On Recommendation Systems in a Sequential Context

Frédéric Guillou

Université de Lille 3 Sciences Humaines et Sociales

École Doctorale Sciences Pour l'Ingénieur

THÈSE DE DOCTORAT

préparée au sein de CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille
et du centre de recherche Inria Lille - Nord Europe

co-financée par Inria et Région Hauts-de-France

Spécialité : **Informatique**

présentée par

Frédéric GUILLOU

ON RECOMMENDATION SYSTEMS IN A SEQUENTIAL CONTEXT

sous la direction de **Philippe PREUX**
et le co-encadrement de **Romaric GAUDEL** et **Jérémie MARY**

Soutenue publiquement à **Villeneuve d'Ascq**, le **02 décembre 2016** devant le jury composé de :

M. Massih-Reza **AMINI**	Université Grenoble Alpes	Rapporteur
M. Younès **BENNANI**	Université Paris 13	Rapporteur
M. Romaric **GAUDEL**	Université Lille 3	Examinateur
M. Jérémie **MARY**	Université Lille 3	Examinateur
M. Mohamed **NADIF**	Université Paris-Descartes	Examinateur
M. Vianney **PERCHET**	ENS Cachan	Examinateur
M. Philippe **PREUX**	Université Lille 3	Directeur

UNIVERSITÉ LILLE 3

Abstract

On Recommendation Systems in a Sequential Context

by Frédéric GUILLOU

This thesis is dedicated to the study of Recommendation Systems under a sequential setting, where the feedback given by users on items arrive one after another in the system. After each feedback, the system has to integrate it and try to improve future recommendations. Many techniques or evaluation methods have already been proposed to study the recommendation problem. Despite that, such sequential setting, which is more realistic and represent a closer framework to a real Recommendation System evaluation, has surprisingly been left aside. Under a sequential context, recommendation techniques need to take into consideration several aspects which are not visible for a fixed setting. The first one is the exploration-exploitation dilemma: the model making recommendations needs to find a good balance between gathering information about users' tastes or items through exploratory recommendation steps, and exploiting its current knowledge of the users and items to try to maximize the feedback received. We highlight the importance of this point through the first evaluation study and propose a simple yet efficient approach to make effective recommendation, based on Matrix Factorization and Multi-Armed Bandit algorithms. The second aspect emphasized by the sequential context appears when a list of items is recommended to the user instead of a single item. In such a case, the feedback given by the user includes two parts: the explicit feedback as the rating, but also the implicit feedback given by clicking (or not clicking) on other items of the list. By integrating both feedback into a Matrix Factorization model, we propose an approach which can suggest better ranked list of items, and we evaluate it in a particular setting.

Keywords. Recommendation Systems, Sequential Recommendation, Collaborative Filtering, Matrix Factorization, Multi-Armed Bandits, Sequential Feedback, Learning to Rank

UNIVERSITÉ LILLE 3

Résumé

Des Systèmes de Recommandation dans un Contexte Séquentiel

par Frédéric GUILLOU

Cette thèse porte sur l'étude des Systèmes de Recommandation dans un cadre séquentiel, où les retours des utilisateurs sur des articles arrivent dans le système l'un après l'autre. Après chaque retour utilisateur, le système doit le prendre en compte afin d'améliorer les recommandations futures. De nombreuses techniques de recommandation ou méthodologies d'évaluation ont été proposées par le passé pour les problèmes de recommandation. Malgré cela, l'évaluation séquentielle, qui est pourtant plus réaliste et se rapproche davantage du cadre d'évaluation d'un vrai système de recommandation, a été laissée de côté. Le contexte séquentiel nécessite de prendre en considération différents aspects non visibles dans un contexte fixe. Le premier de ces aspects est le dilemme dit d'exploration vs. exploitation: le modèle effectuant les recommandations doit trouver le bon compromis entre recueillir de l'information sur les goûts des utilisateurs à travers des étapes d'exploration, et exploiter la connaissance qu'il a à l'heure actuelle pour maximiser le feedback reçu. L'importance de ce premier point est mise en avant à travers une première évaluation, et nous proposons une approche à la fois simple et efficace, basée sur la Factorisation de Matrice et un algorithme de Bandit Manchot, pour produire des recommandations appropriées. Le second aspect pouvant apparaître dans le cadre séquentiel surgit dans le cas où une liste ordonnée d'articles est recommandée au lieu d'un seul article. Dans cette situation, le feedback donné par l'utilisateur est multiple: la partie explicite concerne la note donnée par l'utilisateur concernant l'article choisi, tandis que la partie implicite concerne les articles cliqués (ou non cliqués) parmi les articles de la liste. En intégrant les deux parties du feedback dans un modèle d'apprentissage, nous proposons une approche basée sur la Factorisation de Matrice, qui peut recommander de meilleures listes ordonnées d'articles, et nous évaluons cette approche dans un contexte séquentiel particulier pour montrer son efficacité.

