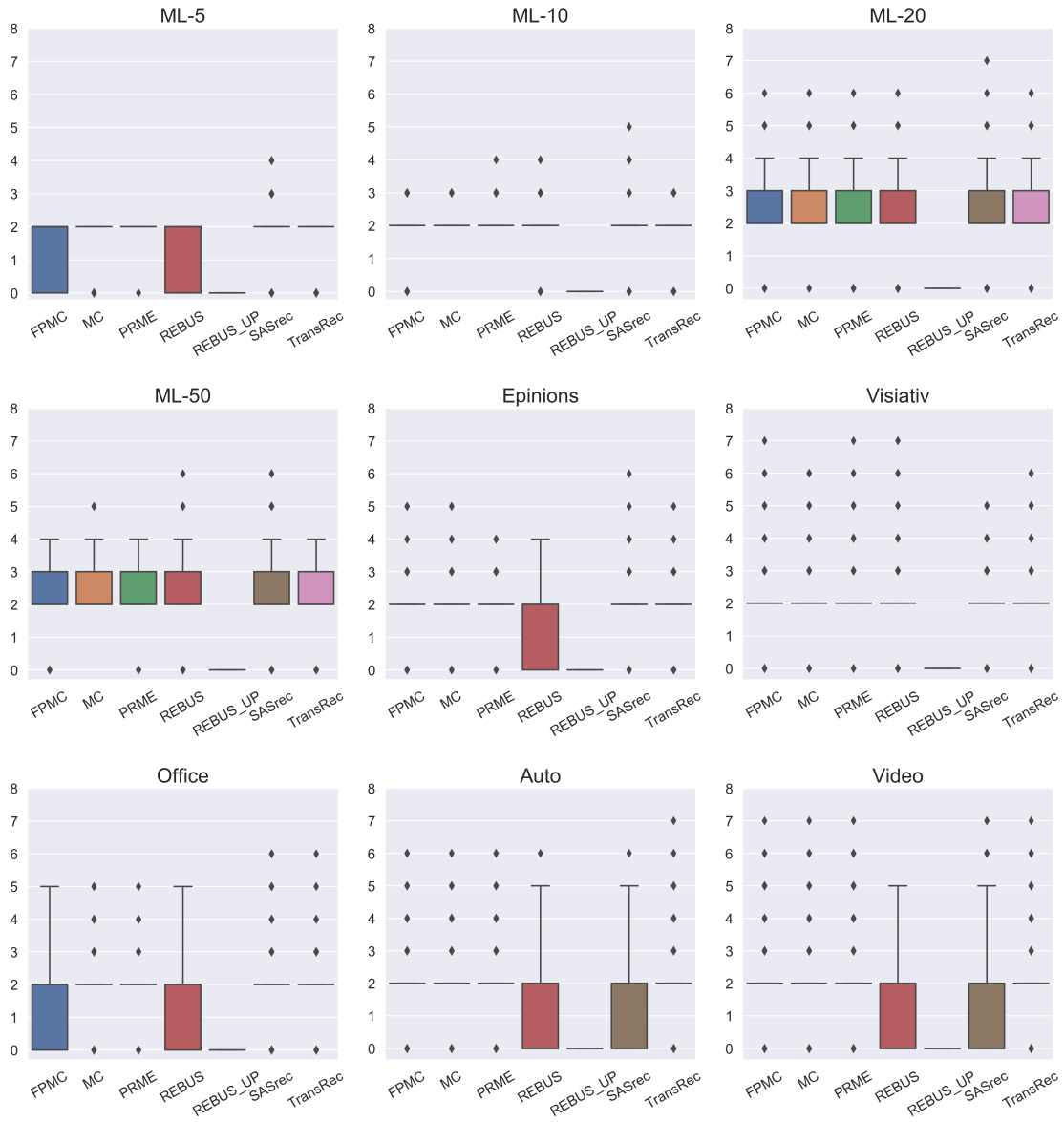

Figure 4.6: Number of items in the explanations given by **SDSD**.

note that there is no sequence of size 1, otherwise it would mean that the order of the item has no impact on the recommendation. That being said, most of the explanations contain two items and are simplest in terms of sequence structure. It is also normal that **SDSD** finds explanation for **FPMC**, **MC**, **PRME** and **TransRec** even if they use a Markov chains of order one. The search for explanations shows that the order of the items is important, which is indeed the case by construction of these models. However, when the explanations consist of more than 2 items, this is due to the fact that they are items close to the last items of the explanation.

It is important to note that our method cannot fully explain models that embed a user vector to make recommendations such as **FPMC**, **MC**, **PRME** and **TransRec**. The explanation for these models is partial in the sense that it is personalized to the user reducing the coverage for the explanation on other users. On the contrary, it can fully explain the recommendation of **REBUS** or **SASrec** because only items are used to make a recommendation.

Comparison of local explanations: The goal in this experiment is to compare one by one explanations found by **UPSD** and **SDSD** for each model, and see if there are any similarities between model explanations. In figures 4.7 and 4.8, we respectively report Jaccard similarity between each model explanation found by **UPSD** and a normalization of the longest common subsequence (LCS) between each model explanation found by **SDSD**. For **UPSD** and **SDSD**, we can report that two groups stand out. The first one gathers **TransRec**, **PRME**, **MC** and **FPMC** models that have more than 50% of similar explanations. And the second one is made of **REBUS** and **SASRec** which have more than 40% of similar explanations. **REBUS_UP** is close to **REBUS** but this is not the case for the others models. Note that for **SDSD**, similar explanations of **REBUS_UP** with all other models occur when there is no explanation.

4.5.6 Assessing Local Explanations Regarding Other Users

We are now studying how local explanations can be generalized to other users. Table 4.6 reports the proportion of explanations that are user-specific (i. e., only applicable for a single user), applicable to a small number of users (i. e., less than 6) and applicable to at least 6 users. We can observe that the explanations provided by **UPSD** are much more generalizable than the explanations provided by **SDSD**: Less than 20% of the **UPSD** explanations are not applicable to other users while in contrast this holds for more than 50% for the explanations provided by **SDSD**. More than 60% of the **UPSD** explanations are applicable to at least 6 users while it is barely 10% for the explanations provided by **SDSD**.

Explanations which are generalizable, i.e. descriptions which are also present in other user sequences, are also of high quality (w.r.t. AUC, HIT1 and HIT25). HIT1 is equal to the proportion of the item explained by subgroups that are equal to the recommendation made by the model, while HIT25 is a less restrictive measure. AUC estimates how high the item explained by subgroups is ranked on average by the model. In Figure 4.9, we report the aggregated quality [7] over all the datasets of generalizable explanations which are applicable to at least 6 users. Two groups of models emerge from these experiences. The explanations

[7]The individual results of each dataset are reported in Appendix figures 7.1 and 7.2 for **UPSD** and figures 7.3 and 7.4 for **UPSD**.

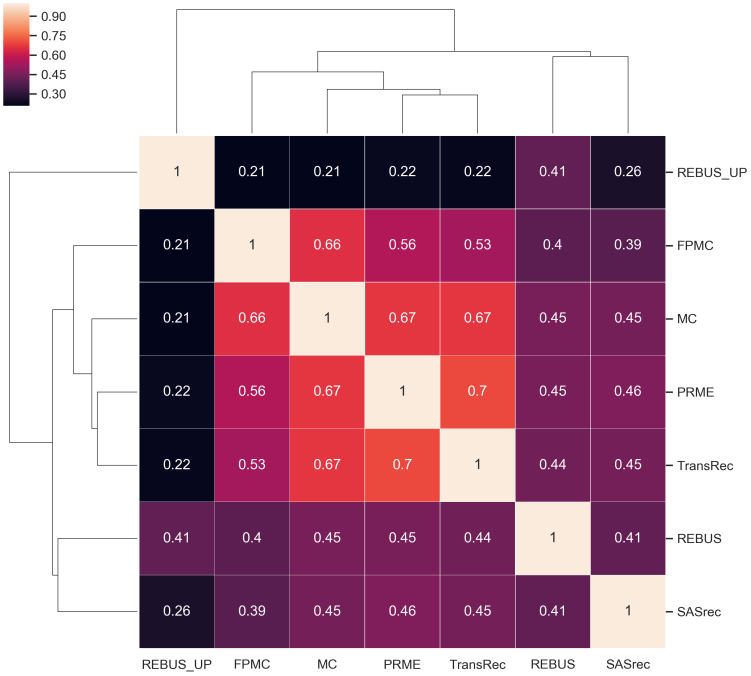

Figure 4.7: Jaccard similarity between each model explanation found by **UPSD**.

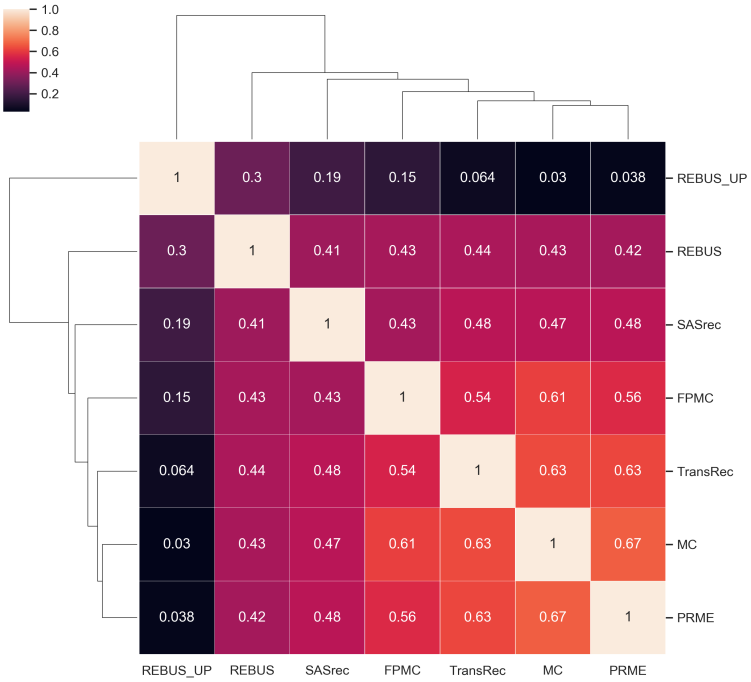

Figure 4.8: Normalization of the longest common subsequence (LCS) between each model explanation found by **SDSD**.

Table 4.6: Proportion of explanations that apply (1) to a single user (user-specific), (2) between 2 and 5 users, and (3) equal or greater than 6 users.

UPSD

	FPMC	**MC**	**PRME**	**REBUS**	**REBUS_UP**	**SASrec**	**TransRec**
User-specific	0.02	0.03	0.04	0.14	0.20	0.20	0.04
2 to 5 users	0.17	0.16	0.15	0.12	0.15	0.15	0.16
$\geqslant$ 6 users	0.81	0.81	0.81	0.74	0.57	0.65	0.80

SDSD

	FPMC	**MC**	**PRME**	**REBUS**	**REBUS_UP**	**SASrec**	**TransRec**
User-specific	0.60	0.68	0.68	0.48	0.00	0.60	0.67
2 to 5 users	0.15	0.17	0.17	0.14	0.00	0.13	0.17
$\geqslant$ 6 users	0.10	0.11	0.11	0.08	0.00	0.07	0.10

of **REBUS**, **REBUS_UP**, **SASRec** and **TransRec** get a better performance than those of other models for all metrics. These four models are the best in the state of the art. They make it possible to model some user habits which cannot be achieved with other models. Thus, their explanations are more accurate when applied to other users. In the case of **TransRec**, it is interesting to note that despite the embedding of a user vector to make recommendations, the explanation is still relevant and applicable to other users.

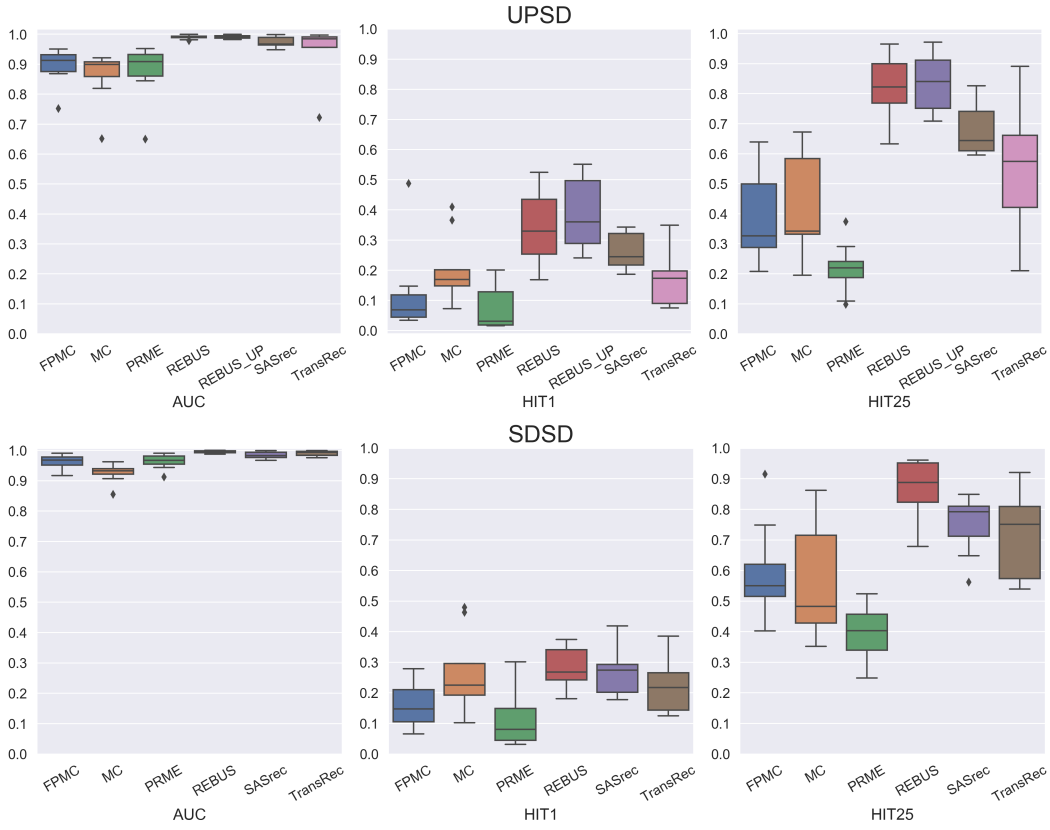

Figure 4.9: Performance metrics (AUC, HIT1 and HIT25) evaluated on **UPSD** subgroups descriptions found in at least 6 users (Top) and on **SDSD** patterns found in at least 6 users (Bottom) for each model. Values are averaged over all the datasets.

Figure 4.10 shows the ratio of unique explanations over the total number of explanations (the number of users). Thus, the lower the ratio, the lower the number of different explanations. The explanations hold for several users since the average ratio is around 0.5 for **UPSD**. For **SDSD**, the aforementioned models (that find explanatory sequences) have a high ratio. Most of the explanations on the sequential dynamics are different. This observation does not hold for models that best exploit the order of user items to make recommendation (**REBUS** and **SASRec**): They return explanations that hold for other users.

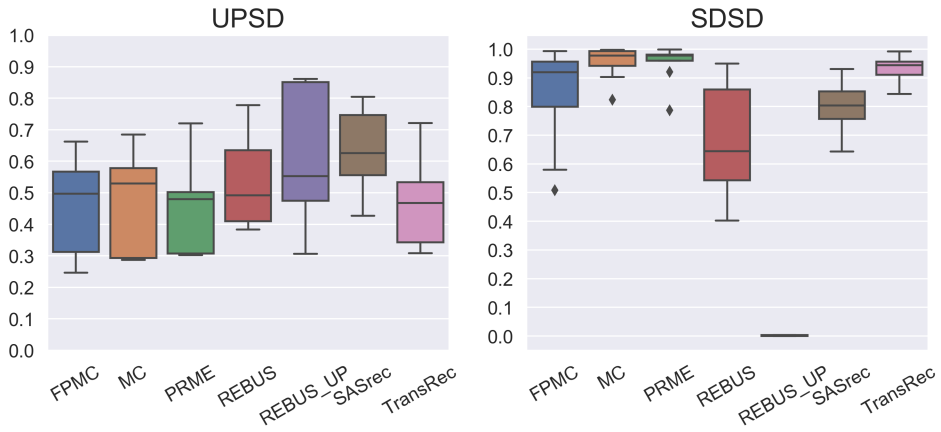

Figure 4.10: Ratio of unique explanations over the total number of explanations obtained by **UPSD** (left) and **SDSD** (right) with respect to the model used. Values are averaged over all datasets.

4.5.7 Towards Global Explanation of Black Box Models for Sequential Recommendation

We now investigate how a set of local explanations can explain most of the recommendations of a recommendation model. This experiment aims to study how many local explanations we need in order to provide a global explanation of a black box model for sequential recommendation. Indeed, a set of high quality of local explanations, here called global explanation, can give the expert/administrator of the recommender system a good overview of the recommendation that will be made by the model.

Identifying good local explanations is similar to the weighted set cover problem. We thus use a greedy strategy to iteratively select the local explanation that has the best cover of the remaining users (i. e., those that are not yet covered by an already chosen local explanation). Figures 4.11 and 4.12 report the fidelity of the global explanation related to a recommendation model according to the number of local explanations (greedily selected). The higher the metric (e. g., AUC, HIT1 and HIT25), the higher the fidelity of the global explanation to the related black box.