Keywords. Systèmes de Recommandation, Recommandation Séquentielle, Filtrage Collaboratif, Factorisation de Matrice, Bandit Manchot, Feedback Séquentiel, Apprentissage de Classement

Acknowledgements

First, I would like to express my gratitude to my supervisors, Philippe Preux, Romaric Gaudel and Jérémie Mary. I am grateful for your guidance, your patience, and for the research insight or all advices shared with me during these three years. The help and encouragement you provided allowed me to pursue my research and finish this thesis, and I thank you for it.

Then, I would like to thank Massih-Reza Amini and Younès Bennani who kindly agreed to review my PhD thesis, as well as Mohamed Nadif and Vianney Perchet for having accepted to be part of my committee.

Thanks to Professor Jo at Inha University, where I got my first research experience, and which encouraged me to start a PhD.

Merci à toute l'équipe SequeL et ses membres (passés ou présents) pour leur sympathie et aide: Amir, Gergely, Prashanth, Balázs, Alessandro, Bilal, Marta, Hadrien, Michal, Tomáš, Daniele (special thanks for the useful script on the Grid!), Florian, Ronan, Julien, Alexandre, Jean-Bastien, Emilie, Olivier, Daniil, ainsi qu'Amélie.

Merci à l'Inria pour le soutien financier et logistique, au Ministère de la Recherche et de l'Enseignement Supérieur, au FUI Hermès et à la région des Hauts-de-France pour leur participation financière à ma thèse.

Merci à l'équipe pédagogique de l'université Lille 3 pour son accueil et leur bonne collaboration durant mes activités d'enseignement.

Merci à la plateforme d'essai Grid'5000, soutenue par un groupement d'intérêt scientifique incluant l'INRIA, le CNRS, RENATER et plusieurs universités ainsi que des organisations (voir https://www.grid5000.fr), qui m'a permis de réaliser la plupart des expériences rapportées dans cette thèse.

Merci à toute ma famille pour le soutien et les encouragements. Special thanks to 현정 who gave me a lot of support during these last two years of my PhD.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivations

Despite their omnipresence online on e-commerce, news, social website, **Recommendation Systems** are still part of a relatively recent field. The research about this topic indeed started to grow significantly at the end of the 90's (Resnick and Varian, 1997) with the growth of e-commerce applications, and even more since the Netflix Challenge (Bennett et al., 2007) which was only set up ten years ago. The importance of implementing accurate Recommendation Systems has been widely acknowledged by the industrial world in the last two decades, and it has quickly become one of the most popular applications of Machine Learning or Data Mining techniques.

Facing the abundance of products and information available to him, the customer usually feels disoriented about making a choice, and implementing an efficient system that can help and guide the customer toward items he would like can translate for the company in a great increase of the traffic or sales. With the era of Big Data, making effective systems has become a complex task due to the high number of users, items and information available. Two main approaches have emerged to solve this problem. The first type of recommendation technique is based on existing information about the users or items, like demographic information or textual description. This is referred as Content-Based Recommendation (Pazzani and Billsus, 2007). The second type exclusively relies on the feedback collected from the past and is referred as Collaborative Filtering (Ekstrand et al., 2011). The idea behind it is that users with similar behavior (based on their feedback) in the past will also have similar behavior in the future. Among Collaborative Filtering approaches, Matrix Factorization methods (Koren et al., 2009) have been the major focus on the Recommendation Systems field, because they allow for a high level of abstraction inside the recommendation models, while being able to provide accurate suggestions to users.

Aside from Recommendation Systems, *Reinforcement Learning (RL)* (Sutton and Barto, 1998) has been very recently under the spotlight among Machine Learning approaches due to some dazzling success (combined with the use of Deep Learning) in several public-friendly domains, such as video games or the game of Go. Programs able to play Atari Games better than

human (Mnih et al., 2013), or capable of playing Go against human and beat them with the example of AlphaGo (Silver et al., 2016), have been implemented by formulating the problem trying to be solved as a RL problem and requiring the program to learn "how" to play properly.

Reinforcement Learning methods more generally target at solving sequential decision-making problems, where the decision is taken under uncertainty. In a typical RL setting, an agent is considered to interact with its environment. At each time step, the "system" made of the agent and its environment is in a given state. The agent perceives this state, and decides on an action to perform; once the action has been performed, the agent observes the consequences of its action and the new state. Technically, these consequences are named a "return": it may be beneficial to the agent (a "reward") or not (a "cost"). The state transition function is unknown to the agent, as well as the mapping from a (state, action) pair to the return. These two functions (the transition function, and the return function) are unknown to the agent and are stochastic. Usually, these functions are constant in time (stationary). A fundamental assumption is that the state contains all the information necessary to find the best action to perform in this state. If not, the problem is said to be "partially observable". The goal is for the agent to learn an optimal policy, which is a mapping from the set of states to the set of actions; the policy is optimal in the sense that some functions of the returns are optimized.

In this setting, the agent needs to learn from the consequences of its actions rater than being taught explicitly as would happen in a supervised learning scenario.

The Figure 1.1 describes the RL setup.

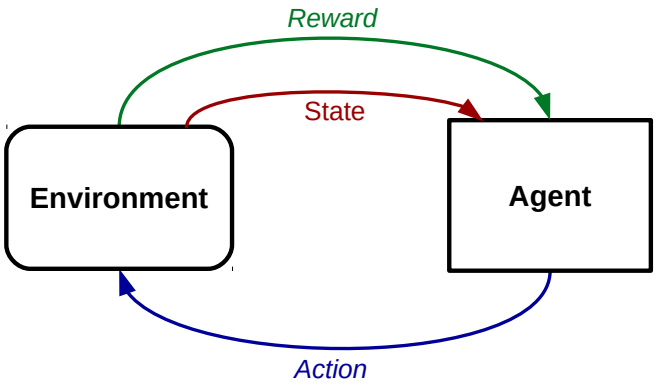


FIGURE 1.1: Diagram explaining Reinforcement Learning.

From the reward signal given by the environment, the goal for the agent is then to find an optimal policy, meaning the agent needs to find the sequence of actions such that the reward obtained through time will be maximized. Reinforcement Learning problems are usually modelled as Markov Decision Process (MDP) and solved using Reinforcement Learning algorithms or Dynamic Programming (Szepesvári, 2010; Gosavi, 2009).

A specific sub-field of the Reinforcement Learning is known as *Multi-Armed Bandits*. Similarly to the standard RL setting, the agent also learns from the reward received from the environment, but in this case the MDP does not have any state transitions: the environment is only formed of one state, and the reward only depends on which action is taken by the learning agent. In such a case, the agent only has to explore the set of actions and find the best set of actions to maximize the reward, but there is no notion of a sequence of states anymore.

Despite its recent successes, the field of applications for RL is still limited to top-end research companies in specific domains such as military applications (helicopter control), robotics, or games (Go, poker, video games...), but most commercial companies using Machine Learning applications do not view the problem they target from a Reinforcement Learning point of view, either due to their unawareness of the RL aspect of the problem they tackle, or due to the complexity of applying such approaches.

While many approaches have been proposed for Recommendation Systems, and even though the view of the industrial world have recently acknowledged the importance of RL in applications, there has yet been little to no study about fitting the recommendation problem into a Reinforcement Learning setting (Shani et al., 2005). And yet, the RL approach would suit perfectly to embed the one characteristic which remains difficult to grasp and implement in a real-world Recommendation System: the need for adaptability.

In the field of Recommendation Systems, some trends are emerging every day while others are fading away, new customers are starting to use the system, some customer's tastes change through time or depending on the context or location, some new items are released while other disappear from the recommendation application... In some applications like news recommendation, the system needs to adapt at every moment given the fickleness of trendy news. Moreover, the recommendations made by the system itself also potentially influence the user and can change his tastes, which is an highly complex effect to quantify. Surprisingly, most researches still study the recommendation problem within a fixed, batch setting perspective, where the data used to learn the model and the data used to make the evaluation are split and fixed, while using a RL setting would allow to tackle the issue coming from the inherent online and sequential context of the recommendation problem.

We focus on this thesis on the aspects related to sequential recommendation and set up an offline sequential evaluation methodology accordingly. Consequently from studying recommendation as a sequential process, some important aspects inherent to the Recommendation Systems appear. Due to the traditional evaluation settings using a batch setting, only little research has been made on these aspects, and we propose here simple techniques to take them into account in the implementation of the Recommendation System.

We study specifically two aspects of Recommendation Systems ensuing from the sequential context:

1. The **exploration-exploitation dilemma**: this is a direct consequence of a RL setting. In a RL problem, the learning agent needs to explore states and actions in order to gather feedback as reward, but also exploit the knowledge it gathered until now to optimize its policy. We do not consider in this thesis the system as a Markov Decision Process which is the complex form of Reinforcement Learning, but restrict to the point of view of the Multi-Armed Bandit setting, where there is only one state. The RS here only has to tackle the exploration-exploitation dilemma about the set of actions to take (*i.e.* about the set of recommendation to make to users).