In figures 4.11 and 4.12, the global explanations (set of local explanations) built from black box models quickly reach an AUC value close to 1 in most of the cases. For instance, 40 local explanations from **UPSD** related to **REBUS**, **REBUS_UP**, **SASRec** and **TransRec** are enough to obtain an AUC of 0.9 for all of the users on Visiativ. For HIT1, the best global

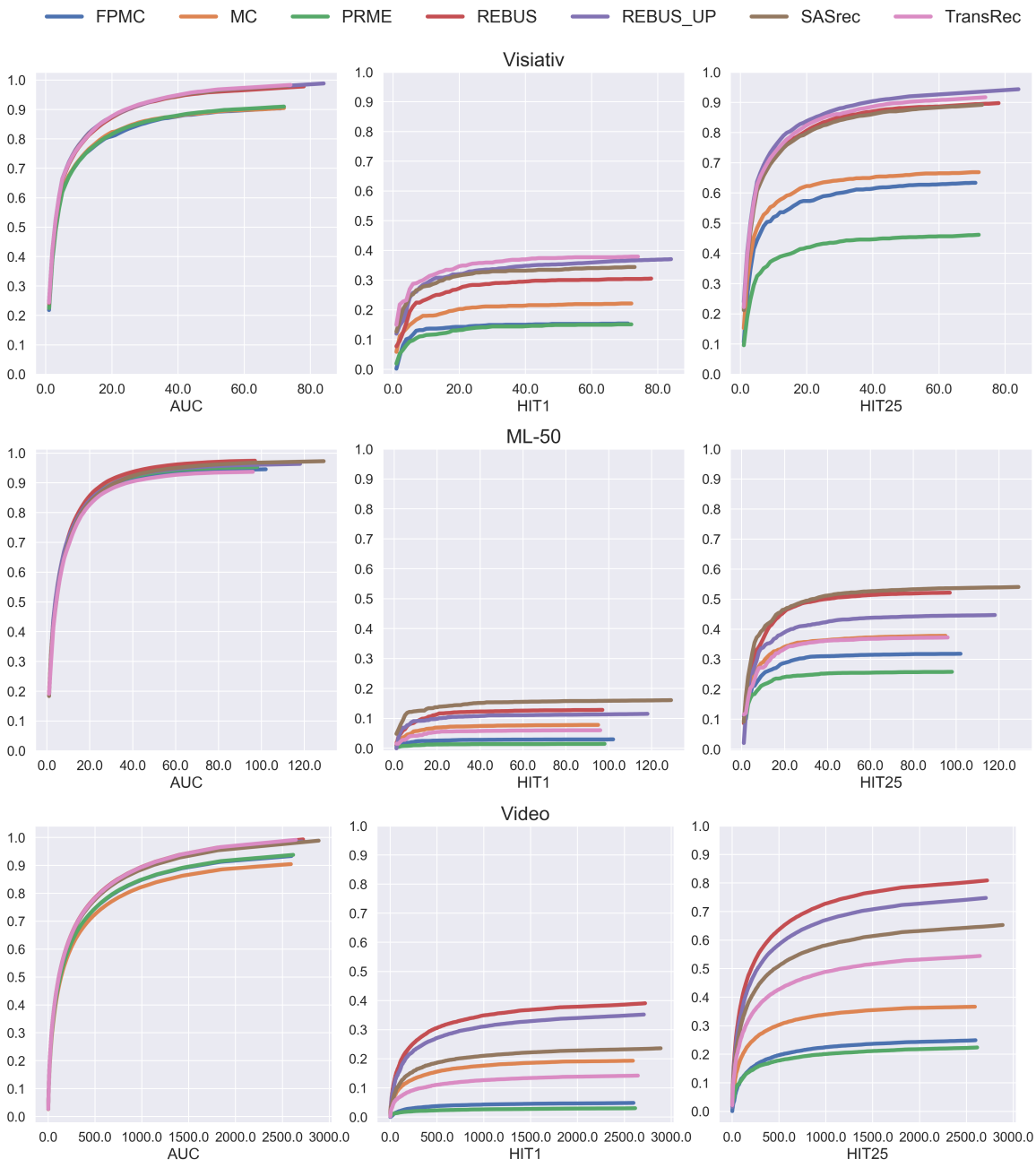

Figure 4.11: AUC, HIT1 and HIT25 values with respect to the number of **UPSD** patterns selected to build the global explanation related to a recommendation model for Visiativ (first row), ML-50 (second row) and Video (last row) datasets. The higher the value, the higher the fidelity of the global explanation to the black box recommender model.

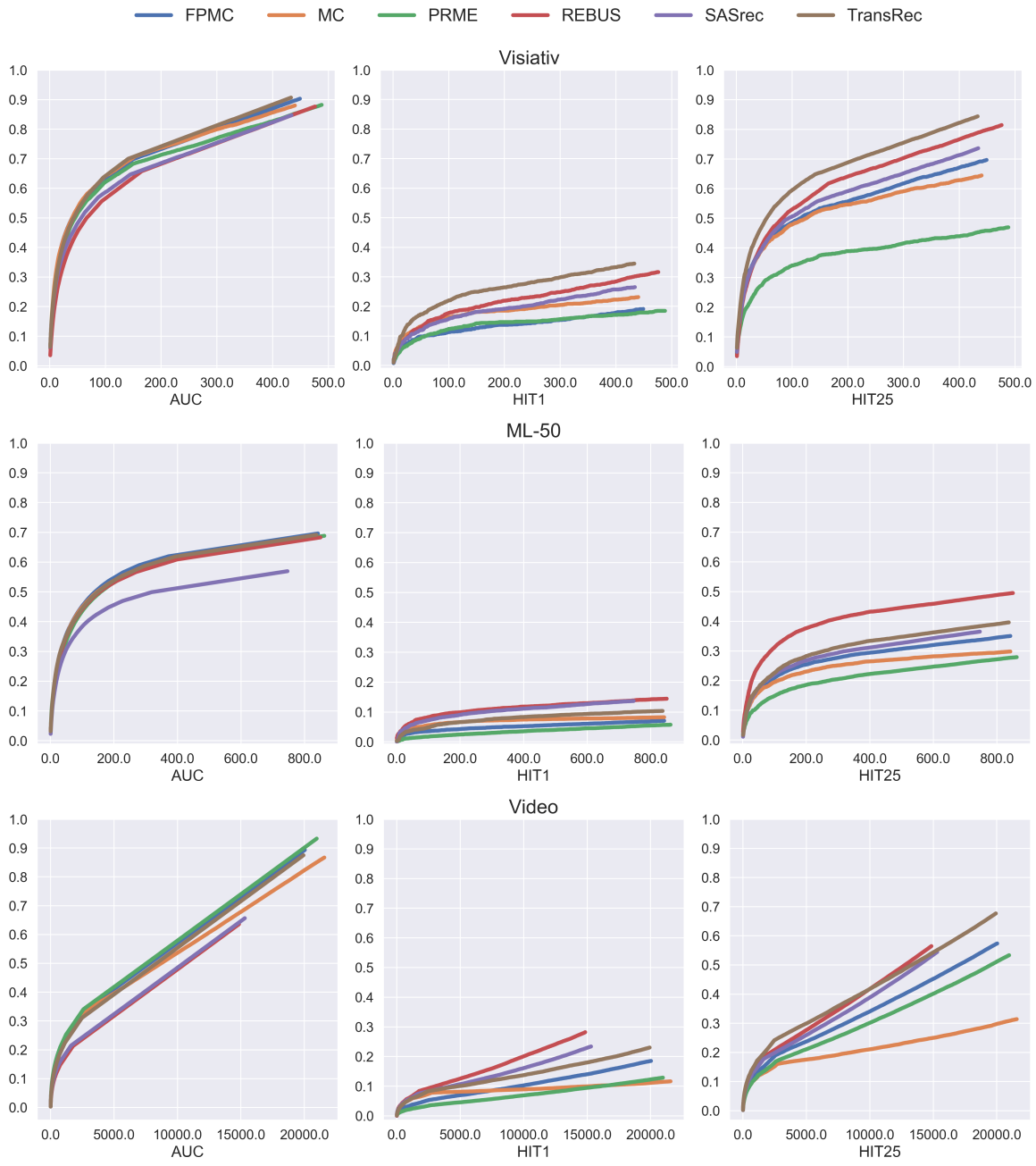

Figure 4.12: AUC, HIT1 and HIT25 values with respect to the number of **SDSD** patterns selected to build the global explanation related to a recommendation model for Visiativ (first row), ML-50 (second row) and Video (last row) datasets. The higher the value, the higher the fidelity of the global explanation to the black box recommender model.

explanations are close to 0.4 for Visiativ and Video datasets with about one quarter of the selected subgroups, which means that one quarter of the local explanations allow to exactly explain the next recommended item for 40% of the users. Notice that providing a faithful explanation of the recommended item is very challenging. The results are much better when considering a relaxed HIT version (e.g., HIT25). As an example, a global explanation of **REBUS** on the Video dataset provides some items that are in the top 25 recommended items considering the recommendations made by **REBUS** for 80% of the users. The results for **SDSD** are similar but need more local explanations because **SDSD** provides sequences as local explanations which are more restrictive than itemset. The main difference is related to the fact that the AUC value for each global explanation is more distant from 1, because **SDSD** does not provide an explanation when the order of the items has no impact on the recommendation. For instance, this happens for around 20% of the user sequences of Visiativ with **REBUS**. However, other **SDSD** local explanations (from other users of Visiativ) can be applied, resulting in an AUC value larger than 0.8 (i.e, 0.9 with 460 **SDSD** patterns).

4.6 Conclusion

In this chapter, we presented an approach for explaining recommendations made by specific models of recommender systems. We made use of the recommendation history utilizing subgroup discovery techniques to identify the active data used for recommendation. In general, this methodological approach for generating according recommendation histories itself is model-agnostic and thus has broad applicability. For evaluating and assessing the proposed approach, we presented a set of experiments performing a deep analysis of 7 state-of-the-art models constructed on 9 datasets, including one real-world dataset. We presented an extensive discussion of our results in context. Our results show that the proposed approach provides local explanations based on the user preferences (**UPSD**) or on the order of actions performed by the user (**SDSD**) if this order actually impacts the recommendation. These explanations are easy to interpret, make it possible to identify the past items at the origin of the recommendation, and allow a better transparency for user. Moreover, we have seen that explanations can be apply to other users. Hence, the most explainable models are those for which explanations cover most of the users and have a high quality with respect to AUC and HIT_X. The most explainable models with our methods are **REBUS**, **SASrec** and **TransRec**. This result was expected as they do not model the users. Indeed, **REBUS** and **SASrec** only model items and do not exploit user embedding, while in **TransRec**, the impact of user embedding is light. Our experiments have shown that the sequential explanations generalize to other users for models that take the order of user items into account, which validates both our approach and the fact that these models make recommendations based on the sequential aspect of the user history. It is also possible to provide global explanations of a model based on a wisely selected set of local explanations.

The proposed approach focus of the explanation of the Top-1 recommended item. For future work, we aim to extend our approach in order to provide explanations for a list of recommended items. Also, we aim to investigate interpretable recommender system based on the local explanations in more details. This requires the combination of patterns of both the **UPSD** and **SDSD** methods. Further interesting options concern exploiting additional information about the time and the respective attributes of the item interactions.

Chapter 5

Implementing a Recommender System in an industrial context: The Visiativ experience

Recommender systems have received a lot of attention over the past decades, with the proposal of many models. Researchers have set up new systems by taking advantage of the most advanced and sophisticated models of Artificial Intelligence and Machine Learning, making them more powerful but also more complex. However, the level of adoption of these new models by companies is rather limited, if we exclude the largest companies in the information technology sector. Indeed, the implementation of a recommender system for a company is not trivial. It requires expertise which can be costly and undoubtedly hinders the implementation of sophisticated recommender systems for many businesses. Actually, there are still many companies that only recommend the most popular items. For others that wish to implement an advanced recommender system, the abundance of models and their complexity make the choice difficult. In this chapter, we share our experience on the implementation and the integration of a recommender system for the collaborative platform of the French company Visiativ. We present how we chose and implemented a recommender system and the strategy adopted to monitor its performances in production (A/B Testing and choice of key performance indicators). This was also a good opportunity to test in a real-world our model presented in Chapter 3.

5.1 Introduction

Collaborative filtering recommender systems offer personalized recommendations that are based on the history of users in the system and their resemblance to the histories of other users. In the 'digital era', they are a bridge between users and products of a company. As a consequence, they have received a lot of attention over the past decades, with the proposal of many models. Every company tries to take benefit from recommender system to improve users experience. However, as discussed in (Paraschakis, 2016), companies rarely go beyond the trivial most popular item lists, neighborhood-based methods, or matrix factorization models. While a sophisticated recommender system would provide relevant personalized

recommendations using interactions between users and company's resources (e.g., products, tutorials), its implementation requires expertise and is costly.

In this chapter, we share our feedback about the implementation and the integration of a recommender system within the collaborative platform of Visiativ, the company behind this work. This recommender system aims to identify Computer Aided Design (CAD) resources relevant for the users of the *myCADservices* platform. Initially, no recommender system was offered on this platform, but users could rank resources according to their popularity. Starting from the observation that the downloadable resources on this platform require a variable level of knowledge (that is to say that there is a learning logic between them), we were interested in sequential recommendation systems. We decided to limit the scope of our study to methods that are only based on user sequences to make recommendations (in our case, interactions between users and resources), without taking into account information such as content. The objective here was to ease the implementation of such a system. However, most of the existing methods of sequential recommendation based only on user sequences capture the sequential dynamics (recent actions) of a user without personalization, which limits the impact of the user's past on the recommendations. This led us to propose REBUS, presented in Chapter 3, a new metric embedding model where only items are projected and which uses frequent substrings as a proxy to learn conditional distributions of item appearance in the user's sequential history. We were able to assess and compare the performance of REBUS with other recommender systems on benchmarks used for research purposes and to show that our method outperforms state-of-the-art models, especially on sparse datasets.

In this chapter, we study the implementation of REBUS in an industrial context and the evaluation of the impact of this new recommender system from a business point of view. First, we discuss the implementation of a recommender system inside *myCADservices* platform in Section 5.2. We report an empirical study of several state-of-the-art models in Section 5.3. In Section 5.4, these empirical results are compared with the ones obtained in production, using the *A/B testing* evaluation method. Finally, Section 5.5 provides concluding remarks.

5.2 Implementation of a recommender system in Visiativ context

Implementing a recommender system into a service in production is not a trivial task and requires expertise. This is especially true when the recommendation model has been coded as a prototype for research purposes, meaning it is not ready to be directly used in production. In this section, we discuss the implementation strategy used in *myCADservices* and the difficulties encountered.

Figure 5.1 illustrates the implementation of the recommender system into myCADservices. We have chosen to use a web service that controls the recommender system. A daily script retrieves the data necessary for the recommender system. Then, it requests the new recommendations from the web service and finally updates the *myCADservices* database with these new recommendations. The *myCADservices* database and website communicate continuously to display the recommendations and track when a user clicks on a recommendation. The web service, the recommender system and the daily script are located in distinct IT environments from *myCADservices* database and website for security reasons, and also to ease their evolution.

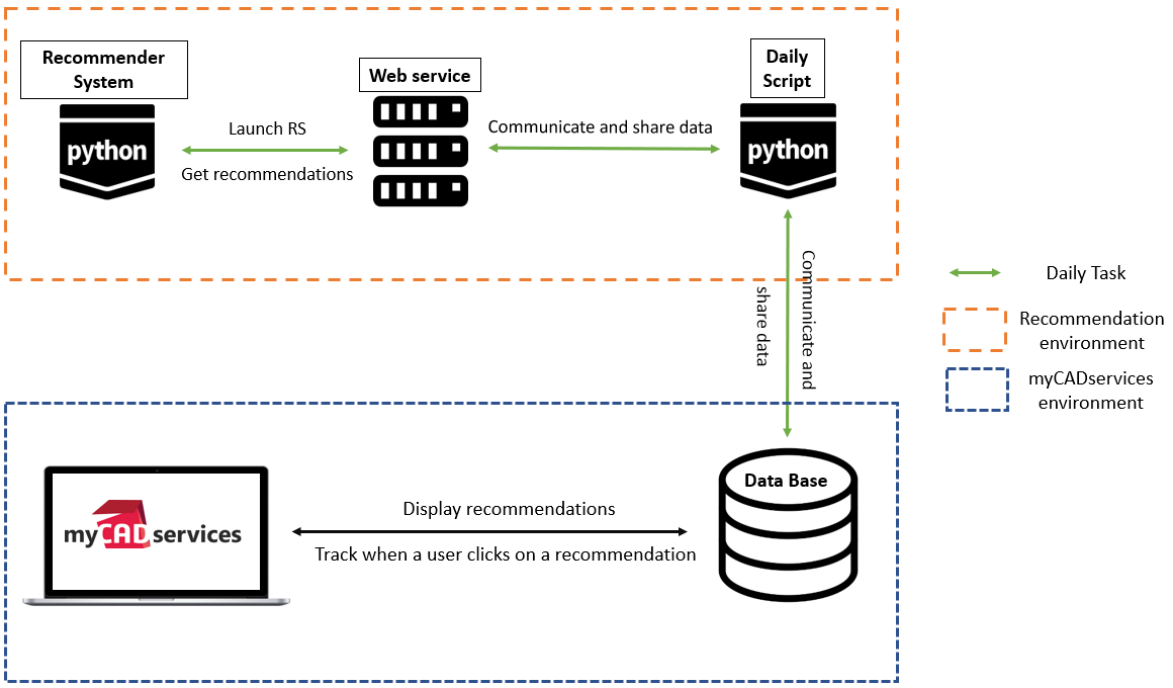

Figure 5.1: Overview of the implementation of a recommender system for myCADservices.

It is important to note that not all users are eligible to a personalized recommendations. Indeed, to have a personalized recommendation the user must have already downloaded at least 1 resource. As the *myCADservices* platform provides a multitude of services, a large proportion of users are currently not eligible. For ineligible users, the platform provides the 25 most popular resources as a recommendation.

5.2.1 The different components

To implement a recommender system inside *myCADservices*, the following components are needed:

- **A recommender system**: The ones we used are coded in Python using Tensorflow. Several recommendation models are included such as Most Popular model, BPR-MF (Rendle et al., 2009), FMC (Rendle et al., 2010), FPMC (Rendle et al., 2010), PRME (Feng et al., 2015), TransRec (He et al., 2017a), SASRec (Kang and McAuley, 2018), **REBUS** and all its variants presented in Chapter 3. After the training step, the recommender system returns the Top-25 recommendations and the evaluation of the models used in two separated csv files;

- **A web service**: It is based on Flask, a micro web service framework written in Python, and Celery, an asynchronous task queue which is based on exchange of distributed messages (in our case Redis). The web service controls the recommender system according to the requests received. There are two possible requests:

1. A *POST* request to launch the recommender system. A task identifier [1] is given in order to follow the execution. This request needs three files: (1) a file that contains all the user's histories, (2) another file that contains the list of recommender models to use, (3) and a last file that contains the list of users with their affected recommendation model. The two last files allow us to use several recommender model and perform A/B testing ;
2. A *GET* request to track the execution of the recommender system. If the task is ended, it returns a ZIP file with all the recommendations and the evaluations – using scientific metrics – of the models used, otherwise the execution state is returned (e.g RUNNING, UNKNOWN or FAILURE) with the associated error message.

- **A daily script**: Initially, *myCADservices* had to communicate directly with the web service (hence its creation). However, due to the workload of the myCADservices development team, direct communication between the web service and *myCADservices* environment could not be achieved. We then developed this script which explains why this script is in the recommendation environment as can be seen in Figure 5.1 and why the web service does not communicate directly with *myCADservices* environment. The negative effect of this change is that the recommender environment – which contains the recommender system, the web service and the daily script – has direct access to *myCADservices* database. The purpose of this script is to request data relating to the recommendation from *myCADservices* database in order to create the three files necessary for the *POST* request of the Web service and to update the database with the information given by the recommender system. This daily script is called by a task planner of CELERY;
- **The *myCADservices* database**: It stores recommendations, related data and communicates with the daily script as well as *myCADservices* website. It is through this database that the administrator pilots the recommender system. Indeed, the administrator can modify three tables that have an impact on the recommendations provided to users:
  1. A table to choose which model to use and to define its hyperparameters ;
  2. A table to manage users affectation to the recommendation model ;
  3. A table to manage A/B testing in progress.
- **The *myCADservices* website**: It displays the recommendations and notify the database when a user clicks on a recommendation.

5.2.2 Exchanges between the different components

Figure 5.2 shows the exchange of information between all components during the daily task in order to update the top 25 recommendations for each eligible user. There are six steps to accomplish this daily task:

[1] This task identifier corresponds to the *ID* task given by CELERY.

1. The *daily script* requests the *myCADservices* database to get the data needed for the *recommender system*. This includes user's histories, recommendation models to use, and the assignment of users to the recommendation model;

2. The *daily script* sends the obtained data via a POST request to the *web service*. The *web service* checks the data and, if the data is in the correct format, it returns the *ID* task to the *daily script*;

3. The *web service* launches the *recommender system*;

4. Every 30 seconds the *daily script* requests the recommendations via the GET request using the *ID* task until the *recommender system* ends;

5. When the *recommender system* ends, the *web service* returns the recommendations and evaluations of the models to the *daily script*;

6. The *daily script* pre-processes the recommendations and evaluations, and sends them to the database in order to update the top 25 recommendations for each eligible user.