   A good example of exploration-exploitation dilemma emerges when a new customer (or item) enters the system: at this point in time, there is little to no knowledge available about him (or it). This issue is known as the **cold-start issue**. In the Recommendation System case, the dilemma can be seen as finding the balance between the exploration, needed to discover the user's tastes (or the item's features, to know to which kind of user it would be suitable to recommend it), and the exploitation, which means recommending items according to the current information available (possibly not entirely reliable since the knowledge about users or items is not perfect). This dilemma also appears for long-term users, because their tastes can change through time, and exploration is then needed to avoid recommending following the past interests of the user which are not conform to his current tastes.

2. The **impact of the feedback** given by users. Recommendation models need to learn and adapt quickly to the user's tastes to make accurate suggestions. For this reason, any feedback bringing information has to be taken into consideration. Some Recommendation Systems are based only on feedback given by the user in the form of a rating on items, while other systems also make use of implicit feedback gathered from the webpage (for instance clicks on items, or the time spent by the user on the page). Some feedback implied by the behavior of the user can increase the convergence of the learning for the algorithm. A good example is when a Recommendation Systems is displaying a ranked list of items from top to bottom of the screen to the user (such as Youtube videos for example): in such a case, the feedback given by clicking on one item in this list implicitly implies a preference on this item over non-clicked items which were viewed higher on the list, since the user scrolled down without clicking on them. The use of this information inside the learning model of the Recommendation System can lead to lists of items ranked more appropriately for future recommendations, and increase user's satisfaction.

Even though these two aspects are studied separately in this thesis, they are both related to the need of adaptability for real-world Recommendation Systems, and are attempting at reducing the cold start effect by learning faster and more efficiently when only a small fraction of information is available. They also highlight the need of a sequential evaluation to exhibit such effect, since a fixed evaluation setting would not be relevant to study them.

1.2 Overview of the thesis

This thesis is structured as follows:

- The Chapter 2 gives a definition of the recommendation problem, the type of feedback possibly received by the system, the various goals that need to be achieved by the system, and the field of applications in which Recommendation Systems are used. A detailed state of the art about recommendations techniques is given, with a special attention given to Collaborative Filtering methods based on Matrix Factorization, which are used in other chapters of this thesis. A section is finally dedicated to evaluation metrics and scenarios used traditionally.
- The Chapter 3 studies Recommendation Systems from a sequential point of view and brings another more realistic evaluation methodology. By incorporating Multi-Armed Bandits algorithm into a Collaborative Filtering approach, we present a simple yet efficient algorithm to tackle the recommendation problem in such a sequential context. We make extensive experiments on large datasets to show that the approach can perform accurate recommendations in a very short time. Since the model is studied sequentially and needs to be updated, we also study how the performance is affected depending on how often or on how much data the update is performed.
- Chapter 4 takes another point of view by considering an expanded setting, where a list of items is suggested to the user instead of a single item. The recommendation setting brings the need to tackle the problem from a ranking point of view, as items need to be ordered correctly, to put items preferred by the user on the top of the recommended list. By representing the interaction between the user and the RS in the form of a click on items in the recommended list, the setting also raises other questions and brings new possibilities to tackle the recommendation issue more efficiently, by considering the feedback on both the item chosen by the user and those not chosen among the list. We integrate these possibilities into our approach and provide a method which outperforms several state of the art algorithms.
- Chapter 5 gives a discussion on difficulties and challenges related to real data and evaluation. The first part of this chapter highlights some intrinsic aspects of Recommendation Systems and the issues emerging when trying to evaluate them. Then, we set up a more realistic setting than the one described in Chapter 3 for an offline sequential evaluation, and discuss about the results. Finally, some insight is given on aspects to be careful about when building real-world Recommendation Sytems. This insight is built from our successful paper in a Recommendation System Challenge (Guillou et al., 2014), on a real-world dataset.
- Finally, some conclusions are drawn on Chapter 5, and lines of future work are drawn, to possibly extend research beyond the frame of this thesis.

Chapter 2

Recommendation Systems

In this chapter, we first present and define Recommendation Systems (RS): their goals, the variety of data they use for their tasks, and their applications. An overview is given over the wide range of approaches traditionally applied to perform recommendation. An emphasis is especially put on Collaborative Filtering (CF) methods based on Matrix Factorization, which are the most popular among RS methods, and will be used in our approaches in the following chapters of this thesis. The second focus is put on evaluation metrics and scenarios commonly used in the field of RS, since our evaluation scheme will distinctly differ from these traditional methodologies.