As we can see in Figure 5.1, *myCADservices* database and website communicate continuously to display recommendations to users and also to track back when a user clicks on a recommendation.

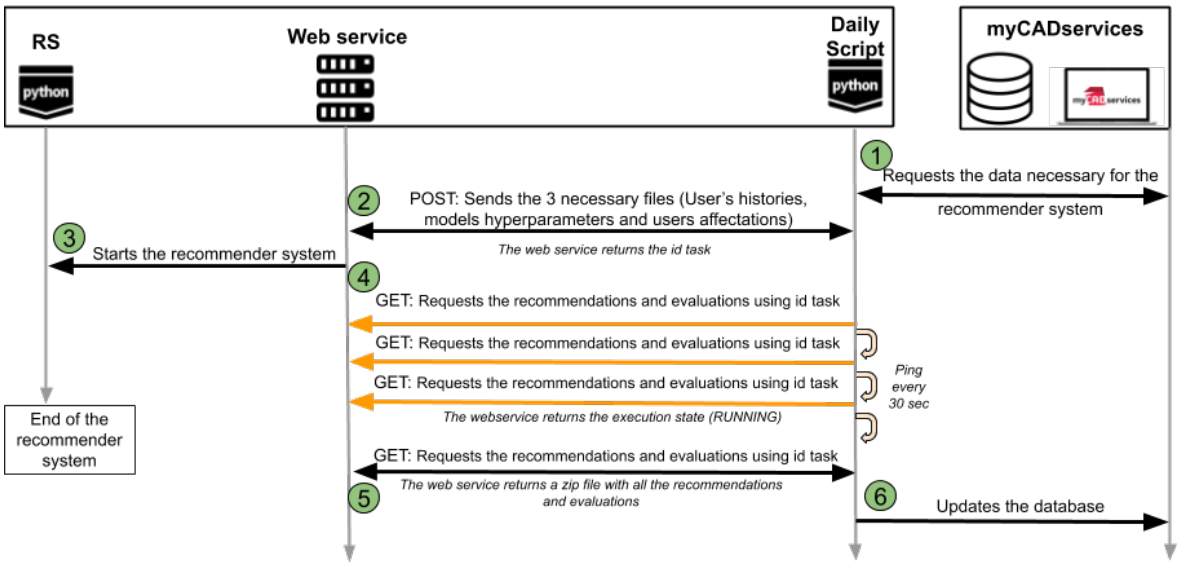

Figure 5.2: Overview of the exchanges between the different components during the daily task.

5.2.3 Which model to choose?

The profusion of recommender systems makes it difficult to choose the system adapted to a particular context. To make an informed choice, we adopted a two-step methodology. The first step was to perform an empirical study on several datasets, including *myCADservices*,

to compare several recommender systems. The second step consisted in studying a small number of recommender systems selected during the first step, within the *myCADservices* environment, in order to compare and measure their impact in this context.

5.3 Empirical study

We carry out this empirical study in two steps. First, we compare different recommendation models across several datasets, including the *myCADservices* dataset. This allows us to identify the best performing recommendation models. In a second step, we carry out a deep study on the *myCADservices* dataset with the models that seem to us to be the most promising from the first step.

5.3.1 General study

In the Chapter 3, we present a thorough empirical study of 10 recommendation models – including our model **REBUS** – over 12 datasets – including *myCADservices* dataset. The performances of the different methods on each dataset is reported in Section 3.4.4. We can observed that **REBUS** outperforms other models on most of the datasets, having the best AUC value. **REBUS** works well on all other metrics as well. Additionally, we can see that **SASRec** is a promising model, it performs well on *myCADservices* dataset and have the second best AUC value in overall. **TransRec** is also a promising model as it performs well on *myCADservices* dataset having the best NDCG_25 and NDCG_50. It also has the second best value for HIT_25, HIT_50, NDCG_25 and NDCG_50 in overall.

This study saved us time, because it led us to abandon the underperforming models. From it, we retain the three models ***REBUS***, **TransRec** and *SASrec* as they are the most efficient models. In addition, we keep $\textbf{REBUS}_{XMC}$, a variant of **REBUS**, which shows good performance (see Tables 3.8 and 3.9) and the Most Popular model, called hereafter **POP**, as it is the model we propose to use for not eligible users (those who have not yet downloaded a resource).

5.3.2 Focus on the *myCADservices* dataset

Now let us present an in-depth study of the five previously select recommender systems on the *myCADservices* dataset.

Datasets: To evaluate the performance of the selected models, we consider 3 different configurations of the *myCADservices* dataset obtained by modifying two parameters during the preprocessing: userMin and itemMin that are respectively the minimum number of interactions that a user and an item must have to be retained in the dataset. The main characteristics of the 3 different configurations of the *myCADservices* dataset are reported in Table 5.1. The control of the minimum number of interactions that an item and a user must have, allows us to evaluate the performance of the 5 selected models for different configurations of *myCADservices* dataset. We can see from Table 5.1 that if we decrease userMin and itemMin, the numbers of considered users and items increase. The advantage of using a smaller userMin is that it increases the number of eligible users who will have a personalized recommendation. Notice that, the dataset named *Visiativ* in Chapter 3 and 4 comes from

myCADservices dataset. The difference is that we update the *myCADservices* dataset for this study in order to have the downloads from November 2014 to December 2019.

Table 5.1: Main characteristics of *myCADservices* dataset with different preprocessing. USERMIN and ITEMMIN are the minimum number of interactions a user and an item must have to be considered. The first row is the initial dataset without preprocessing.

USERMIN	ITEMMIN	#Users	#Items	#Actions	#A/#U	#A/#I	Sparsity
1	1	6274	1004	32081	5.11	31.95	99.49%
3	3	2822	804	27317	9.68	33.98	98.80%
4	4	2191	731	25270	11.53	34.57	98.42%
5	5	1812	684	23637	13.04	34.56	98.09%

Experimental settings: We apply the same experimental setting as in Section 3.4.3 Chapter 3. That is, the models are evaluated on the most recent item. The second most recent item is used for tuning model hyperparameters, and other items are used for training models. Model performance is assessed by three metrics widely used in sequential recommendation: (1) the Area Under the ROC Curve (AUC) that estimates how high each user's ground-truth item is ranked on average; (2) the Hit Rate at position X (either 5, or 25) that returns the average number of times the ground-truth item of each user is ranked in the top X items; (3) the Normalized Discounted Cumulative Gain at position X (either 5, or 25) that is a position-aware metric which assigns larger weights to higher positions.

We modify the grid search in order to explore more combinations of hyperparameters. We fix the following parameters for all models: The learning rate is set to 0.001, the batch size to 128 and we stop the training if there is no improvement of the AUC validation for 250 epochs. We compute the AUC validation every 25 epochs. The dimension of the learned latent vectors k are taken in $\{10, 20, 30, 40, 50\}$. All regularization hyperparameters are taken in $\{0, 0.001, 0.01, 0.1, 1\}$. For models with other hyperparameters we have tried those given by the authors. Regarding specific hyperparameters of **REBUS**, $\alpha \in \{0.1, \ldots, 1.0\}$ and $\gamma \in \{0.0, 0.1, \ldots, 1.0\}$.

Performance study: The performances of the different methods on every dataset are reported in Table 5.2[2]. **REBUS** and **REBUS**$_{XMC}$ have almost the same performance for each metrics and outperforms other models having the best AUC value and HIT_25 value. **TransRec** shows fair performance having the best HIT_X and NDCG_X value when USERMIN and ITEMMIN are equal to 3 and 5. Also, **SASRec** outperforms **TransRec** on AUC value and has fair performance on HIT_X and NDCG_X.

For all models, we can observe that AUC performance increases when USERMIN and ITEMMIN are small while HIT_X and NDCG_X tend to slightly decrease. This is interesting because it shows that we can take a small value of USERMIN and ITEMMIN (e.g USERMIN and ITEMMIN equal to 3) while preserving good performance. This configuration allows us to maximize the number of eligible users. This is why we have chosen USERMIN and ITEMMIN equal to 3 to be the configuration tested in the industrial study.

[2]We only show HIT_5, HIT_25, NDCG_5 and NDCG_25 in this table. HIT_10, HIT_50, NDCG_10 and NDCG_50 are reported in Appendix 7 Table 7.10.

Notice that we do not evaluate performance of models with userMin and itemMin under 3 because a user needs at least 3 items to be evaluated in our experimental setting. According to the performance observed when userMin and itemMin equal to 3, we have selected 4 models that have been evaluated in the myCADservices environment:

1. The first two models are **REBUS** and **REBUS**$_{XMC}$ because they outperform other models on AUC and have comparable performance on other metrics;

2. The third model is **SASRec**. We can see that **TransRec** is better than **SASRec** and **REBUS** on HIT_X and NDCG_X. However **SASRec** and **REBUS** do not suffer from the cold-start problem, since they can make recommendations to users who have interacted only with a single item of the system as discuss in Subsections 3.3.8 and 3.4.5. For instance, **SASRec** and **REBUS** are able to make a recommendation for 6252 users (number of users who interacted with a single item that has at least 3 interactions in the dataset) where **TransRec** is able to make a recommendation for only 2822 users (number of users who have at least 3 interactions with items that also have at least 3 interactions);

3. The last model is **POP** because it is the model we propose to use for ineligible users.

5.4 Industrial study

We report here a thorough industrial study to evaluate the selected models into the *myCADservices* platforms. The aim here is twofold : (1) to confirm that the results obtained in the empirical study are still observable in the real-world environment, and (2) to measure the impact of the recommendation system on *myCADservices*.

To confirm that the hypotheses made in the empirical study are relevant for our application (i.e Model A outperform Model B), we used the A/B testing methodology. In our case, this involves to randomly split eligible users into two groups and affect them to a different recommendation model. The two groups have the same interface: a banner that displays 3 recommendations. Furthermore, the user can navigate through the Top-25 recommendations. Due to the relatively low activity of the platform (about 15 downloads per day) and in order to limit some effects that could bias our study (e.g. one-off increased activity, novelty effect, etc.), we chose to perform this study over 6 months. Consequently, it was not possible to compare all models. Our analysis is based on the following business indicators:

- Evolution of users (total number of users and percentage of eligible users);
- Evolution of downloads (total number and daily progression);
- Evolution of the performance of scientific metrics for sequential recommendation;
- Evolution of clicks on recommendations (total number, daily progression, and conversion rate from a click on a recommendation to the download of the recommended resource).

Table 5.2: AUC, HIT_5, HIT_25, NDCG_5 and NDCG_25 for *myCADservices* dataset with different preprocessing. The two numbers after *myCAD* are respectively the value of USERMIN and ITEMMIN in the Table 5.1. Best obtained results are in bold, second best obtained results are underline. The last column is the average performance on these 3 different preprocessing.

	Metric	myCAD_3_3	myCAD_4_4	myCAD_5_5	Avg(All)
POP	AUC	0.8255	0.8128	0.8110	0.8286
	HIT5	15.09%	15.05%	15.35%	15.36%
	HIT25	33.71%	33.67%	34.46%	34.58%
	NDGC5	9.10%	9.11%	9.53%	9.32%
	NDGC25	14.15%	14.13%	14.66%	14.47%
TransRec	AUC	0.8975	0.8946	0.8967	0.9007
	HIT5	**32.32%**	28.68%	**34.62%**	**30.25%**
	HIT25	55.01%	55.86%	**58.37%**	55.32%
	NDGC5	**25.21%**	20.78%	**26.31%**	**22.49%**
	NDGC25	**31.44%**	28.19%	**32.78%**	**29.34%**
SASRec	AUC	0.9041	0.8995	0.8993	0.9060
	HIT5	28.04%	**30.60%**	31.64%	30.03%
	HIT25	53.51%	55.63%	56.71%	55.10%
	NDGC5	20.58%	**22.51%**	24.25%	22.40%
	NDGC25	27.55%	**29.36%**	31.12%	29.26%
REBUS	AUC	0.9060	0.9020	0.9012	0.9070
	HIT5	28.33%	28.45%	30.43%	29.00%
	HIT25	**56.23%**	56.95%	57.26%	56.13%
	NDGC5	19.91%	20.07%	21.25%	20.65%
	NDGC25	27.56%	27.92%	28.68%	28.11%
REBUS$_{1MC}$	AUC	**0.9065**	**0.9021**	**0.9016**	**0.9073**
	HIT5	28.18%	30.05%	31.75%	30.05%
	HIT25	55.98%	**57.55%**	57.65%	**56.25%**
	NDGC5	19.98%	21.02%	22.28%	21.45%
	NDGC25	27.61%	28.57%	29.40%	28.66%

Each of these indicators are calculated on each group of the A/B test and are considered to be useful to measure the impact of the recommender system on *myCADservices*.

It is important to notice that we have two categories of users:

- **The eligible users** who downloaded at least one resource. They are randomly assigned either to group A or to group B of the tests.
- **The ineligible users** who have not yet download any resources. For them, the platform provides 25 most popular resources without personalization as a recommendation, later

call *Default_RS*. *Default_RS* are obtained by the **POP** model.

We first begin by analyzing the impact of the recommender systems on *myCADservices*. Then, we report an analyze of two A/B tests: the first one compares ***REBUS*** and **POP** models; The second one compares $\boldsymbol{REBUS}_{1MC}$ and **SASRec**.

5.4.1 Impact of recommender systems

To analyze the impact of recommender systems on *myCADservices*, we report the business indicators starting 6 months before the beginning of the first A/B test.

In a first step, we analyze the impact of the implementation of a recommendation system on the whole *myCADservices* platform using two indicators: the evolution of the number of users and the evolution of the number of downloads. As can be seen on the left of figures 5.3 and 5.4, the number of users and the number of downloads increase steadily during the observation period, that is to say before and during the evaluation periods of the recommendation systems. When we look closely at the progress of these two indicators (On the right of figures 5.3 and 5.4), we observe a decrease in the average number of new users and downloads per week for *A/B Test 2*. For instance, there were on average 160 new users per week (resp. 110 downloads per week) before using a recommender system, compared to 110 new users per week (resp. 75 downloads per week) at the end of the *A/B Test 2*.

From these two indicators, we can conclude that the recommender system had no impact on the platform overall activity. Note that the significant drop of activity observed for *A/B Test 2* is potentially linked to the COVID-19 crisis. Nevertheless, this can be explained by the fact that the recommender system is only present in the *resource center* which is a small part of the whole *myCADservices* platform. From Figure 5.5 (right), we notice that a large part of the users do not use the *resource center* (today more than 90% of users have never downloaded any resources).

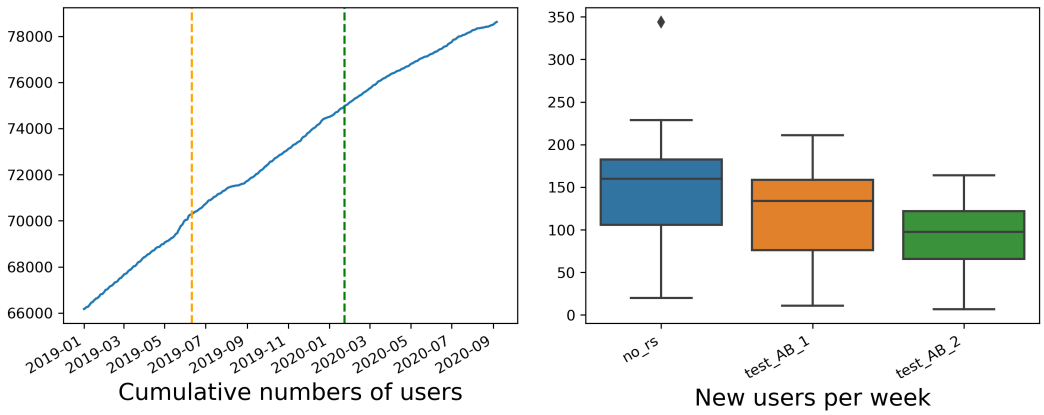

Figure 5.3: Evolution of the cumulative number of users (left) and number of new users per week (right). Orange (resp. green) line refers to the start of A/B Test 1 (resp. A/B Test 2).

In a second step, we analyze the impact of the implementation of a recommendation system in the *resource center* of *myCADservices* platform. From the left of Figure 5.5, we can see the evolution of the cumulative numbers of clicks on recommendation for the

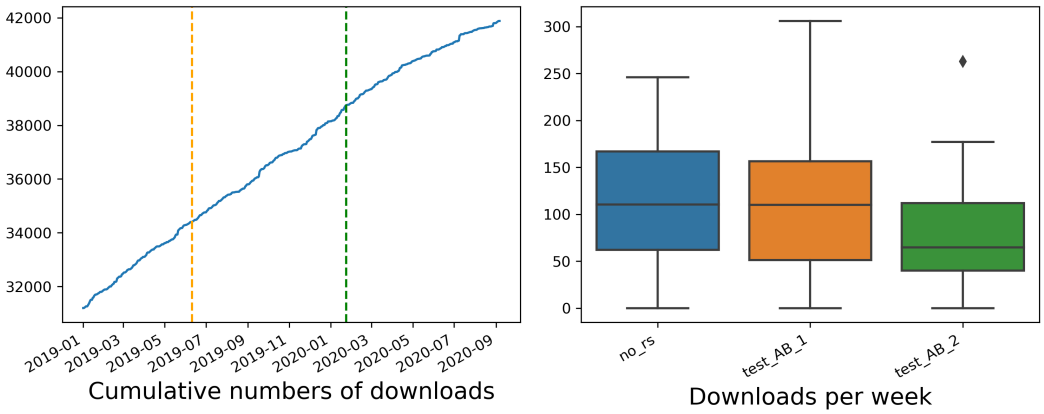

Figure 5.4: Evolution of the cumulative number of downloads (left) and number of downloads per week (right). Orange (resp. green) line refers to the start of A/B Test 1 (resp. A/B Test 2).

eligible users, of ineligible users and the combination of both type of users. There were 1395 clicks on recommendations within 14 months. About 60% of those clicks are made from the personalized recommender system (models in A/B testing) provided to eligible users which represent 8.5% of *myCADservices* users. The other 40% of those clicks comes from non-eligible users – which represent 91.5% of *myCADservices* users – who clicked on the recommendations provided by *Default_RS* (*Default_RS* are obtained by the **POP** model). This confirms that a large part of the users do not use the *resource center* and more than the half of the activity of the *resource center* is due to 8.5% of *myCADservices* users (eligible users). If we compare the two A/B tests using Figure 5.6, we can notice that there is no significant difference and the decrease activity related to COVID-19 is not visible here. In fact, there are more clicks in overall for *A/B Test 2* than for *A/B Test 1* (728 against 663). We have also calculated the conversion rate of a click on a recommendation into a download of the related resource. We can observe that the two A/B tests have almost the same conversion rate, 38.6% for the *A/B Test 1* and 41.7% for the *A/B Test 2*. However, the conversion rate is much lower for the *Default_RS*, 27.4% which it is normal because the recommendation aren't personalized. Nonetheless, this allows to convert more than a quarter of the ineligible users who clicked on a recommendation to eligible user.