Contents

2.1 Definition: The Recommendation Problem

Recommendation (or Recommender) Systems (RS) are techniques or software programs which aim at providing accurate or useful suggestions of some items to a user (Resnick and Varian, 1997). *User* is the general term to designate the entity to which the recommendation is provided, while *Item* labels the product being recommended. Depending on the field of application for the system, the user is usually an individual, but this term can also designate a group of people, a company... while recommended items can be as various as a computer to buy, a movie to watch, a song to listen to, a news to read, etc.

After the user navigated on the RS or accepted a recommendation, some feedback (a rating for example) is given to the system about the item purchased or consumed. In order to suggest more personalized and appropriate recommendations in future appearances of this user, the RS will make use of all past interactions and feedback of the user. If available, the system can potentially use as input other diverse sources of data, such as information about the user or item. For instance, demographic information given by the user to fill in his profile, like his age, gender, profession can be used as *User Model*, while information about the item like the genre of a movie or its director and actors can be used to build an *Item Model*. Social network of a given user and his relations to other users can also be a source of information. Finally, some information coming from the context, like the location of the user or temporal information (the season, the current time of the day...) can be included to improve the model and the recommendation accuracy. A survey about the use of this additional information is available in (Shi et al., 2014). However, the main information leading to personalized recommendation to the user resides in the feedback he gives to the RS.

2.1.1 Feedback

Once an item has been recommended, a feedback is usually requested from the RS to the user, in order to improve its model and make future recommendations more accurate. The feedback given by a user to the RS can take various forms and is usually divided into two sorts:

- The **explicit Feedback** is a feedback directly given by the user to the system: the user's involvement is thus required to obtain such information. It is often given in the form of a *rating* on a numerical scale such as the one-five stars scale often used in movie RS. Other forms of explicit feedback exist (Schafer et al., 2007): a unary (or binary) answer such as a "like" (and possibly "dislike") button, or on an ordinal scale like "strongly disagree / disagree / neutral / agree / strongly agree" answer to a questionnaire. Another possible explicit feedback consists in a tag or a text comment about an item: this is possibly a more helpful form of feedback for the system or other users as it contains more information than a simple rating. However, such type of feedback is more complex to integrate into the model.

- The **implicit Feedback** (Oard et al., 1998) can be collected without the user giving directly his opinion to the system. It can be inferred from any detectable user's behavior, such as clicking on an item, bookmarking a page or news, putting an item into the basket, watching a video or listening to a song entirely... The main advantage of implicit feedback is that it does not require the user's involvement, but it can be more noisy than explicit feedback and needs to be treated more carefully.

Notice that implicit feedback is more abundant than explicit feedback due to its nature, but it is also less reliable as the true user's preferences are not directly expressed. Moreover, a feedback (especially implicit one) usually needs to be preprocessed to be used correctly by algorithms.

2.1.2 Representation and Solutions

The feedback given by the user is usually represented in the form of a matrix of ratings, where each user is a row of the matrix, each item is a column of the matrix, and the entry in the matrix represents the rating given by the user on a given item.

We now introduce some notations for the following sections about the recommendation problem. We consider :

- $\mathcal{N} = \{u_1, \ldots, u_N\}$ is the set of N users,
- $\mathcal{M} = \{i_1, \ldots, i_M\}$ is the set of M items,
- $\mathbf{R}$ of size $N \times M$ is the matrix of ratings representing users' preferences,
- $r_{u,i}$ in $\mathbf{R}$ represents the tastes of user u with regards to item i,
- $\mathcal{S}$ designates the set of known entries of $\mathbf{R}$,
- $\mathcal{I}(u)$ is the set of items i for which $(u, i) \in \mathcal{S}$ (the items for which the user u gave a rating),
- $\mathcal{J}(i)$ is the set of users u for which $(u, i) \in \mathcal{S}$ (the users who have rated item i).

The illustration in Figure 2.1 displays a recommendation scenario with $N = 5$ and $M = 6$.

	i_1	i_2	i_3	i_4	i_5	i_6
u_1	5	5		1	2	4
u_2			3	4		
u_3	3	1	4		2	
u_4		5		1	3	4
u_5	4	3		4		5

$\mathbf{R}$

FIGURE 2.1: Example of a matrix of preferences used in a RS, for five users and six items and with a 1 to 5 stars rating.

In this example, $\mathcal{I}(u_3) = \{i_1, i_2, i_3, i_5\}$ and $\mathcal{J}(i_3) = \{u_2, u_3\}$. An empty rating in the matrix signifies that no preference were given by the user on the item (for example, user u_1 did not indicate his preference about item i_3). The matrix of ratings is actually usually very sparse: the number of items and users in a real-word RS grows very large, and a user will give his preferences about a small fraction of all the items in the system.