With these observations, we can conclude that the activity related *resource center* is mainly due to regular users of the resource center (eligible users) and the recommendation system have a positive impact on them. Indeed, if we consider all recommender systems of both test A/B, there are 487 downloads due to them. Note that, for the same period there were 7477 downloads. Although, personalized recommender systems are more effective (i.e. More clicks on recommendation) and provide better recommendations (i.e. Better conversion rate) than *Default_RS*. However, it seems advisable to provide recommendations via *Default_RS* on the homepage of the platform to promote the resource center to ineligible users.

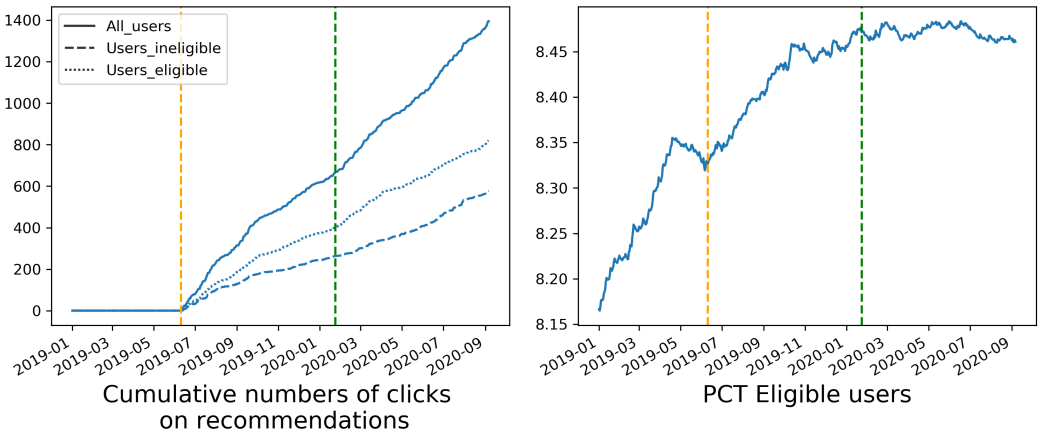

Figure 5.5: Evolution of the cumulative number of clicks on recommendations (left) and evolution of the percentage of eligible users (right). Orange (resp. green) line refers to the start of A/B Test 1 (resp. A/B Test 2).

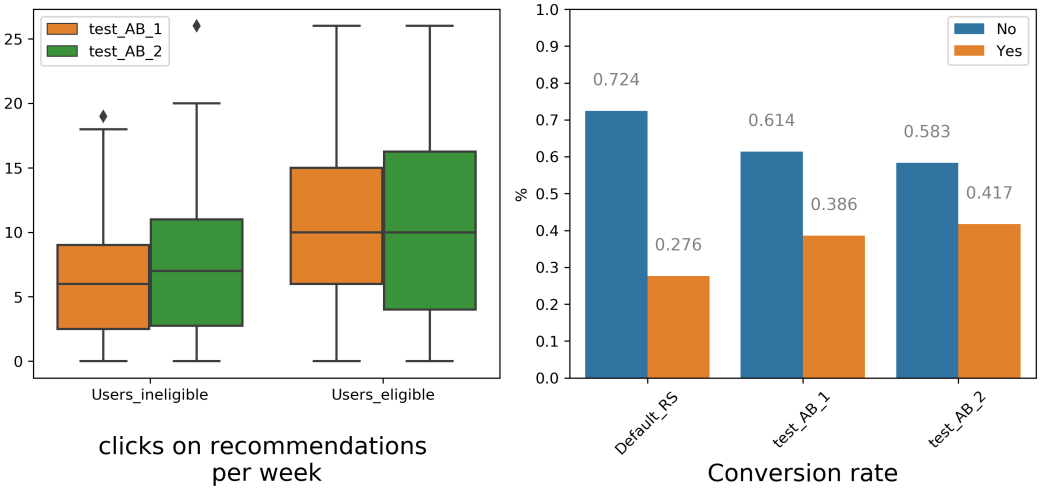

Figure 5.6: Number of clicks on recommendations per week (left) and conversion rate of clicks to downloads (right).

5.4.2 A/B Test 1: REBUS versus POP (2019/06/10 – 2020/01/23)

In this first A/B test, we have compared ***REBUS*** and **POP** model (model that recommends the most popular items) during 227 days. The aim of this first test is to confirm the superiority of **REBUS** over **POP** observed in empirical study. And also to see if choosing **POP** as the default recommender system for ineligible users is relevant.

Before starting to compare with the business indicators for the two models of the *A/B Test 1*. We recorded daily, and with the same empirical study evaluation protocol as Section 5.3, the performances of the two models with the scientific metrics: AUC, HIT_X and NDCG_X. The aim was to monitor the "scientific" performance of the models and to ensure that the chosen hyperparameters allow the model to perform well over time. From Figure 5.7, we can

see that **REBUS** outperforms POP on all the scientific metrics during the entire A/B Test 1. Furthermore, there is no significant variation of these metrics of both models and the gap remains constant. These results show that we can keep the hyperparameters during a long time period while maintaining the same level of "scientific" performance. This is certainly due to the activity of the platform and we can only apply this conclusion to our case study.

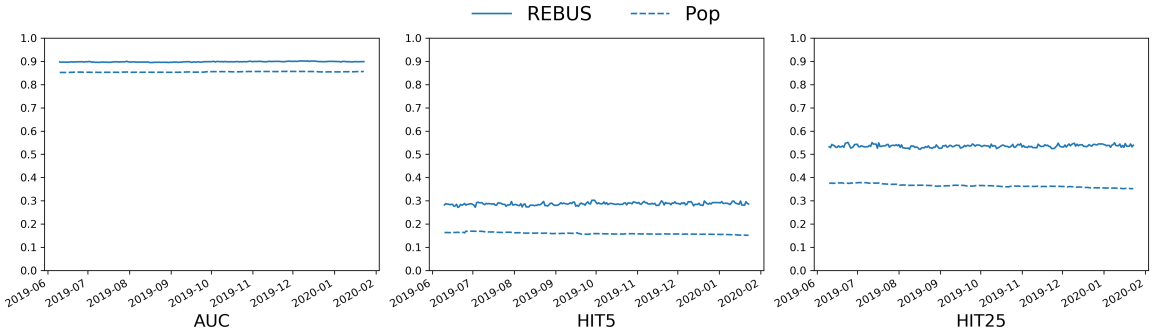

Figure 5.7: Evolution of the "scientific" performance (AUC, HIT_5 and HIT_25) of **REBUS** and **POP** during the entire *A/B Test 1*.

Now that we know that "scientific" performance persists over time for both models, we can then analyze the business indicators to compare them. To our great surprise, regarding the number of clicks on the recommendations shown in Figure 5.8, the recommendation model **POP** and **REBUS** gets almost the same number of clicks, 205 clicks for **POP** against 195 for **REBUS**. It can also be noted that **POP** had taken a quite significant lead at the middle of the *A/B Test 1* but in the long term **REBUS** filled the gap. Indeed, **REBUS** seems more regulated on the number of clicks on the recommendations per week (see center of Figure 5.8). Moreover, the conversion rate of clicks to downloads is also very similar, 38.5% for **POP** against 39.3% for **REBUS**.

In addition to this, we can see in Figure 5.9 that users with **REBUS** tend to click on the first 3 recommended resources displayed in the banner[3] while users with **POP** model click more on recommended resources beyond the third rank. The first three recommendations of ***REBUS*** seem to be more relevant than that of **POP**.

Nevertheless from these observations, the superiority of **REBUS** shown in the empirical study (see Section 5.3) is not evident here. This can be explained by the fact that the functionality was new. Indeed, during the implementation of the *A/B Test 1* we have modified the user interface to add a banner that displays the recommendations. It would have been wise to wait a few months before the start of this A/B test to remove the novelty aspect which certainly biased a bit the beginning of the *A/B Test 1*. Despite this, these results confirmed our choice to use the most popular resources for ineligible users. Moreover, it also seems relevant to provide two types of recommendations to eligible users: (1) recommendations based on a personalized recommendation system (e.g. **REBUS** or SASRec) and (2) recommendations from the most popular resources. This strategy is also adopted in many platforms such as Netflix, Youtube or Spotify.

[3]The banner displays 3 recommendations and users can navigate through the Top-25 recommendations

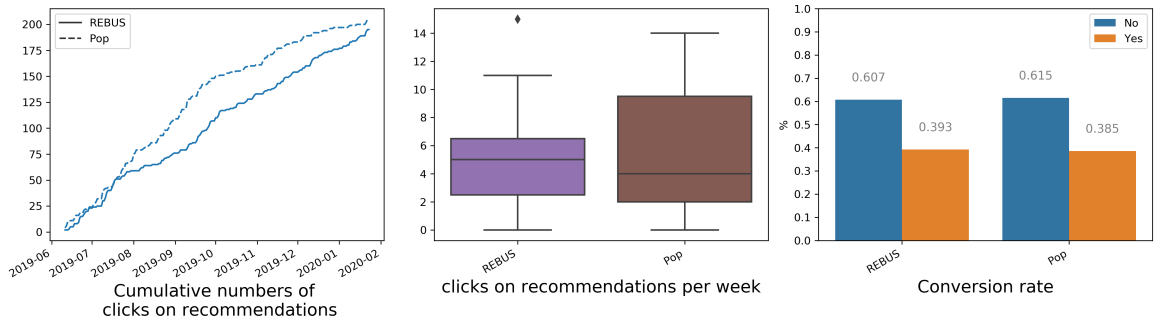

Figure 5.8: Evolution of the cumulative number of clicks on recommendations (left), number of clicks on recommendations per week (center) and conversion rate of clicks to downloads (right) for *A/B Test 1*.

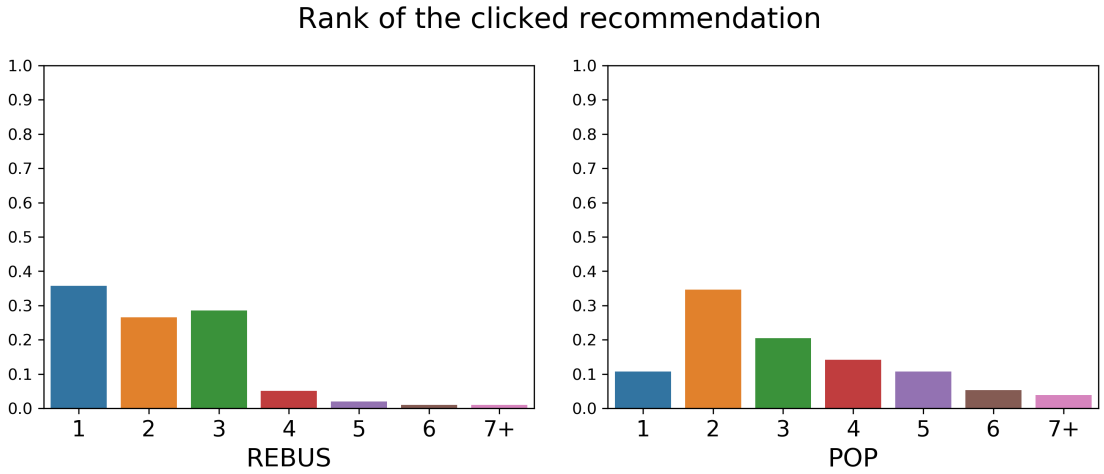

Figure 5.9: Repartition of the rank of the clicked recommendation for **REBUS** (left) and for Pop (right) during the *A/B Test 1*. The rank is the position of the recommended resources in the banner displayed to users.

5.4.3 A/B Test 2: REBUS$_{1MC}$ versus SASRec (2020/01/23 – 2020/09/06)

In this second A/B test, we have compared ***REBUS***$_{1MC}$ and **SASRec** during 227 days. The aim of this second test is to confirm the small superiority of **REBUS** over **SASRec** observed in empirical study.

Similar to *A/B Test 1*, we have monitored the "scientific" performance of ***REBUS***$_{1MC}$ and **SASRec** using AUC, HIT_X and NDCG_X. Like the results observed for *A/B Test 1*, there is no significant variation for the scientific metrics and the two models are equivalent. These results confirm our conclusion from *A/B Test 1* that we can keep the hyperparameters chosen for a model and have the same performance for a long time. In addition to this, from the Figure 5.10 we changed the Y axis scale in order to zoom and see which models stand out. On the AUC value, we can see that ***REBUS***$_{1MC}$ outperforms **SASRec** while **SASRec** ***REBUS***$_{1MC}$ outperforms on the HIT_5. For HIT_50, both models have the same performance.

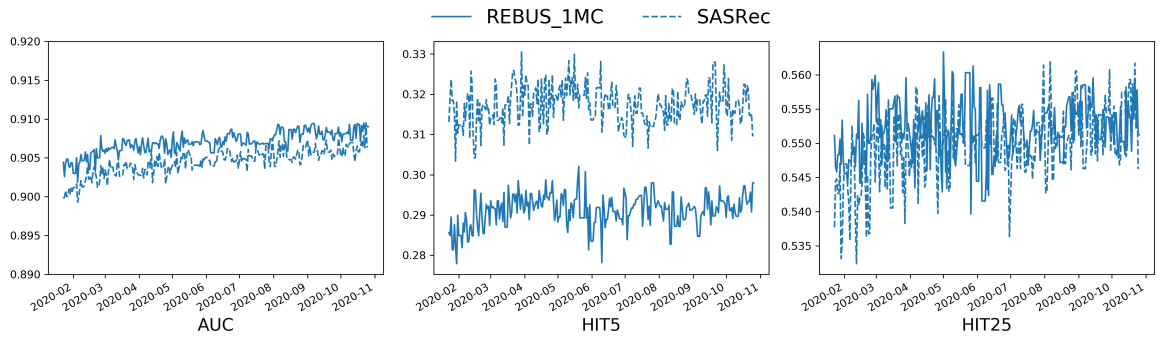

Figure 5.10: Evolution of the "scientific" performance (AUC, HIT_5 and HIT_25) of ***REBUS***$_{1MC}$ and **SASRec** during the entire *A/B Test 2*.

Now that we confirm that "scientific" performance persists over time for both models, we can then analyze the business indicators to compare them. In Figure 5.11, we see that ***REBUS***$_{1MC}$ gets more clicks than **SASRec** (225 against 190). Both models have almost the same conversion rate, 41.3% for ***REBUS***$_{1MC}$ compared to 42.1% for **SASRec**. From Figure 5.12, we can observe that users of ***REBUS***$_{1MC}$ and **SASRec** have almost the same behavior and click mostly on the 3 first recommended resources. More than 70% of clicks come from the first three recommendations, whether for ***REBUS***$_{1MC}$ or **SASRec**. The banner's interface has been modified to facilitate the exploration of the recommended resources. This explains the minor difference observed between *A/B Test 1* and *A/B Test 2* for ***REBUS*** and ***REBUS***$_{1MC}$.

This A/B test shows that it remains complex in practice to determine the best recommendation model even if ***REBUS***$_{1MC}$ seems slightly ahead.

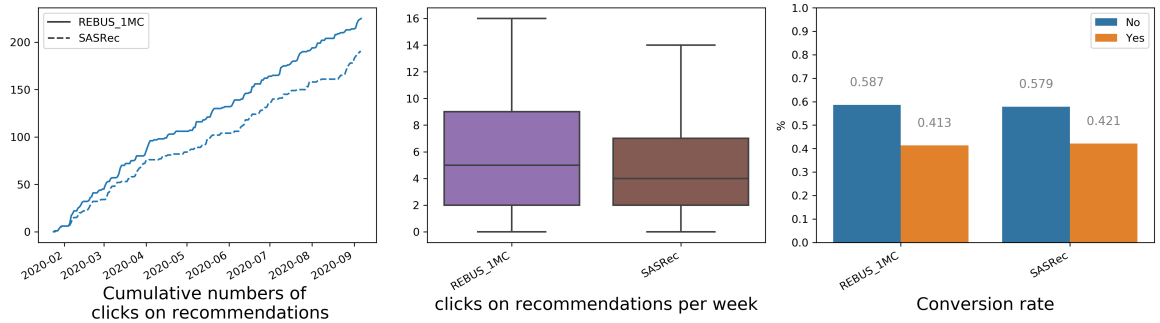

Figure 5.11: Evolution of the cumulative number of clicks on recommendations (left), number of clicks on recommendations per week (center) and conversion rate of clicks to downloads (right) for *A/B Test 2*.

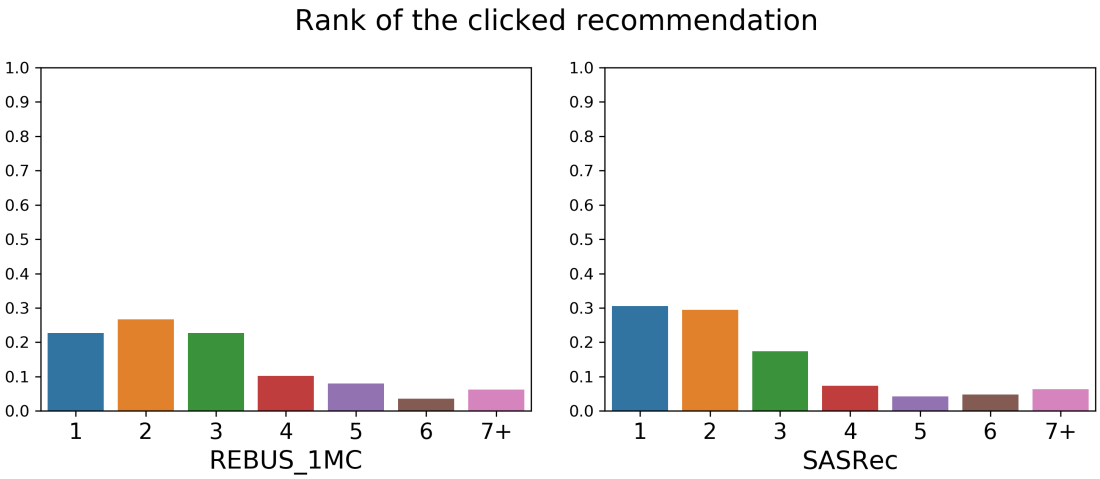

Figure 5.12: Repartition of the rank of the clicked recommendation for $\boldsymbol{REBUS}_{1MC}$ (left) and for **SASRec** (right) during the *A/B Test 2*. The rank is the position of the recommended resources in the banner displayed to users.

5.5 Conclusion

We reported our experience about the deployment of a recommender system in an industrial context (the French SME Visiativ). Recommender systems are essential to provide personalized and relevant recommendations to users. Thus the choice of the system must be carefully made. Nevertheless, the comparison between different models is difficult. There are numerous recommendation models and each of them has it pros and cons. We used 3 different metrics (i.e., AUC, HIT_X and NDCG_X). No model outperforms all others for every dataset and metric. Note that, other evaluation metrics could be used such Precision, Recall, MAP and MRR. Furthermore, as discussed in (Said and Bellogín, 2014) and in Section 2.4, there are several possible evaluation strategies. Given all these elements, determining which is the best model is extremely difficult and requires expertise to choose the metrics and the evaluation strategy fitting to the problem. We identified some good candidates based on their performances in our empirical studies on several benchmarks. A further study in the industrial application reports some mitigated results: while **REBUS** model significantly outperforms the Popularity-based model in the previous study, its superiority is not obvious when considering business indicators (i.e., #clicks, conversion rate). This makes the identification of the good model even more complex. However, this does not mean that we cannot be confident in the results of the empirical evaluation on benchmarks. Many other factors have to be considered when launching an evaluation in an industrial context: the interface, the advertising campaign and the attraction of novelty (users tend to click when something is new whatever the model), the users of the platform. Moreover, we report that the implementation of the recommender system in the *resource center* of *myCADservices* platform had a positive impact despite the decline in activity related to COVID-19, for instance 6% of downloads are due to the recommendation system.

In future work, it would be interesting to use multi-armed bandit instead of traditional A/B testing. The advantage of the multi-armed bandit compared to A/B testing is that it can

simultaneously explore the performance of several models and gradually favor the exploitation of the best one. This could certainly reduce the duration of the test (Scott, 2015) and allows to iterate more quickly. In the future, potential test could be done with either a new model or a new user interface. We also plan to implement our method of explanations presented in Chapter 4 on the *myCADservices* platform and see if this has a positive impact on the recommendation system.

Chapter 6

Conclusion

6.1 Summary

The work presented in this thesis aims at addressing the problem of sequential recommendation. First, regarding this problem, we have proposed in Chapter 3 **REBUS**, that uses a simple architecture based on a metric embedding model. **REBUS** is a unified metric model that only embeds items in a Euclidean space in order to learn a representation of the user preferences and sequential dynamics. While other existing models only use fixed-order Markov Chains, regardless of the users and their considered items, one of the strengths of **REBUS** is that it uses frequent sequences to identify the part of user history that is the most relevant for recommendation. These sequences are then used to estimate Markov Chains of variable orders, adapted to the user's profile. Also by using frequent sequences, we can have some insight to explain a recommendation. Despite its simple architecture, we have demonstrated in an extensive empirical study on numerous datasets that **REBUS** outperforms the state-of-the-art sequential recommendation models on sparse datasets – which is the most standard configuration in real cases – and has comparable performances on dense datasets. This empirical study also provides evidences that **REBUS** has good performance even if the user histories are very short (that corresponds to cold-start users). Finally, **REBUS** is easily customizable, as it is based on a trade-off between sequential dynamics and user preferences fixed by the hand-user with the hyperparameters γ, and also as it relies on the use of a set of sequences that can be automatically learnt or replaced by any other Markov Chains (of fixed or variable order). These elements makes it possible to adapt **REBUS** to different contexts, such as sparse or dense datasets.

Second, thanks to the link of this research with the company Visiativ, we were able to implement and test the performance of **REBUS** on the *myCADservices* platform (Chapter 5). The objectives of this study were to determine the efficiency of a recommender system in a real environment, and using A/B testing methodology, to confirm that the hypotheses made in the empirical study are relevant for our application (i.e Model A outperforms Model B). In a general way, the implementation of the recommender system in the *resource center* of *myCADservices* platform had a positive impact on the activity of the platform where 6% of downloads are a direct consequence of the recommendation system. On the other side, the A/B tests show some mitigated results. For the first A/B test, despite the fact that **REBUS** significantly outperforms the Popularity-based model in the empirical study, its superiority

was not obvious when considering business indicators. For the second A/B test, the slight superiority of **REBUS** over **SASRec** observed in empirical study was visible only on the number of clicks. These two A/B tests show that assumptions made in the empirical study are not always satisfied in real case studies using business indicators. However, this does not mean that we cannot be confident in the results of the empirical evaluation on benchmarks. Many other factors have to be considered when launching an evaluation in an industrial context: the interface, the advertising campaign and the attraction of novelty (users tend to click when something is new whatever the model), as well as the users of the platform. New A/B tests will be launched on *myCADservices* platform in order to test new recommendation models or even to modify the user recommendations display interface. To speed up test iterations, A/B tests could be replaced by multi-armed bandit which have the advantage of gradually favoring the best solution while simultaneously exploring the performances of the two solutions and exploiting the best results.

Finally, regarding the lack of works that attempted to explain recommender systems, we proposed, in Chapter 4, a model-agnostic approach for explaining recommendations made by sequential recommendation models based on implicit feedback. We used the recommendation history using subgroup discovery techniques to identify the active data used for recommendation. The results of our experimental study demonstrate that our method provides (1) explanations that are easy to interpret providing itemset or sequence of items at the origin of the recommendation, and (2) explanations that are generalizable (i.e. explanations can be applied to other users). Moreover, it is also possible to provide "global explanations" (i.e. a set of local explanations that explains most of the recommendations) of a model based on a wisely selected set of local explanations. It could be interesting to implement our approach for explaining recommendations in *myCADservices* platform and see if this has a positive impact on the recommendation system.

6.2 Perspectives

The work presented in this manuscript opens several research avenues toward both relevant and explainable recommendations.

6.2.1 Improving REBUS

Learning and personalizing the trade-off between long-term and short-term dynamics.

A first easy way of improvement for **REBUS** would be to learn the trade-off (hyperparameter γ) between long-term and short-term dynamics. The advantage of learning γ is that it facilitates the configuration of **REBUS** because the trade-off between long-term and short-term dynamics would be automatic. To go even further we could learn two parameters for the trade-off between long-term and short-term. The first one would be a global parameter to all users (γ) and the second one would be a vector of personalized parameters associated to each user (γ_u a vector of size $|U|$). This would allow **REBUS** to give more weight to the sequential dynamics in the case of a new user and potentially give more weight to the user preferences in the case of a regular user with a long history.

Improving the selection of items for the user preferences and the sequential dynamics.

Another direction of improvement for **REBUS** would be to enhance the selection of items in the user preferences part. Currently to model user preferences, **REBUS** uses all items from the user's history and each item has the same weight. However, it seems reasonable to think that the general tastes (user preferences) of regular users evolve over time. One proposition that may be interesting to investigate would be to find a function that assigns a weight to items based on frequency of their occurrences and their distribution over the user's history. However, it is necessary to make sure that this function would capture different phenomenon from the sequential part of **REBUS**. In the same line of work that the one used to identified short-term dynamics, where frequent sequences are used to identify the part of user history that best match the sequential dynamics, a more challenging proposal could be to identify one or several itemsets that best represent the current user tastes using pattern mining techniques.

Eventually, **REBUS** represents the sequential dynamics with a single sequence. It could be interesting to take into account several sequences, provided by different pattern mining techniques. We will have to deal with several sets of representative sequences of users' histories and choose the most relevant ones from each set. At the end, all of chosen sequences will have to be carefully unified in order to not overwhelming the user preferences part of **REBUS**.

Create a content/context-aware recommender system using the architecture of REBUS.

Our model, **REBUS**, only used the implicit feedback to predict the next user item. If content and/or context information is available, it would be interesting to use them in order to improve the performance of recommendations. It would be interesting to create a new model with the same architecture as **REBUS**, which embeds items in a Euclidean space, but where context and content information would be embedded too. For instance, if we have the categories of items, we can embed this information in order to find out which item category the user prefers. The main challenge is to find a way to learn and unify different embedding in a metric model[1].

6.2.2 Extend our work on explanation of sequential recommendation

Explanation for a list of recommended items.

One of the limitations of our proposed approach in Chapter 4 is that we only explain one recommendation at a time (in our case the Top-1 recommendation) using subgroup discovery. Indeed, subgroup discovery provides an overview on the relationships between a *target variable* (or target attribute) and *explaining variables*. However, a recommender system generally does not return a single item, but gives a larger order set of items. Therefore, it would be interesting to extend our approach in order to be able of explaining a set or a sequence of recommendations. We can identify four possible extensions of explanations:

1. An un-ordered set of items that explains an un-ordered set of recommended items;

[1]Without using Factorization Machine as in (Pasricha and McAuley, 2018).

2. An un-ordered set of items that explains a sequence of recommended items;
3. A sequence of items that explains an un-ordered set of recommended items;
4. A sequence of items that explains a sequence of recommended items.

For the first two extensions, it is possible of using aggregation functions (e.g. weighted sum for the first one and Kendall rank correlation coefficient for the second one) to keep the protocol of **UPSD**, i.e. perturbation using item suppression and use of SD-MAP. For the third extension, it is possible of using aggregation functions to keep the protocol of **SDSD**, i.e. perturbation using item permutation and use of SeqScout algorithm. The challenge is in the aggregation function which will have to reflect the quality of the list of recommended items for each perturbation in order to determine the relevant itemsets or relevant sequence of items. The fourth extension is more complicated to realize because it is necessary to have an algorithm that makes possible to detect discriminant sequences with a numerical target.

Creation of interpretable recommender system based on explanations.

It may also be interesting to investigate an interpretable recommender system based on local explanations. The goal would be to make a recommendation model based on the knowledge acquired from black box models. The advantage of an explanation-based recommender system is that its recommendations are fully explainable by the rules used by the system. A first version of the model could be created from the rules found by **UPSD** and **SDSD** for only one model. The challenge relies on finding the best rules using two sets of rules. We can go even further, by creating a model that uses N sets of rules from several models.

Extend our methods to explain more type of recommender system.

For the moment, our approach is a model that is agnostic of the recommendation system used, as much as it is based on implicit feedback. However, if a model embeds a user vector to make a recommendation, our approach is currently not able to fully explain the recommendation. There is still a part of shadow on user embedding. A first improvement would be to explain how the model embeds the user. In addition, there are models that use content/context information in addition to user feedback. It would be interesting to be able to explain these models as well. This would lead to a fully model agnostic approach.

Chapter 7

Appendix

Table 7.1: Best hyperparameters for BPR, FMC, FPMC, PRME, TransRec and **REBUS** on each dataset.

Dataset	Models	minCount	L	λ_θ	bias_reg	α	γ
Epinions	BPR	∅	∅	0.1	0.1	∅	∅
Epinions	FPMC	∅	∅	0.1	∅	∅	∅
Epinions	FMC	∅	∅	0.1	∅	∅	∅
Epinions	PRME	∅	∅	0.01	∅	0.2	∅
Epinions	TransRec	∅	∅	0.001	0.001	0.1	∅
Epinions	REBUS	2	3	0.001	0.001	1	0.7
Foursq	BPR	∅	∅	0.01	0.001	∅	∅
Foursq	FPMC	∅	∅	0.001	∅	∅	∅
Foursq	FMC	∅	∅	0.01	∅	∅	∅
Foursq	PRME	∅	∅	0.001	∅	0.5	∅
Foursq	TransRec	∅	∅	0.001	1	0.001	∅
Foursq	REBUS	2	15	0.001	0.001	0.3	∅
Adressa	BPR	∅	∅	0.001	0	∅	∅
Adressa	FPMC	∅	∅	0	∅	∅	∅
Adressa	FMC	∅	∅	0	∅	∅	∅
Adressa	PRME	∅	∅	0	∅	0.2	∅
Adressa	TransRec	∅	∅	0	0	0.001	∅
Adressa	REBUS	2	2	0	0	0.3	0.2
Visiativ	BPR	∅	∅	0.01	0	∅	∅
Visiativ	FPMC	∅	∅	0.01	∅	∅	∅
Visiativ	FMC	∅	∅	0.01	∅	∅	∅
Visiativ	PRME	∅	∅	0.1	∅	0.2	∅
Visiativ	TransRec	∅	∅	0.01	0	0.1	∅
Visiativ	REBUS	2	10	0.01	0.01	1	0.5
Auto	BPR	∅	∅	0.1	0.1	∅	∅
Auto	FPMC	∅	∅	0.1	∅	∅	∅
Auto	FMC	∅	∅	0.1	∅	∅	∅
Auto	PRME	∅	∅	0.01	∅	0.5	∅
Auto	TransRec	∅	∅	0.01	0.001	0.01	∅
Auto	REBUS	2	8	0.001	0.001	1	0.7
Office	BPR	∅	∅	0.1	0.1	∅	∅
Office	FPMC	∅	∅	0.1	∅	∅	∅
Office	FMC	∅	∅	0.1	∅	∅	∅
Office	PRME	∅	∅	0.01	∅	0.5	∅
Office	TransRec	∅	∅	0.01	0.001	0.01	∅
Office	REBUS	2	8	0.001	0.001	1	0.7
Video	BPR	∅	∅	0.01	0.01	∅	∅
Video	FPMC	∅	∅	0.01	∅	∅	∅
Video	FMC	∅	∅	0.01	∅	∅	∅
Video	PRME	∅	∅	0.001	∅	0.5	∅
Video	TransRec	∅	∅	0.01	0.01	0.01	∅
Video	REBUS	2	10	0.001	0.001	0.3	0.3
ML-5	BPR	∅	∅	0.1	0.1	∅	∅
ML-5	FPMC	∅	∅	0.1	∅	∅	∅
ML-5	FMC	∅	∅	0.1	∅	∅	∅
ML-5	PRME	∅	∅	0.1	∅	0.2	∅
ML-5	TransRec	∅	∅	0.01	0	0.1	∅
ML-5	REBUS	2	3	0.01	0.01	0.3	0.5
ML-10	BPR	∅	∅	0.01	0.001	∅	∅
ML-10	FPMC	∅	∅	0.01	∅	∅	∅
ML-10	FMC	∅	∅	0.01	∅	∅	∅
ML-10	PRME	∅	∅	0.01	∅	0.2	∅
ML-10	TransRec	∅	∅	0	0.001	0.1	∅
ML-10	REBUS	2	8	0.001	0.001	1	0.7
ML-20	BPR	∅	∅	0.01	0.001	∅	∅
ML-20	FPMC	∅	∅	0.01	∅	∅	∅
ML-20	FMC	∅	∅	0.01	∅	∅	∅
ML-20	PRME	∅	∅	0.01	∅	0.5	∅
ML-20	TransRec	∅	∅	0.01	0.001	0.01	∅
ML-20	REBUS	2	8	0.001	0.001	0.7	0.5
ML-30	BPR	∅	∅	0.01	0.001	∅	∅
ML-30	FPMC	∅	∅	0.01	∅	∅	∅
ML-30	FMC	∅	∅	0.01	∅	∅	∅
ML-30	PRME	∅	∅	0.01	∅	0.5	∅
ML-30	TransRec	∅	∅	0.01	0.001	0.01	∅
ML-30	REBUS	2	10	0.001	0	0.6	0.4
ML-50	BPR	∅	∅	0.01	0.001	∅	∅
ML-50	FPMC	∅	∅	0.01	∅	∅	∅
ML-50	FMC	∅	∅	0.01	∅	∅	∅
ML-50	PRME	∅	∅	0.01	∅	0.5	∅
ML-50	TransRec	∅	∅	0.01	0	0.01	∅
ML-50	REBUS	2	10	0.001	0.001	0.6	0.3

Table 7.2: Best hyperparameters for SASRec on each dataset.

Dataset	Models	num_blocks	num_heads	l2_emb	maxlen	droprate
Epinions	SASRec	2	1	0.01	50	0.5
Foursq	SASRec	2	1	0.001	200	0.2
Adressa	SASRec	2	1	0	50	0.2
Visiativ	SASRec	2	1	0	200	0.5
Auto	SASRec	2	1	0.001	50	0.5
Office	SASRec	2	1	0.001	50	0.5
Video	SASRec	2	1	0.001	100	0.2
ML-5	SASRec	2	1	0.001	50	0.5
ML-10	SASRec	2	1	0	200	0.5
ML-20	SASRec	2	1	0.001	200	0.2
ML-30	SASRec	2	1	0	200	0.2
ML-50	SASRec	2	1	0	50	0.2

Table 7.3: Best hyperparameters for CASER on each dataset.

Dataset	Models	neg_samples	L	T	l2	nv	nh	droprate	ac_conv	ac_fc
Epinions	CASER	3	5	3	0	2	32	0.5	relu	tanh
Foursq	CASER	3	5	3	0.001	16	64	0.5	sigm	iden
Adressa	CASER	3	5	3	0	4	16	0.3	relu	relu
Visiativ	CASER	3	5	2	0.001	2	4	0.5	iden	relu
Auto	CASER	3	5	3	0.001	16	16	0.5	relu	relu
Office	CASER	3	5	1	0.001	16	4	0.5	tanh	relu
Video	CASER	3	5	3	0	16	16	0.5	relu	relu
ML-10	CASER	3	5	2	0.001	1	4	0.5	relu	relu
ML-20	CASER	3	9	2	0.001	8	8	0.5	iden	iden
ML-30	CASER	3	5	1	0.001	4	32	0.5	relu	iden
ML-5	CASER	3	5	3	0.001	8	8	0.5	tanh	relu
ML-50	CASER	3	5	1	0	8	4	0.5	relu	iden

Table 7.4: Best hyperparameters for S-KNN on each dataset.

Dataset	Models	k	sample_size	sampling	similarity
Epinions	sknn	200	500	recent	binary
Foursq	sknn	500	500	recent	jaccard
Adressa	sknn	500	1000	random	cosine
Visiativ	sknn	500	500	recent	jaccard
Auto	sknn	500	500	recent	binary
Office	sknn	500	500	recent	cosine
Video	sknn	500	500	recent	cosine
ML-5	sknn	500	500	recent	jaccard
ML-10	sknn	500	2000	recent	jaccard
ML-20	sknn	500	2000	recent	jaccard
ML-30	sknn	500	2000	recent	jaccard
ML-50	sknn	500	2000	recent	jaccard

Table 7.5: HIT_5, HIT_10, NDCG_5 and NDCG_10 for the different models on Epinions, Foursquare, Adressa, Visiativ, Amazon-Automotive, Amazon-Office-Product and Amazon-Video-Games datasets. The last row, called *Improv. vs Best*, shows the improvement in percentage of our method compared to the other best model. The last column is the average performance on these 7 datasets, the average of all datasets is included in Table 7.6.