From all the information available, the RS has to maximize a utility function f over the set L_u of items which can be recommended for each user u. This function predicts how much a user would like the items. Algorithm 1 displays an algorithm for recommendation.

Algorithm 1: A recommendation algorithm.

Input: the user u,
the set of possible items to recommend L_u,
the size of the recommendation list s

1 1. Predict the score of the utility function $f(u, i)$ for each item $i \in L_u$;
2 2. Create a list L'_u by selecting the top s items with the highest value for $f(u, i)$;
3 3. Return L'_u to recommend;

The set of possible items to recommend depends on the type of application, and an item already rated does not always imply it cannot be recommended again. In a movie RS for example, it is not really advisable to recommend movies for which the feedback has already been given recently, as it is unlikely that people would want to watch the same movie one more time right after watching it. On the opposite, a music RS can recommend a song for which a high rating has recently been given, as the user would probably like to listen to the song again if it suits his mood or current tastes.

Once the RS has a user and the list of possible items to recommend as input, the recommendation problem is usually solved following two possible sort of techniques:

- The first way is by doing *Matrix completion*, which means filling the empty entries in the matrix of ratings. This is the most widespread approach. The system will first make a *prediction* of the rating value for every recommendable item for the target user, and then provide the recommendation based on these predictions. In this case, the utility function in the Algorithm 1 attempts to predict the rating.
- Another way is by using *Ranking approaches*, which is detailed in Section 2.7. These approaches tackle the problem from a different perspective, assuming there is no need to predict a rating for all items. In fact, only finding the correct order of preferences between items which are being recommended is sufficient to satisfy the user. Ranking approaches and evaluation are starting to gain ground due to the flaws noticed in predictive evaluation (cf. Section 2.6 for more details).

Note that it is possible to create a ranking approach from the first method which predicts ratings, simply by ranking items based on the predicted ratings. However, there are more natural methods directly targeting at optimizing the order of recommended items. Regardless of the perspective used, all Recommendation Systems aim to achieve the same goals which are detailed in the next section.

2.1.3 Goals of Recommendation Systems

From a business point of view, the primary goal of a RS is of course to increase the profit of the company, through a higher number of sales. To reach this objective, the system has to be able to meet some requirements:

- *Know what the user wants / Relevance*: quite obviously, the RS has to suggest items that are relevant to the users' tastes, because users are more likely to buy or consume items they have interest in. However, solving this task alone is not sufficient to satisfy a user, as it is explained in the next point.
- *Diversity, Novelty and Serendipity*: firstly, if the RS always suggests items of the same sort to a user, there is a risk that this user would get bored or would not like any item. Thus, the system has to pay attention to put items of different types, bringing diversity. Advertising more diverse items also helps the system to gather feedback about a wider set of products, and can be beneficial for future recommendation.

  Secondly, even if it can be relevant, the recommended item also has to be something the user has not already bought or experienced in the past: some novelty is required.

  Lastly, the serendipity implies to surprise the user by recommending items he does not expect. Compared to novelty, the suggested item would belong to a category the user did not expect at all. This can sometimes lead the user to widen its area of interest, and help to increase sales diversity. Serendipity is tightly related to the exploration/exploitation dilemma addressed in Chapter 3.
- *User satisfaction and fidelity*: finally, another goal of the RS is to increase the user satisfaction and fidelity. A good user interface and accurate recommendations might encourage the user to connect and use the site again. Giving explanations about why a specific recommendation has been provided is also usually appreciated by users. User fidelity to a RS implies more feedback is received from this user, leading to more refined knowledge of his tastes and consequently to better recommendations for this particular user.

These key goals are common to all RS in any application case, and they need to be integrated carefully during the implementation of the system.

2.1.4 Applications

The range of applications where recommendations can be done is wide and diverse (Schafer et al., 2001). Here are the main fields in which recommendation are applied nowadays:

1. E-commerce: the system is recommending to consumers some products they are likely to buy, like CDs or books (Linden et al., 2003).
2. Entertainment: the system recommends to the user items like movies (Miller et al., 2003) , music... The profit is usually made from advertising or subscription to the website.
3. Services: for example, the recommendation of apartments to rent, of doctors, of travel packages. The service usually makes money by taking some fees out of the payment.
4. Social: recommendation of possible friends or online dating are the two main uses of social recommendation. The profit is usually made through advertising or subscription.
5. Content: examples are online news (Das et al., 2007), recommendation of web pages, tweets (Chen et al., 2012), or display of ads.

The Table 2.1 displays a list of examples, with famous companies where RS are used and the categories of items they recommend.

TABLE 2.1: Some examples of products to recommend and some companies using RS for these products.

Item to recommend	Company
Movies, TV shows	Netflix, IMDb
Books, DVDs, Electronic supplies...	Amazon
Online videos	YouTube
News	Google News
Music	Deezer, Pandora, last.fm
Restaurants	Yelp
Travel	Tripadvisor
Room, Apartments	Airbnb
Advertisements	Google Search
Friends	Facebook
Job offers or People	LinkedIn

Each of these RS has its own characteristics and specific goals depending on items to offer, and they all differ on the information available to build the RS: sometimes the RS only uses the feedback, in other cases there can be demographic information or item information. The feedback received from users also depends on the application (for Music, it will usually be a "like/dislike" explicit feedback and some implicit feedback from the listening behavior on songs, while for travel, it will be an explicit feedback made of a textual comment and one or several ratings on a ordinal scale).

Finally, to conclude this section, the Figure 2.2 makes an overview of the recommendation process.

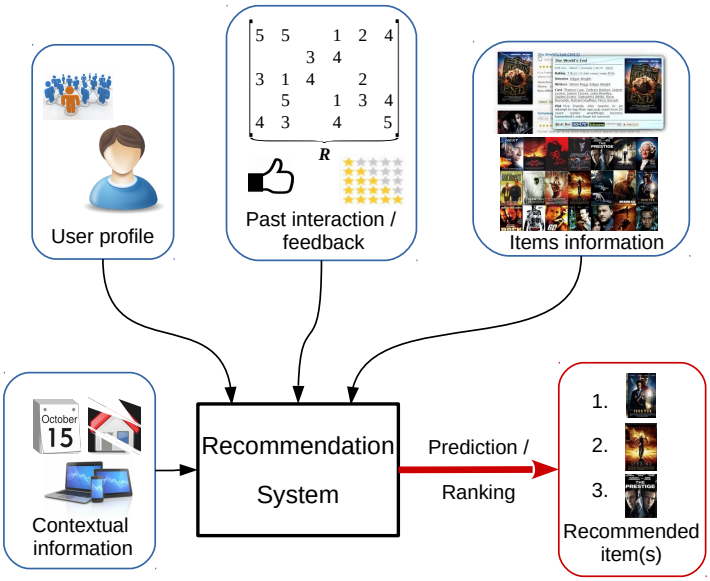


FIGURE 2.2: The recommendation process, with every type of information the RS can eventually use.

All frames in blue represent the possible information taken as input, but recommendation approaches do not necessarily use all of them. For example, the Collaborative Filtering approaches usually only make their recommendations based on previous feedback. The red frame represents the output of the RS. In this case, it consists in a list large of three items, but the size of the list can potentially go from one to a high number of items.

In the next section, we now focus on the techniques used in the black box in this Figure, which will make the prediction or Ranking. We give an overview over traditional recommendation approaches in the next section.

2.2 Approaches to Recommendation

A large range of methods have been proposed to make accurate recommendations. We give in this section a brief overview of the main categories of algorithms, except the Collaborative Filtering (CF) methods using Matrix Factorization. A specific section will be dedicated to these methods, as they are the most successful and are also used in our approaches in the following chapters. As this is only a short overview over the classical approaches, the reader interested in more details can look into (Adomavicius and Tuzhilin, 2005; Shapira et al., 2011).

2.2.1 Content-Based Filtering

Content-Based Filtering methods use external information about the items, such as keywords, tags, or profile written as texts in actual language to perform recommendation (Lops et al., 2011). By using item features, the system can provide recommendations of non-experienced items that have a similar thematic to those he liked in the past. For example, if a user gives a positive feedback about a musical artist where the description contains the tags "rock" and "90s", the RS is able to recommend other artists with the exact same tags (or similar ones), because the user will be likely to have an interest in them too.

Content-based approaches make an analysis of the set of items for which the user expressed a preference in the past. Based on the description of these items, these approaches can build a profile of the user's interests. Then, this user profile is matched against other items' description to decide which item to recommend.

There are several advantages using these approaches:

- They only make use of the ratings of the target user to perform recommendation, they do not need ratings from other users.
- A new item in the RS can be recommended based on its features, even if it has not been rated by any user. This solves the issue called item cold-start (cf. Section 2.4).
- It is easy to provide an explanation to the user about why this item has been recommended, since the content about the items like keywords or textual profile is accessible to the system.

However, Content-Based recommendation techniques also come with important drawbacks:

- These methods cannot perform good recommendation for new users who have perform little to no recommendation, as they need to have information about which items the user likes before the first recommendation. This is referred as user cold-start.
- The serendipity goal cannot be reached with such approaches, as they tend to over-specialize over a set of items with similar content for a given user.
- Finally, the biggest issue is that a good knowledge over the domain of items is required to create the content used in the recommendation process. Sometimes, only a partial knowledge can be captured by the content, or even worse, it is also possible that no information is available at all.