	Metric	**Epinions**	**Foursq**	**Adressa**	**Visiativ**	**Auto**	**Office**	**Games**	**Avg**
POP	HIT5	0.77%	17.59%	7.00%	11.68%	0.96%	0.02%	1.20%	5.60%
	HIT10	1.37%	29.33%	11.80%	18.35%	1.59%	0.05%	1.90%	9.20%
	NDGC5	0.40%	10.94%	3.82%	7.42%	0.54%	0.02%	0.76%	3.41%
	NDGC10	0.59%	14.72%	5.41%	9.56%	0.74%	0.02%	0.99%	4.57%
FMC	HIT5	0.81%	25.15%	28.69%	22.72%	0.96%	0.46%	3.72%	11.79%
	HIT10	1.02%	35.84%	42.82%	31.97%	1.46%	0.76%	5.98%	17.12%
	NDGC5	0.55%	18.48%	19.39%	16.87%	0.62%	0.28%	2.34%	8.36%
	NDGC10	0.61%	21.93%	23.95%	19.86%	0.78%	0.38%	3.06%	10.08%
BPR	HIT5	1.12%	19.56%	8.02%	15.27%	0.86%	0.52%	2.24%	6.80%
	HIT10	1.53%	30.12%	14.26%	24.52%	1.57%	0.74%	3.96%	10.96%
	NDGC5	0.74%	12.93%	4.53%	9.76%	0.55%	0.36%	1.37%	4.32%
	NDGC10	0.87%	16.33%	6.55%	12.73%	0.78%	0.43%	1.92%	5.66%
FPMC	HIT5	0.67%	31.56%	30.36%	22.72%	0.70%	0.41%	4.05%	12.92%
	HIT10	0.84%	41.36%	44.62%	31.25%	1.16%	0.71%	6.92%	18.12%
	NDGC5	0.46%	24.03%	20.47%	16.22%	0.45%	0.25%	2.49%	9.19%
	NDGC10	0.51%	27.18%	25.06%	18.97%	0.60%	0.35%	3.41%	10.87%
PRME	HIT5	0.72%	29.32%	30.10%	25.52%	0.66%	0.69%	4.16%	13.03%
	HIT10	1.00%	38.69%	43.65%	32.69%	1.16%	1.18%	6.83%	17.88%
	NDGC5	0.55%	23.57%	20.80%	19.56%	0.41%	0.40%	2.63%	9.70%
	NDGC10	0.63%	26.58%	25.16%	21.84%	0.57%	0.56%	3.49%	11.26%
TransRec	HIT5	0.91%	35.06%	31.79%	27.46%	1.41%	1.00%	3.27%	14.41%
	HIT10	1.26%	44.62%	45.77%	36.63%	2.27%	1.83%	5.64%	19.72%
	NDGC5	0.65%	26.62%	22.61%	20.11%	0.88%	0.60%	2.08%	10.51%
	NDGC10	0.76%	29.70%	27.11%	23.07%	1.15%	0.86%	2.84%	12.21%
S-KNN	HIT5	1.26%	47.16%	12.46%	21.58%	2.56%	2.97%	5.37%	13.34%
	HIT10	1.65%	54.09%	18.31%	31.18%	3.53%	3.90%	8.35%	17.29%
	NDGC5	0.99%	41.28%	8.78%	15.42%	1.79%	2.12%	3.50%	10.55%
	NDGC10	1.12%	43.53%	10.67%	18.51%	2.10%	2.42%	4.46%	11.83%
CASER	HIT5	0.81%	16.56%	25.17%	20.86%	0.58%	0.36%	1.79%	9.45%
	HIT10	1.37%	25.86%	38.94%	30.90%	1.24%	0.54%	3.19%	14.58%
	NDGC5	0.52%	10.39%	16.56%	14.29%	0.38%	0.18%	1.14%	6.21%
	NDGC10	0.70%	13.39%	20.99%	17.50%	0.60%	0.24%	1.59%	7.86%
SASRec	HIT5	0.98%	32.64%	28.65%	25.52%	0.78%	0.98%	3.05%	13.23%
	HIT10	1.56%	41.56%	43.98%	35.63%	1.24%	1.67%	5.02%	18.66%
	NDGC5	0.60%	24.53%	18.58%	17.82%	0.49%	0.60%	1.93%	9.22%
	NDGC10	0.78%	27.41%	23.50%	21.08%	0.63%	0.82%	2.56%	10.97%
REBUS	HIT5	1.14%	47.10%	32.18%	25.45%	1.48%	1.21%	2.90%	15.92%
	HIT10	1.74%	53.14%	46.30%	35.34%	2.47%	2.14%	4.82%	20.85%
	NDGC5	0.71%	41.72%	22.45%	16.64%	0.93%	0.74%	1.82%	12.14%
	NDGC10	0.91%	43.67%	27.00%	19.81%	1.25%	1.04%	2.44%	13.73%
Improv. vs Best	HIT5	-9.52%	-0.13%	1.23%	-7.32%	-42.19%	-59.26%	-46.00%	10.48%
	HIT10	5.45%	-1.76%	1.16%	-3.52%	-30.03%	-45.13%	-42.28%	5.73%
	NDGC5	-28.28%	1.07%	-0.71%	-17.26%	-48.04%	-65.09%	-48.00%	15.07%
	NDGC10	-18.75%	0.32%	-0.41%	-14.13%	-40.48%	-57.02%	-45.29%	12.45%

Table 7.6: HIT_5, HIT_10, NDCG_5 and NDCG_10 for the different models on ML-5, ML-10, ML-20, ML-30 and ML-50 datasets. The last row, called *Improv. vs Best*, shows the improvement in percentage of our method compared to the other best model. The last column is the average performance on the 12 datasets, including datasets of Table 7.5.

	Metric	**ML-5**	**ML-10**	**ML-20**	**ML-30**	**ML-50**	**Avg(ML)**	**Avg(All)**
POP	HIT5	1.92%	2.25%	2.07%	1.94%	1.94%	2.02%	4.11%
	HIT10	3.76%	3.81%	4.02%	3.99%	4.21%	3.96%	7.01%
	NDGC5	1.13%	1.33%	1.25%	1.12%	1.13%	1.20%	2.49%
	NDGC10	1.74%	1.83%	1.88%	1.79%	1.86%	1.82%	3.43%
FMC	HIT5	2.90%	5.10%	6.08%	5.80%	6.31%	5.24%	9.06%
	HIT10	5.46%	8.21%	9.37%	9.46%	10.12%	8.52%	13.54%
	NDGC5	1.70%	3.35%	3.84%	3.76%	4.10%	3.35%	6.27%
	NDGC10	2.52%	4.35%	4.91%	4.94%	5.32%	4.41%	7.72%
BPR	HIT5	2.82%	2.78%	2.96%	2.72%	2.75%	2.81%	5.13%
	HIT10	5.13%	5.53%	5.25%	4.80%	5.27%	5.20%	8.56%
	NDGC5	1.69%	1.74%	1.84%	1.59%	1.63%	1.70%	3.23%
	NDGC10	2.43%	2.62%	2.58%	2.26%	2.43%	2.46%	4.33%
FPMC	HIT5	2.25%	4.54%	5.78%	6.16%	6.24%	4.99%	9.62%
	HIT10	4.11%	7.52%	9.47%	10.23%	10.35%	8.34%	14.04%
	NDGC5	1.38%	2.90%	3.53%	3.93%	4.02%	3.15%	6.68%
	NDGC10	1.97%	3.85%	4.71%	5.24%	5.34%	4.22%	8.10%
PRME	HIT5	2.90%	4.49%	3.86%	4.97%	5.38%	4.32%	9.40%
	HIT10	5.30%	8.30%	7.29%	8.49%	9.44%	7.76%	13.67%
	NDGC5	1.75%	2.80%	2.30%	2.95%	3.27%	2.61%	6.75%
	NDGC10	2.52%	4.03%	3.39%	4.07%	4.57%	3.72%	8.12%
TransRec	HIT5	4.22%	5.33%	4.24%	4.67%	4.87%	4.67%	10.35%
	HIT10	7.58%	8.76%	8.30%	7.85%	8.41%	8.18%	14.91%
	NDGC5	2.62%	3.22%	2.69%	2.96%	3.12%	2.92%	7.35%
	NDGC10	3.68%	4.32%	3.99%	3.98%	4.25%	4.04%	8.81%
S-KNN	HIT5	3.28%	4.29%	3.49%	3.15%	3.00%	3.44%	9.21%
	HIT10	5.46%	7.40%	6.06%	5.98%	5.70%	6.12%	12.63%
	NDGC5	2.19%	2.73%	2.07%	1.97%	1.89%	2.17%	7.06%
	NDGC10	2.89%	3.73%	2.89%	2.87%	2.76%	3.03%	8.16%
CASER	HIT5	1.92%	2.83%	4.21%	4.42%	5.91%	3.86%	7.12%
	HIT10	3.76%	5.27%	7.48%	8.10%	10.22%	6.96%	11.41%
	NDGC5	1.15%	1.81%	2.64%	2.81%	3.67%	2.42%	4.63%
	NDGC10	1.76%	2.59%	3.70%	3.99%	5.04%	3.42%	6.01%
SASRec	HIT5	3.43%	5.08%	4.29%	5.66%	5.73%	4.84%	9.73%
	HIT10	5.93%	8.16%	7.92%	9.84%	9.65%	8.30%	14.35%
	NDGC5	2.07%	3.23%	2.68%	3.60%	3.68%	3.05%	6.65%
	NDGC10	2.87%	4.21%	3.84%	4.94%	4.93%	4.16%	8.13%
REBUS	HIT5	4.19%	4.31%	3.92%	4.50%	4.65%	4.32%	11.09%
	HIT10	7.34%	7.53%	7.67%	7.90%	8.25%	7.74%	15.39%
	NDGC5	2.59%	2.68%	2.49%	2.80%	2.84%	2.68%	8.20%
	NDGC10	3.59%	3.73%	3.69%	3.89%	3.98%	3.77%	9.58%
Improv. vs Best	HIT5	-0.71%	-19.14%	-35.53%	-26.95%	-26.31%	-17.56%	7.15%
	HIT10	-3.17%	-14.04%	-19.01%	-22.78%	-20.29%	-9.15%	3.22%
	NDGC5	-1.15%	-20.00%	-35.16%	-28.75%	-30.73%	-20.00%	11.56%
	NDGC10	-2.45%	-14.25%	-24.85%	-25.76%	-25.47%	-14.51%	8.74%

Table 7.7: HIT_5, HIT_10, NDCG_5 and NDCG_10 for the different models that do not suffer of the problem of cold-start users. The 3 last rows, called *Improv. vs Best*, *Improv. vs SASRec* and *Improv. vs S-KNN*, shows the improvement in percentage of our method compared to the best model, SASRec and S-KNN.

	Metric	Epinions	Foursq	Adressa	Visiativ	Auto	Office	Games	Avg
POP	HIT5	1.22%	14.06%	14.83%	15.93%	1.08%	0.04%	1.46%	6.94%
	HIT10	2.10%	24.42%	29.40%	20.69%	1.70%	0.06%	2.12%	11.50%
	NDGC5	0.76%	8.39%	10.30%	8.97%	0.64%	0.03%	0.89%	4.28%
	NDGC10	1.04%	11.72%	15.19%	10.50%	0.83%	0.03%	1.10%	5.78%
FMC	HIT5	0.79%	21.72%	26.51%	23.78%	1.45%	0.48%	3.70%	11.20%
	HIT10	1.17%	31.82%	39.01%	33.06%	2.08%	0.83%	6.20%	16.31%
	NDGC5	0.52%	15.58%	18.34%	17.80%	1.00%	0.31%	2.34%	7.98%
	NDGC10	0.64%	18.82%	22.42%	20.81%	1.20%	0.43%	3.14%	9.64%
S-KNN	HIT5	1.64%	34.85%	36.37%	28.24%	3.91%	4.43%	7.03%	16.64%
	HIT10	2.17%	42.53%	45.38%	38.53%	5.05%	5.45%	9.94%	21.29%
	NDGC5	1.35%	29.27%	28.41%	21.55%	2.85%	3.40%	4.79%	13.09%
	NDGC10	1.52%	31.75%	31.32%	24.85%	3.22%	3.72%	5.72%	14.59%
SASRec	HIT5	1.11%	23.18%	31.45%	25.74%	0.97%	0.80%	3.20%	12.35%
	HIT10	1.93%	32.17%	42.46%	35.73%	1.66%	1.43%	5.14%	17.22%
	NDGC5	0.68%	16.46%	20.95%	18.14%	0.61%	0.48%	2.05%	8.48%
	NDGC10	0.94%	19.36%	24.48%	21.33%	0.83%	0.68%	2.68%	10.04%
REBUS	HIT5	1.37%	30.19%	34.83%	27.35%	1.63%	1.76%	3.40%	14.36%
	HIT10	2.18%	39.42%	45.55%	37.81%	2.60%	2.80%	5.62%	19.43%
	NDGC5	0.89%	23.76%	25.42%	19.50%	1.00%	1.18%	2.16%	10.56%
	NDGC10	1.16%	26.73%	28.89%	22.86%	1.31%	1.51%	2.87%	12.19%
Improv.	HIT5	-16.46%	-13.37%	-4.23%	-3.15%	-58.31%	-60.27%	-51.64%	-13.70%
VS	HIT10	0.46%	-7.31%	0.37%	-1.87%	-48.51%	-48.62%	-43.46%	-8.74%
Best	NDGC5	-34.07%	-18.82%	-10.52%	-9.51%	-64.91%	-65.29%	-54.91%	-19.33%
	NDGC10	-23.68%	-15.81%	-7.76%	-8.01%	-59.32%	-59.41%	-49.83%	-16.45%
Improv.	HIT5	23.42%	30.24%	10.75%	6.25%	68.04%	120.00%	6.25%	16.28%
VS	HIT10	12.95%	22.54%	7.28%	5.82%	56.63%	95.80%	9.34%	12.83%
SASRec	NDGC5	30.88%	44.35%	21.34%	7.50%	63.93%	145.83%	5.37%	24.53%
	NDGC10	23.40%	38.07%	18.01%	7.17%	57.83%	122.06%	7.09%	21.41%
Improv.	HIT5	-16.46%	-13.37%	-4.23%	-3.15%	-58.31%	-60.27%	-51.64%	-13.70%
VS	HIT10	0.46%	-7.31%	0.37%	-1.87%	-48.51%	-48.62%	-43.46%	-8.74%
S-KNN	NDGC5	-34.07%	-18.82%	-10.52%	-9.51%	-64.91%	-65.29%	-54.91%	-19.33%
	NDGC10	-23.68%	-15.81%	-7.76%	-8.01%	-59.32%	-59.41%	-49.83%	-16.45%

Table 7.8: HIT_5, HIT_10, NDCG_5 and NDCG_10 for **REBUS** and his variants on Epinions, Foursquare, Adressa, Visiativ, Amazon-Automotive, Amazon-Office-Product and Amazon-Video-Games datasets. The last column is the average performance on these 7 datasets, the average of all datasets is included in Table 7.9.

	Metric	Epinions	Foursq	Adressa	Visiativ	Auto	Office	Games	Avg
REBUS_UP	HIT5	1.05%	43.73%	31.86%	18.06%	1.43%	1.32%	2.61%	14.30%
	HIT10	1.60%	50.30%	44.45%	27.10%	2.34%	2.11%	4.45%	18.91%
	NDGC5	0.70%	37.30%	23.09%	11.37%	0.90%	0.85%	1.64%	10.84%
	NDGC10	0.88%	39.42%	27.15%	14.26%	1.20%	1.10%	2.23%	12.32%
REBUS_SD$_{1MC}$	HIT5	1.26%	29.45%	29.99%	27.74%	1.25%	1.14%	3.21%	13.44%
	HIT10	1.74%	39.60%	43.20%	37.71%	2.06%	1.99%	5.08%	18.77%
	NDGC5	0.82%	21.89%	20.84%	20.68%	0.79%	0.75%	2.01%	9.68%
	NDGC10	0.97%	25.15%	25.08%	23.89%	1.05%	1.02%	2.61%	11.40%
REBUS_SD$_{2MC}$	HIT5	1.02%	31.34%	30.40%	27.38%	1.48%	1.31%	2.94%	13.70%
	HIT10	1.60%	40.95%	43.83%	37.20%	2.23%	2.21%	4.90%	18.99%
	NDGC5	0.77%	23.52%	21.22%	18.63%	0.91%	0.78%	1.86%	9.67%
	NDGC10	0.96%	26.61%	25.54%	21.83%	1.15%	1.06%	2.49%	11.38%
REBUS_SD$_{3MC}$	HIT5	1.14%	32.12%	30.25%	25.02%	1.44%	1.29%	2.82%	13.44%
	HIT10	2.00%	41.70%	43.64%	35.13%	2.23%	2.31%	4.70%	18.82%
	NDGC5	0.73%	24.53%	20.93%	16.64%	0.88%	0.77%	1.80%	9.47%
	NDGC10	1.01%	27.61%	25.24%	19.89%	1.14%	1.11%	2.40%	11.20%
REBUS_SD	HIT5	0.91%	31.50%	30.22%	27.81%	1.30%	1.21%	3.11%	13.72%
	HIT10	1.53%	41.04%	44.00%	36.99%	2.14%	1.99%	4.87%	18.94%
	NDGC5	0.63%	24.14%	20.95%	19.90%	0.82%	0.72%	1.95%	9.87%
	NDGC10	0.83%	27.21%	25.38%	22.88%	1.09%	0.98%	2.52%	11.56%
REBUS$_{1MC}$	HIT5	1.28%	47.24%	31.81%	25.52%	1.46%	1.24%	2.93%	15.92%
	HIT10	1.88%	53.24%	46.19%	36.70%	2.35%	2.14%	4.88%	21.06%
	NDGC5	0.83%	41.80%	22.17%	16.93%	0.91%	0.75%	1.84%	12.18%
	NDGC10	1.03%	43.74%	26.81%	20.53%	1.20%	1.04%	2.47%	13.83%
REBUS$_{2MC}$	HIT5	1.21%	47.15%	32.02%	23.58%	1.52%	1.32%	2.85%	15.66%
	HIT10	1.74%	53.00%	46.21%	34.77%	2.30%	2.13%	4.83%	20.71%
	NDGC5	0.78%	41.66%	22.35%	15.69%	0.93%	0.85%	1.79%	12.01%
	NDGC10	0.94%	43.55%	26.93%	19.28%	1.18%	1.11%	2.42%	13.63%
REBUS$_{3MC}$	HIT5	1.16%	47.15%	30.39%	23.23%	1.42%	1.27%	2.72%	15.33%
	HIT10	1.86%	53.15%	44.07%	33.33%	2.29%	2.19%	4.55%	20.21%
	NDGC5	0.75%	41.82%	20.93%	15.17%	0.90%	0.83%	1.70%	11.73%
	NDGC10	0.97%	43.76%	25.34%	18.41%	1.18%	1.13%	2.29%	13.30%
REBUS	HIT5	1.14%	47.10%	32.18%	25.45%	1.48%	1.21%	2.90%	15.92%
	HIT10	1.74%	53.14%	46.30%	35.34%	2.47%	2.14%	4.82%	20.85%
	NDGC5	0.71%	41.75%	22.45%	16.64%	0.93%	0.74%	1.82%	12.15%
	NDGC10	0.91%	43.70%	27.00%	19.81%	1.25%	1.04%	2.44%	13.73%

Table 7.9: HIT_5, HIT_10, NDCG_5 and NDCG_10 for **REBUS** and his variants on ML-5, ML-10, ML-20, ML-30 and ML-50 datasets. The last column is the average performance on the 12 datasets, including datasets of Table 7.8.

	Metric	ML-5	ML-10	ML-20	ML-30	ML-50	Avg(ML)	Avg(All)
REBUS_UP	HIT5	3.58%	3.30%	2.85%	2.78%	2.47%	2.99%	9.59%
	HIT10	6.54%	5.78%	5.30%	5.63%	5.02%	5.65%	13.39%
	NDGC5	2.29%	1.99%	1.72%	1.71%	1.53%	1.85%	7.09%
	NDGC10	3.25%	2.79%	2.50%	2.62%	2.34%	2.70%	8.31%
REBUS_SD$_{1MC}$	HIT5	4.19%	5.61%	5.70%	6.08%	6.13%	5.54%	10.15%
	HIT10	6.94%	8.96%	9.60%	9.84%	9.85%	9.04%	14.71%
	NDGC5	2.65%	3.50%	3.52%	3.79%	3.84%	3.46%	7.09%
	NDGC10	3.53%	4.56%	4.77%	4.99%	5.03%	4.58%	8.55%
REBUS_SD$_{2MC}$	HIT5	4.12%	5.51%	6.11%	6.54%	6.44%	5.75%	10.38%
	HIT10	7.07%	9.55%	10.20%	10.76%	10.37%	9.59%	15.07%
	NDGC5	2.55%	3.38%	3.76%	4.22%	3.95%	3.57%	7.13%
	NDGC10	3.50%	4.68%	5.07%	5.58%	5.21%	4.81%	8.64%
REBUS_SD$_{3MC}$	HIT5	3.94%	5.18%	5.88%	6.16%	5.73%	5.38%	10.08%
	HIT10	7.00%	9.27%	10.08%	10.53%	9.84%	9.35%	14.87%
	NDGC5	2.42%	3.23%	3.72%	3.96%	3.62%	3.39%	6.94%
	NDGC10	3.39%	4.54%	5.07%	5.36%	4.95%	4.66%	8.48%
REBUS_SD	HIT5	4.07%	5.02%	5.33%	5.90%	5.68%	5.20%	10.17%
	HIT10	6.97%	8.58%	9.02%	9.99%	9.90%	8.89%	14.75%
	NDGC5	2.56%	3.20%	3.25%	3.73%	3.43%	3.23%	7.11%
	NDGC10	3.48%	4.34%	4.43%	5.04%	4.78%	4.41%	8.58%
REBUS$_{1MC}$	HIT5	4.11%	4.44%	4.09%	4.62%	4.65%	4.38%	11.11%
	HIT10	7.37%	7.75%	7.82%	8.28%	8.25%	7.89%	15.57%
	NDGC5	2.59%	2.64%	2.58%	2.91%	2.96%	2.74%	8.24%
	NDGC10	3.64%	3.71%	3.77%	4.08%	4.12%	3.86%	9.68%
REBUS$_{2MC}$	HIT5	4.12%	4.32%	4.16%	4.60%	4.42%	4.33%	10.94%
	HIT10	7.29%	7.58%	7.22%	8.10%	8.26%	7.69%	15.29%
	NDGC5	2.50%	2.60%	2.52%	2.91%	2.76%	2.66%	8.11%
	NDGC10	3.51%	3.64%	3.50%	4.02%	3.99%	3.73%	9.51%
REBUS$_{3MC}$	HIT5	3.96%	4.12%	4.11%	4.17%	4.02%	4.08%	10.64%
	HIT10	7.00%	7.20%	7.17%	8.00%	7.50%	7.38%	14.86%
	NDGC5	2.41%	2.53%	2.47%	2.58%	2.44%	2.49%	7.88%
	NDGC10	3.38%	3.52%	3.45%	3.80%	3.56%	3.54%	9.23%
REBUS	HIT5	4.19%	4.31%	3.92%	4.50%	4.65%	4.32%	11.09%
	HIT10	7.34%	7.53%	7.67%	7.90%	8.25%	7.74%	15.39%
	NDGC5	2.59%	2.68%	2.49%	2.80%	2.84%	2.68%	8.20%
	NDGC10	3.59%	3.73%	3.69%	3.89%	3.98%	3.77%	9.58%

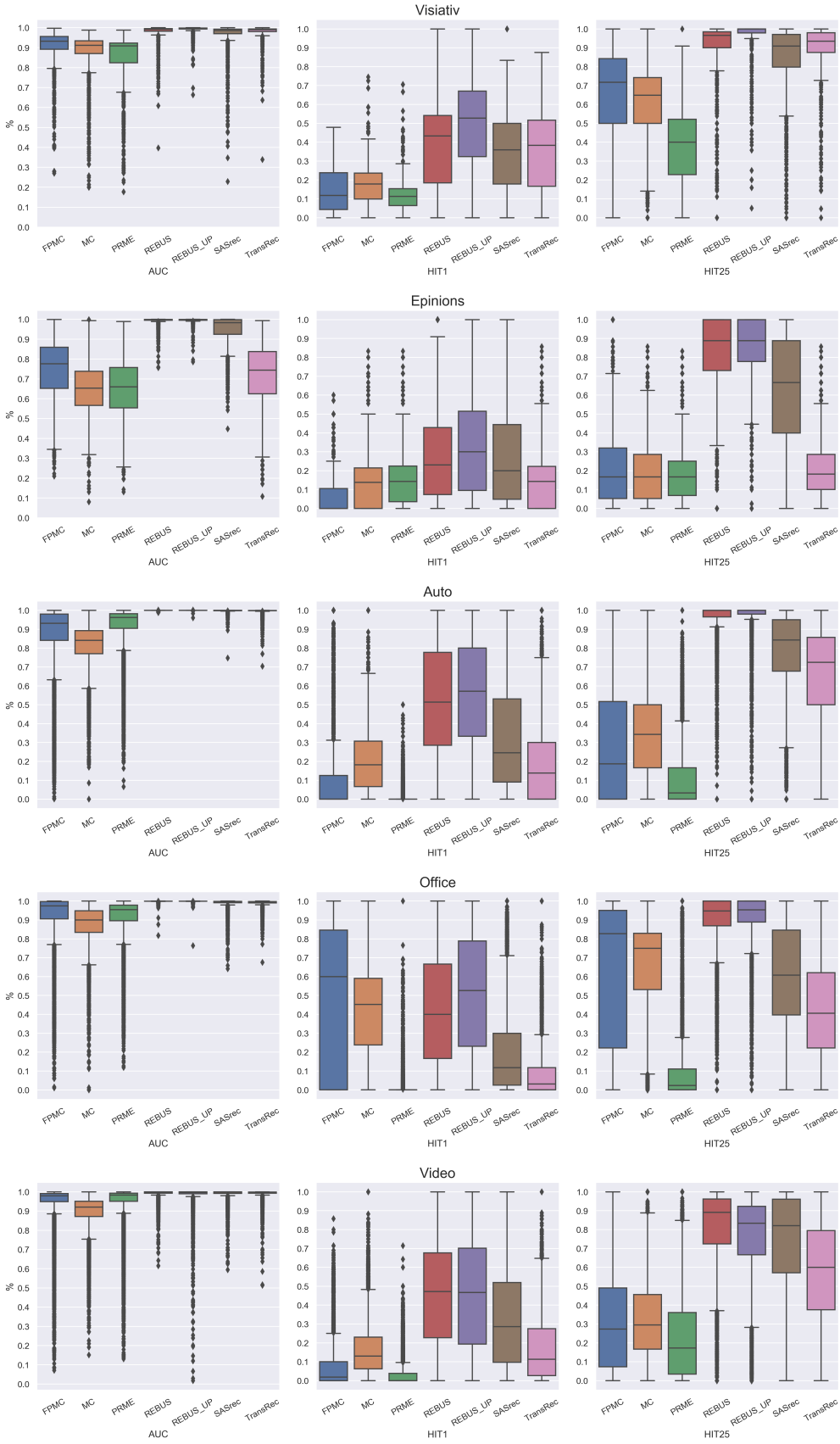

Figure 7.1: Performance metrics for Visiativ, Epinions and Amazon dataset (AUC, HIT1 and HIT25) evaluated on **UPSD** subgroups descriptions found in at least 6 users for each model.

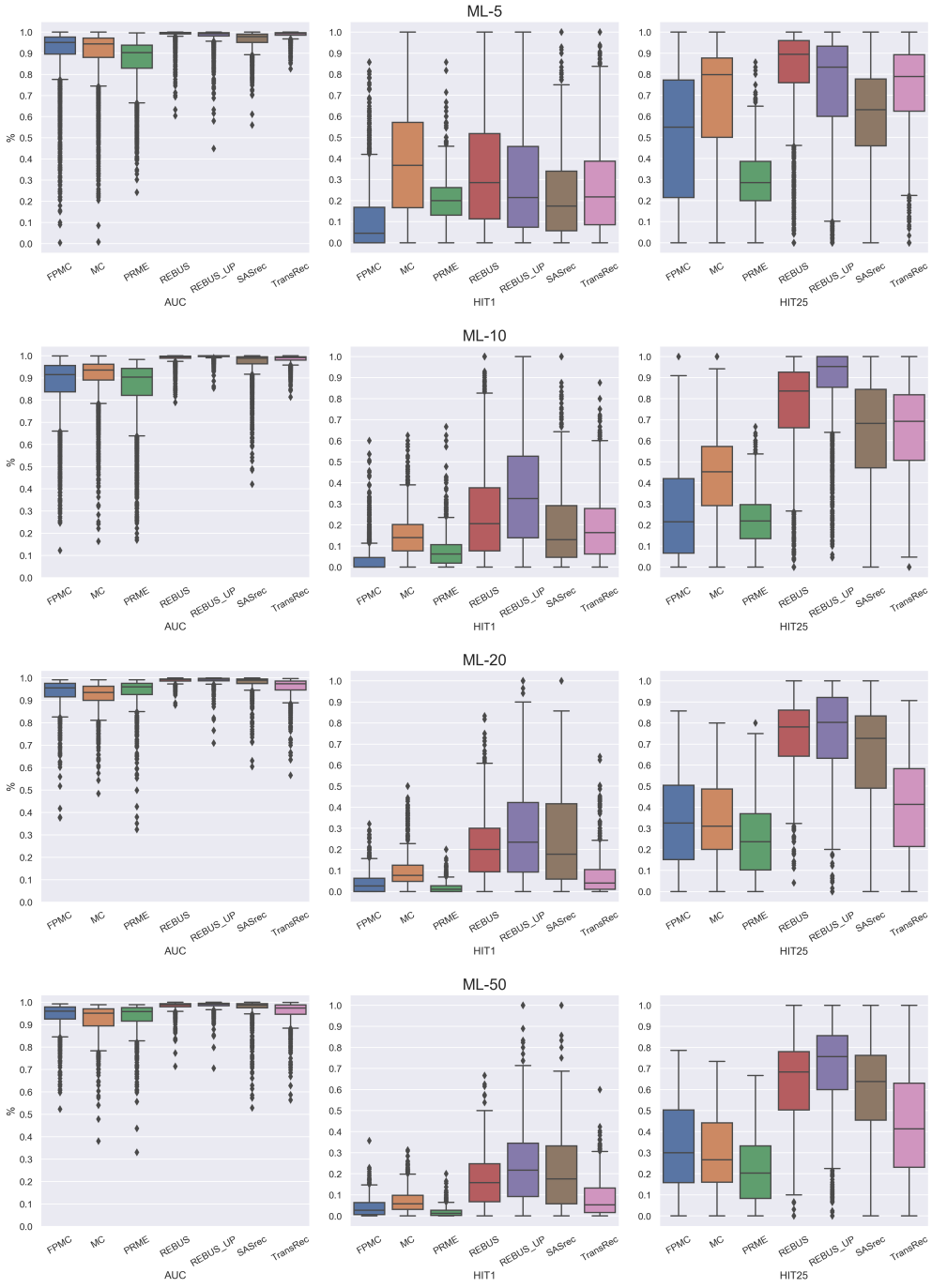

Figure 7.2: Box plots of the performance metrics for ML datasets (AUC, HIT1 and HIT25) evaluated on **UPSD** subgroups descriptions found in at least 6 users for each model.

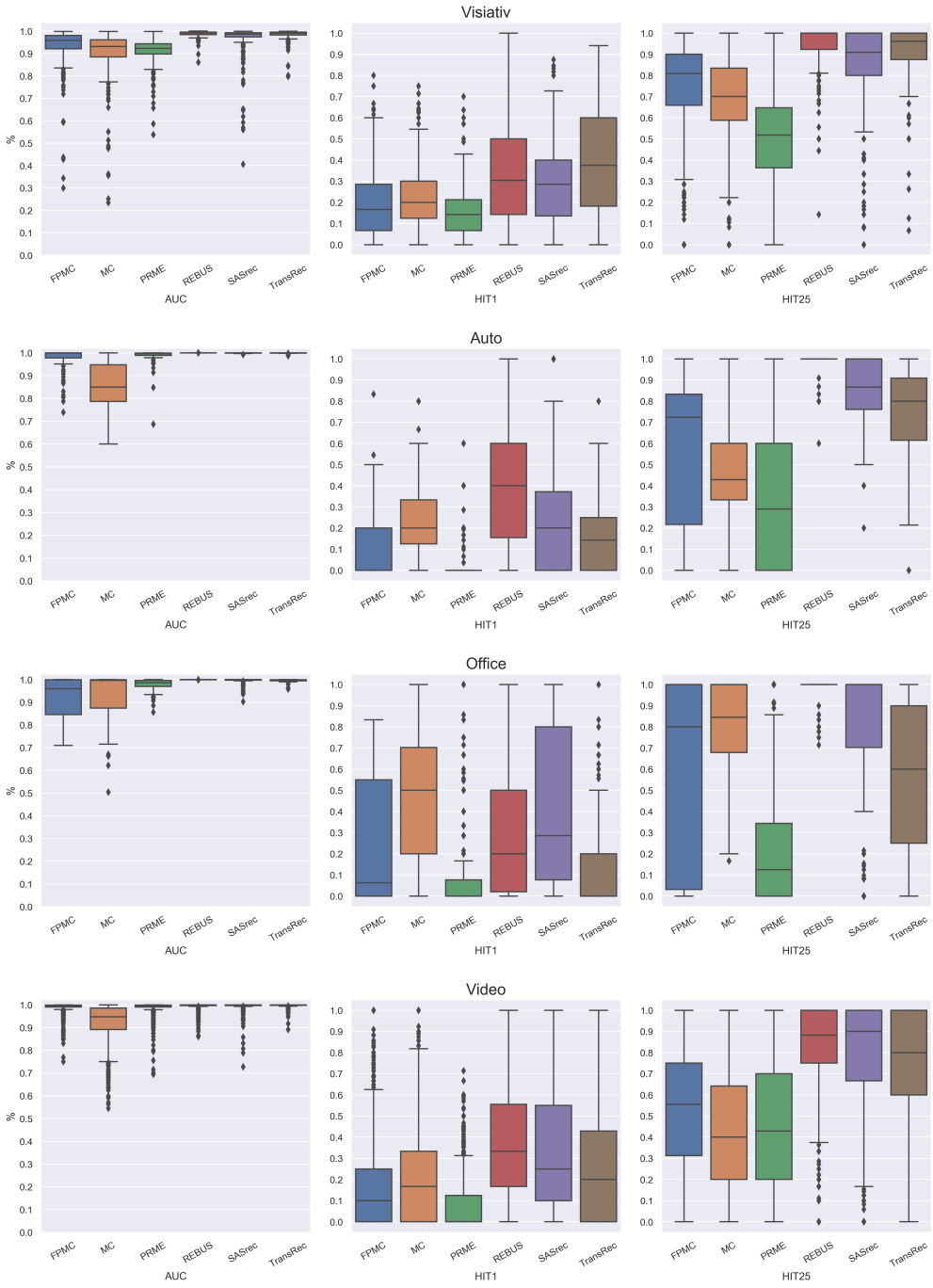

Figure 7.3: Performance metrics for Visiativ and Amazon dataset (AUC, HIT1 and HIT25) evaluated on **SDSD** subgroups descriptions found in at least 6 users for each model.

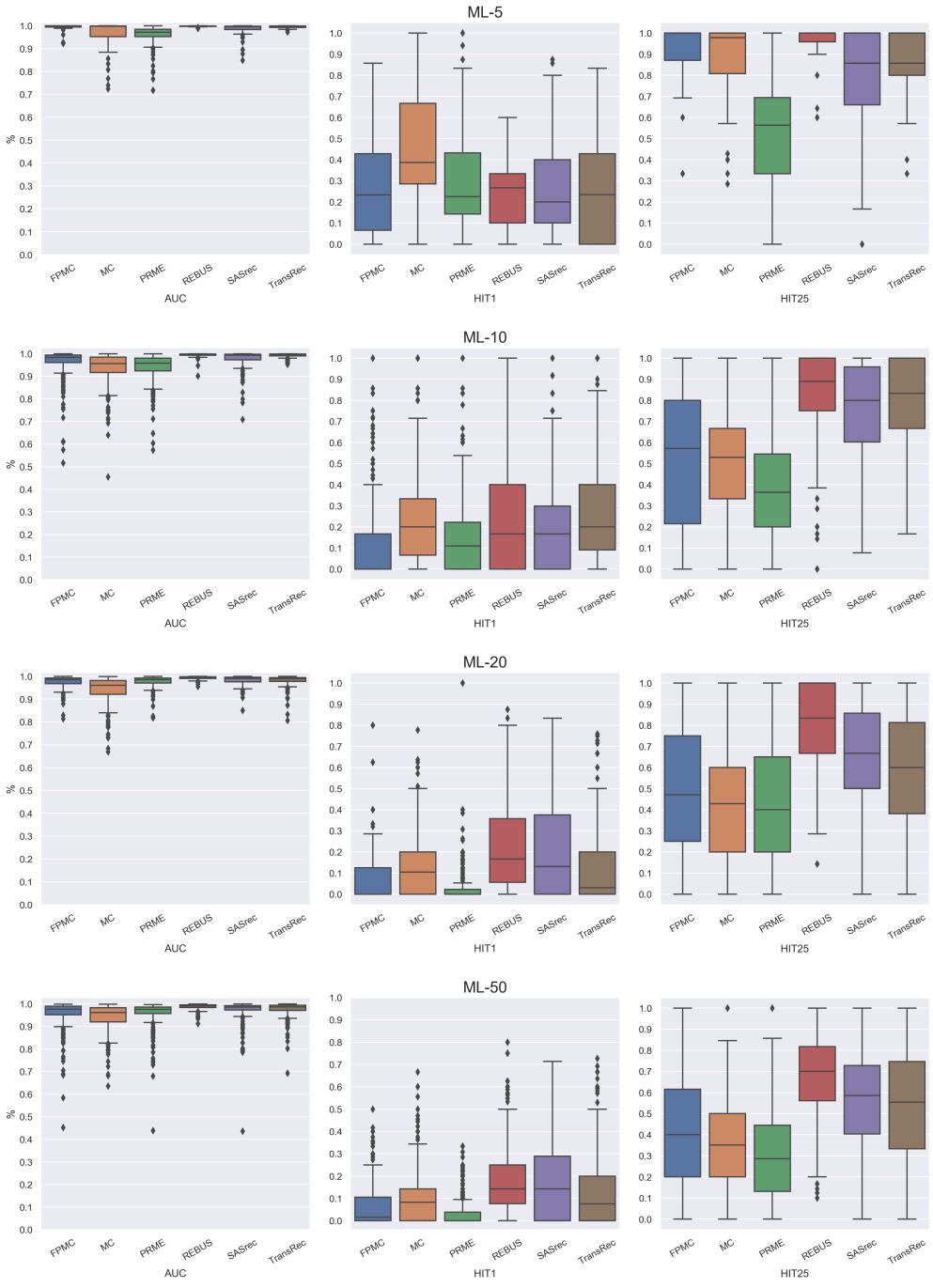

Figure 7.4: Box plots of the performance metrics for ML datasets (AUC, HIT1 and HIT25) evaluated on **SDSD** subgroups descriptions found in at least 6 users for each model.

Table 7.10: HIT_10, HIT_50, NDCG_10 and NDCG_50 for *myCADservices* dataset with different preprocessing. The two numbers after *myCAD* are respectively the value of USERMIN and ITEMMIN in the Table 5.1. The last column is the average performance on these 3 different preprocessing.

	Metric	**myCAD_3_3**	**myCAD_4_4**	**myCAD_5_5**	**Avg(All)**
POP	HIT10	20.98%	20.86%	20.82%	21.04%
	HIT50	46.70%	46.94%	48.32%	47.42%
	NDGC10	11.04%	11.01%	11.31%	11.17%
	NDGC50	16.65%	16.70%	17.33%	16.94%
TransRec	HIT10	41.38%	38.88%	43.13%	39.75%
	HIT50	67.71%	67.52%	70.18%	67.44%
	NDGC10	28.12%	24.06%	29.05%	25.55%
	NDGC50	33.89%	30.43%	35.06%	31.68%
SASRec	HIT10	37.78%	40.12%	41.52%	39.55%
	HIT50	67.25%	68.62%	69.96%	68.22%
	NDGC10	23.72%	25.58%	27.42%	25.48%
	NDGC50	30.20%	31.87%	33.68%	31.79%
REBUS	HIT10	38.85%	39.71%	41.19%	39.68%
	HIT50	68.28%	68.98%	70.18%	68.44%
	NDGC10	23.31%	23.73%	24.74%	24.10%
	NDGC50	29.90%	30.23%	31.17%	30.48%
REBUS$_{1MC}$	HIT10	38.74%	40.71%	42.02%	40.55%
	HIT50	68.43%	68.80%	70.29%	68.69%
	NDGC10	23.41%	24.47%	25.58%	24.83%
	NDGC50	30.01%	30.74%	31.85%	31.06%

Bibliography

Gediminas Adomavicius and YoungOk Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Transactions on Knowledge & Data Engineering*, 24(05):896–911, may 2012. 45

Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer Publishing Company, Incorporated, 1st edition, 2016. 2, 8, 13, 42

Martin Atzmueller. Subgroup discovery. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 5(1):35–49, 2015. 65

Martin Atzmueller and Florian Lemmerich. Fast Subgroup Discovery for Continuous Target Concepts. In *Proc. International Symposium on Methodologies for Intelligent Systems*, volume 5722 of *LNCS*, pages 1–15, Berlin/Heidelberg, Germany, 2009. Springer. 72

Martin Atzmueller and Thomas Roth-Berghofer. The Mining and Analysis Continuum of Explaining Uncovered. In *Proc. Research and Development in Intelligent Systems XXVII. SGAI 2010*, pages 273–278. Springer, 2010. 67

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 26

Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 1341–1350, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. 10, 43

Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54, 2018. 26

Or Biran and Courtenay Cotton. Explanation and Justification in Machine Learning: A Survey. In *IJCAI-17 Workshop on Explainable AI*, 2017. 67

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. 36

Stefan Bloemheuvel, Benjamin Kloepper, Jurgen van den Hoogen, and Martin Atzmueller. Enhancing Sequential Pattern Mining Explainability with Markov Chain Probabilities. In *Proc. Dutch-Belgian Database Day*. Jheronimus Academy of Data Science, Den Bosch, Netherlands, 2019. 67

Geoffray Bonnin and Dietmar Jannach. Automated generation of music playlists: Survey and experiments. *ACM Comput. Surv.*, 47(2):26:1–26:35, 2014. 19

Robin D. Burke. Hybrid recommender systems: Survey and experiments. *User Model. User Adapt. Interact.*, 12(4):331–370, 2002. 12

Robin D. Burke. Hybrid web recommender systems. In *The Adaptive Web, Methods and Strategies of Web Personalization*, pages 377–408, 2007. 12

Robin Burke D. Knowledge-based recommender systems. *Encyclopedia of library and information systems*, 69(Supplement 32):175–186, 2000. 11

Pedro G Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1-2):67–119, 2014. 12

Pablo Castells, Saúl Vargas, and Jun Wang. Novelty and diversity metrics for recommender systems: Choice, discovery and relevance. *Proceedings of International Workshop on Diversity in Document Retrieval (DDR)*, 01 2011. 8

Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 335–344, 2017. 26

Shuo Chen, Joshua L. Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 714–722, 2012a. 11, 22, 24

Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *The Journal of Machine Learning Research*, 13(1):3619–3622, 2012b. 15

Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Sequential recommendation with user memory networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, pages 108–116, New York, NY, USA, 2018. ACM. 32

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111, 2014. 25

Gabriel de Souza Pereira Moreira. CHAMELEON: a deep learning meta-architecture for news recommender systems. In *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018*, pages 578–583, 2018. 12

Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, January 2004. 18

Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2011. 11

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019. 27

Robin Devooght and Hugues Bersini. Long and short-term recommendations with recurrent neural networks. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, UMAP '17, pages 13–21, New York, NY, USA, 2017. ACM. 25

Yi Ding and Xue Li. Time weight collaborative filtering. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 485–492, New York, NY, USA, 2005. ACM. 15

Wouter Duivesteijn and Julia Thaele. Understanding Where Your Classifier Does (Not) Work – The SCaPE Model Class for EMM. In *Proc. ICDM*, pages 809–814. IEEE, 2014. 67

Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. Personalized ranking metric embedding for next new poi recommendation. In *Proc. IJCAI*, pages 2069–2075. AAAI, 2015. 24, 32, 43, 73, 89

Johannes Fürnkranz, Tomás Kliegr, and Heiko Paulheim. On cognitive preferences and the plausibility of rule-based models. *Mach. Learn.*, 109(4):853–898, 2020. 65, 67

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. 38

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016. 25

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013. 25

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019. 67

Frédéric Guillou. *On recommendation systems in a sequential context.* PhD thesis, Université de Lille, 2016. 13

Jon Atle Gulla, Lemei Zhang, Peng Liu, Özlem Özgöbek, and Xiaomeng Su. The adressa dataset for news recommendation. In *Proceedings of the International Conference on Web Intelligence*, WI '17, page 1042–1048, New York, NY, USA, 2017. Association for Computing Machinery. 41

Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1 edition, May 1997. 35, 36

F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM TiiS*, 5(4):19:1–19:19, December 2015. 40, 72

Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *ICDM*, pages 191–200. IEEE, 2016. 24, 32, 34

Ruining He, Wang-Cheng Kang, and Julian McAuley. Translation-based recommendation. In *Proc. RecSys*, pages 161–169, New York, NY, USA, 2017a. ACM. 18, 24, 43, 44, 73, 89

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 173–182, 2017b. 12, 20, 21, 29, 43, 44

Andreas Henelius, Kai Puolamäki, Henrik Boström, Lars Asker, and Panagiotis Papapetrou. A peek into the black box: Exploring classifiers by randomization. *Data Min. Knowl. Disc.*, 28(5-6):1503–1529, 2014. 67

Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, page 230–237, New York, NY, USA, 1999. Association for Computing Machinery. 11

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016a. 18, 25

Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 241–248, 2016b. 26

Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 20

Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee, 2008. 10

Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y. Chang. Improving sequential recommendation with knowledge-enhanced memory networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, pages 505–514, New York, NY, USA, 2018. ACM. 15, 29, 44

Neil Hurley and Mi Zhang. Novelty and diversity in top-n recommendation – analysis and evaluation. *ACM Trans. Internet Technol.*, 10(4), March 2011. 8, 28

Dietmar Jannach and Malte Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 306–310, New York, NY, USA, 2017. ACM. 19, 25, 43

Santosh Kabbur, Xia Ning, and George Karypis. Fism: Factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 659–667, New York, NY, USA, 2013. ACM. 17, 24, 32, 34

Iman Kamehkhosh, Dietmar Jannach, and Malte Ludewig. A comparison of frequent pattern techniques and a deep learning method for session-based recommendation. In *RecTemp@ RecSys*, pages 50–56, 2017. 11

Wang-Cheng Kang and Julian J. McAuley. Self-attentive sequential recommendation. In *Proc. ICDM*, pages 197–206, 2018. 10, 26, 29, 43, 44, 73, 89

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014. 26

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751, 2014. 26

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. 14, 38, 45

Yehuda Koren. 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, 2008. 16, 17

Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 447–456, New York, NY, USA, 2009a. ACM. 15, 32

Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81(2009):1–10, 2009b. 13, 16

Yehuda Koren and Robert M. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 77–118. Springer, 2015. 3, 11, 16, 32

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. 3, 10, 11, 16, 17

Bruce Krulwich. Lifestyle finder: Intelligent user profiling using large-scale demographic data. *AI magazine*, 18(2):37–37, 1997. 11

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 24

Florian Lemmerich, Martin Atzmueller, and Frank Puppe. Fast exhaustive subgroup discovery with numerical target concepts. *Data Min. Knowl. Disc.*, 30(3):711–762, 2016. 65, 70, 72

Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 1419–1428, 2017. 26

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010. 8

Xiaoli Li and Jun Huan. Constructivism learning: A learning paradigm for transparent predictive analytics. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 285–294, 2017. 67

Tie-Yan Liu. *Learning to rank for information retrieval*. Springer Science & Business Media, 2011. 13

Corentin Lonjarret, Marc Plantevit, Céline Robardet, and Roch Auburtin. Recommandation séquentielle à base de séquences fréquentes. In *Extraction et Gestion des connaissances, EGC 2019, Metz, France, January 21-25, 2019*, volume 79, pages 267–272. BoD-Books on Demand, 2019. 5

Corentin Lonjarret, Céline Robardet, Marc Plantevit, Roch Auburtin, and Martin Atzmueller. Why should i trust this item? explaining the recommendations of any model. In *2020 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 526–535. IEEE, 2020. 5, 63

Corentin Lonjarret, Céline Robardet, Marc Plantevit, and Roch Auburtin. Sequential recommendation with metric models based on frequent sequences. *Data Mining and Knowledge Discovery*, 2021. 5, 31, 73, 77

Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011. 12

Jonathan Louëdec, Max Chevalier, Josiane Mothe, Aurélien Garivier, and Sébastien Gerchinovitz. A multiple-play bandit algorithm applied to recommender systems. In *FLAIRS Conference*, pages 67–72, 2015. 8

Malte Ludewig and Dietmar Jannach. Evaluation of session-based recommendation algorithms. *CoRR*, abs/1803.09587, 2018. 19, 21, 22, 43, 45, 46

David R Mandel. Counterfactual and Causal Explanation: From Early Theoretical Views To New Frontiers. In *The Psychology of Counterfactual Thinking*, pages 23–39. Routledge, 2007. 67

Romain Mathonat, Diana Nurbakova, Jean-François Boulicaut, and Mehdi Kaytoue. Seqscout: Using a bandit model to discover interesting subgroups in labeled sequences. In *Proc. DSAA*, pages 81–90, 2019. 65, 70

Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 43–52, 2015. 40, 72

David McSherry. Explanation in recommender systems. *Artificial Intelligence Review*, 24(2): 179–197, 2005. 68

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013. 13

Koji Miyahara and Michael J Pazzani. Collaborative filtering with the simple bayesian classifier. In *Pacific Rim International conference on artificial intelligence*, pages 679–689. Springer, 2000. 11

Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, pages 497–506, Washington, DC, USA, 2011. IEEE Computer Society. 17

James R Norris. *Markov chains*. Cambridge university press, 1998. 21

Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, volume 83, 07 1998. 9

Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *2008 Eighth IEEE International Conference on Data Mining*, pages 502–511. IEEE, 2008. 13, 14

Weike Pan and Li Chen. Gbpr: Group preference based bayesian personalized ranking for one-class collaborative filtering. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013. 10

Dimitris Paraschakis. Recommender systems from an industrial and ethical perspective. In *RecSys'16*, pages 463–466, 2016. 87

Rajiv Pasricha and Julian McAuley. Translation-based factorization machines for sequential recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 63–71, New York, NY, USA, 2018. ACM. 15, 18, 107

Michael J Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial intelligence review*, 13(5-6):393–408, 1999. 11

Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007. 12

Pearl Pu and Li Chen. Trust building with explanation interfaces. In *Proceedings of the 11th International Conference on Intelligent User Interfaces, IUI 2006, Sydney, Australia, January 29 - February 1, 2006*, pages 93–100, 2006. 64

Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*, pages 130–137, 2017. 25

Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. *ACM Comput. Surv.*, 51(4), July 2018. 14, 15, 28

Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010. 17, 18

Steffen Rendle. Factorization machines with libfm. *ACM Trans. Intell. Syst. Technol.*, 3(3): 57:1–57:22, May 2012. 11, 15, 17

Steffen Rendle and Lars Schmidt-Thieme. 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, 2010. 17

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press. 3, 14, 25, 28, 34, 37, 42, 44, 65, 89

Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proc. WWW*, pages 811–820, New York, NY, USA, 2010. ACM. 10, 17, 22, 23, 32, 42, 43, 73, 89

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proc. KDD*, pages 1135–1144, 2016. 4, 64, 67, 68

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-Precision Model-Agnostic Explanations. In *Proc. AAAI*, 2018. 4, 67

Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011. 10

Thomas Roth-Berghofer, Stefan Schulz, Daniel Bahls, and David B. Leake, editors. *Explanation-Aware Computing, Papers from the 2007 AAAI Workshop, Vancouver, British Columbia, Canada, July 22-23, 2007*, volume WS-07-06 of *AAAI Technical Report*, 2007. AAAI Press. 68

Thomas R. Roth-Berghofer and Jörg Cassens. Mapping Goals and Kinds of Explanations to the Knowledge Containers of Case-Based Reasoning Systems. In *Proc. ICCBR*, number 3620 in LNAI, pages 451–464, Berlin/Heidelberg, 2005. Springer. 68

Alan Said and Alejandro Bellogín. Comparative recommender system evaluation: Benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, page 129–136, New York, NY, USA, 2014. Association for Computing Machinery. 29, 44, 102

Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887, 2008. 17

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798, 2007. 11, 19

Pablo Sanchez and Alejandro Bellogín. Time and sequence awareness in similarity metrics for recommendation. *Information Processing & Management*, 57:102228, 05 2020. 15

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295. ACM, 2001. 11

Roger C. Schank. *Explanation Patterns: Understanding Mechanically and Creatively.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1986. 64, 68

Steven L. Scott. Multi-armed bandit experiments in the online service economy. *Applied Stochastic Models in Business and Industry*, 31(1):37–45, January 2015. 30, 103

Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 111–112, New York, NY, USA, 2015. ACM. 19

Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005. 22

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146, 2012. 14, 29

Elena Smirnova and Flavian Vasile. Contextual sequence modeling for recommendation with recurrent neural networks. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pages 2–9, 2017. 26

Frode Sørmo, Jörg Cassens, and Agnar Aamodt. Explanation in case-based reasoning – perspectives and goals. *Artificial Intelligence Review*, 24(2):109–143, 2005. 68

Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. Maximum-margin matrix factorization. In *Advances in neural information processing systems*, pages 1329–1336, 2005. 17

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1441–1450, 2019. 10, 27

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 25

Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016*, pages 17–22, 2016. 25

Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, pages 565–573, New York, NY, USA, 2018. ACM. 10, 26, 32, 43

Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. In *Proc. ICDE workshops*, pages 801–810. IEEE, 2007. 64

Nava Tintarev and Judith Masthoff. Designing and evaluating explanations for recommender systems. In *Recommender systems handbook*, pages 479–510. Springer, 2011. 64, 68

Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *Proc. KDD*, pages 465–474. ACM, 2017. 67

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. 26, 27

Koen Verstrepen and Bart Goethals. Unifying nearest neighbors collaborative filtering. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 177–184, 2014. 11, 19

Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 403–412, New York, NY, USA, 2015. ACM. 23

Shoujin Wang, Longbing Cao, and Yan Wang. A survey on session-based recommender systems. *CoRR*, abs/1902.04864, 2019. 15

Markus Weimer, Alexandros Karatzoglou, and Alex Smola. Improving maximum margin matrix factorization. *Machine Learning*, 72(3):263–276, 2008. 10

Michael R. Wick and William B. Thompson. Reconstructive expert system explanation. *Artificial Intelligence*, 54(1-2):33–70, 1992. 68

Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In *Proc. PKDD*, number 1263 in LNCS, pages 78–87, Berlin/Heidelberg, Germany, 1997. Springer. 65, 69

Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 495–503, New York, NY, USA, 2017. ACM. 15

Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 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, 2016. 19, 20

Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. Attentional factorization machines: Learning the weight of feature interactions via attention networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3119–3125, 2017. 26

Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff G. Schneider, and Jaime G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pages 211–222, 2010. 15

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. 26

Fajie Yuan. *Learning implicit recommenders from massive unobserved feedback*. PhD thesis, University of Glasgow, 2018. 13, 15

Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 227–236, 2016. 28

Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 582–590, 2019. 26

Weihua Yuan, Hong Wang, Xiaomei Yu, Nan Liu, and Zhenghao Li. Attention-based context-aware sequential recommendation model. *Information Sciences*, 510:122–134, 2020. 10

Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 353–362, New York, NY, USA, 2016. ACM. 15

Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1):5:1–5:38, 2019. 20, 25, 32

Tong Zhao, Julian J. McAuley, and Irwin King. Leveraging social connections to improve personalized ranking for collaborative filtering. In *CIKM*, pages 261–270, 2014. 41

FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

NOM : LONJARRET
(avec précision du nom de jeune fille, le cas échéant)

DATE de SOUTENANCE : 12/01/20201

Prénoms : Corentin

TITRE : Sequential recommendation and explanations

NATURE : Doctorat

Numéro d'ordre : 2021LYSEI003

Ecole doctorale : InfoMaths (ED 512)

Spécialité : Informatique

RESUME : Ces dernière années, les systèmes de recommandation ont reçu beaucoup d'attention avec l'élaboration de nombreuses propositions qui tirent parti des nouvelles avancées dans les domaines du Machine Learning et du Deep Learning. Grâce à l'automatisation de la collecte des données des actions des utilisateurs tels que l'achat d'un objet, le visionnage d'un film ou le clic sur un article de presse, les systèmes de recommandation ont accès à de plus en plus d'information. Ces données sont des retours implicites des utilisateurs (appelé «~implicit feedback~» en anglais) et permettent de conserver l'ordre séquentiel des actions de l'utilisateur. C'est dans ce contexte qu'ont émergé les systèmes de recommandations qui prennent en compte l'aspect séquentiel des données. Le but de ces approches est de combiner les préférences des utilisateurs (le goût général de l'utilisateur) et la dynamique séquentielle (les tendances à court terme des actions de l'utilisateur) afin de prévoir la ou les prochaines actions d'un utilisateur.
Dans cette thèse, nous étudions la recommandation séquentielle qui vise à prédire le prochain article/action de l'utilisateur à partir des retours implicites des utilisateurs. Notre principale contribution, REBUS, est un nouveau modèle dans lequel seuls les items sont projetés dans un espace euclidien d'une manière qui intègre et unifie les préférences de l'utilisateur et la dynamique séquentielle. Pour saisir la dynamique séquentielle, REBUS utilise des séquences fréquentes afin de capturer des chaînes de Markov d'ordre personnalisé. Nous avons mené une étude empirique approfondie et démontré que notre modèle surpasse les performances des différents modèles de l'état de l'art, en particulier sur des jeux de données éparses. Nous avons également intégré REBUS dans myCADservices, une plateforme collaborative de la société française Visiativ. Nous présentons notre retour d'expérience sur cette mise en production du fruit de nos travaux de recherche.
Enfin, nous avons proposé une nouvelle approche pour expliquer les recommandations fournies aux utilisateurs. Le fait de pouvoir expliquer une recommandation permet de contribuer à accroître la confiance qu'un utilisateur peut avoir dans un système de recommandation. Notre approche est basée sur la découverte de sous-groupes pour fournir des explications interprétables d'une recommandation pour tous types de modèles qui utilisent comme données d'entrée les retours implicites des utilisateurs.

MOTS-CLÉS : Système de recommandation, Filtrage collaboratif, recommandation séquentielle, Explications

Laboratoire (s) de recherche : Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS)

Directeur de thèse :
Céline Robardet (Professeur des Universités, INSA de Lyon)
Marc Plantevit (Maître de conférences HDR, Université Claude Bernard Lyon 1)

Président de jury :

Composition du jury :
Josiane Mothe (Professeur des Universités, INSPE de l'Académie de Toulouse)
Arnaud Soulet (Maître de conférences HDR, Université de Tours)
Sihem Amer-Yahia (Directrice de recherche, CNRS)
Elisa Fromont (Professeur des Universités, Université Rennes 1)