For more information about Content-Based Filtering, see (Pazzani and Billsus, 2007; Lops et al., 2011)

2.2.2 Demographic Filtering

This type of systems assumes the possibility of partitioning the set of users based on their demographic profile (Krulwich, 1997; Pazzani, 1999). The demographic features such as the country or age of each user will decide to which class he belongs to. Then, a set of rules decides which recommendation to perform depending on the class to which the user belongs.

This type of approaches is similar to classification or regression techniques in which the input features are the demographic features and the output is the user's preferences. They usually do not perform very well as users' tastes cannot be inferred solely from their demographic features, but they can add some predictive power if combined with other methods, in hybrid methods for example (cf. Section 2.4.1).

2.2.3 Knowledge-Based Recommendation

A Knowledge-Based RS (Trewin, 2000) acts in a specific context, where the user is requesting a specific content he wants. This adds a constraint to which recommendation has to be provided. These methods are particularly used in applications where the interaction with the user is rare, like RS about cars or travel packages, since in most cases the user will only use the Recommendation System one to very few times. They aim at solving the lack of ratings given by the user by asking him to give some constraints to restrict the set of possible recommendations.

Methods used for Knowledge-Based recommendation are close to the ones used in Content-Based recommendation, with the main difference being the previous feedback or interactions of the user would not be useful as in the Content-Based case. The system will rather find similarity between the user's current request or needs, and the description of the items.

2.2.4 Neighborhood-Based Collaborative Filtering

These approaches are part of Collaborative Filtering (CF) algorithms, but are often referred as "Memory-based" methods, as opposed to the "Model-based" methods among which lies Matrix Factorization techniques.

The basic idea behind CF, whether the approach is Memory-based or Model-based, is that users who adopted the same behavior in the past will also tend to agree in the future. The term "Collaborative" is used here to signify that the model will discover underlying relations between users and items, and use it to provide personalized recommendations for each user.

CF methods usually only rely on past interactions and feedback (either implicit or explicit) to build their model, which explains their popularity in the RS field. Compared to previous methods like Demographic or Content-Based Filtering, no additional information or knowledge about items is needed to perform recommendation. Another advantage is that the effectiveness of the RS also increases as the user provides more feedback.

Neighborhood-Based Approaches were among the first approaches at the beginning of the research on RS. They are usually defined in two ways (Desrosiers and Karypis, 2011):

- **User-based CF** The assumption here is that similar users share similar interests, and as a consequence, the rating for a user u on an item i which has not been experienced yet can be estimated from the ratings of users similar to u. These similar users are called "neighbors". To discover neighbors, a similarity function is used between the active user and others (*e.g.* the rows of the rating matrix).
- **Item-based CF** (Deshpande and Karypis, 2004) In a similar way, one can estimate the rating for a user u on the item i from the ratings given by the user on items similar to i. In this case, the "neighbors" are the items similar to i. They are found based on the similarity score computed between the current item and other items rated by the user (*e.g.* the columns of the rating matrix).

After the closest neighbors have been discovered, the score for an item is computed by weighting every neighbor's rating by its similarity score. For example, with the User-based method, the predicted rating $\hat{r}_{u,i}$ would be

$$\hat{r}_{u,i} = \frac{\sum_{v \in N(u)} sim(u,v) * r_{v,i}}{\sum_{v \in N(u)} sim(u,v)}, \tag{2.1}$$

where $N(u)$ represents the set of neighbors of u, and $sim(u,v)$ is the score of similarity between user u and user v. A close neighbor will then have more importance in the final prediction.

Similarity measures play here a key role, as they are both used to find neighbors and act as weight in the prediction of the rating. The most widely used similarity functions are Pearson Correlation, Cosine, or Adjusted Cosine, but other similarity measures have also been proposed due to some drawbacks of these three measures (Ahn, 2008; Liu et al., 2014)

The normalization of the rating is another aspect which has been studied about the Neighborhood-Based methods (Herlocker et al., 2002), as users might rate items in different scale: some users might rate all items with a low score while others might have an higher average rating. For this reason, the rating in the prediction equation is often replaced by the mean-centered rating (Breese et al., 1998), which is equal to the rating minus the mean rating for a user (or item depending on the neighborhood method used). Another variant of the prediction function is using Z-score by further dividing the mean-centered rating by the standard deviation of the user's (or item's) observed ratings, for User-based (or Item-based) approaches (Herlocker et al., 2002)

Finally, if K-nearest-neighbors (K-NN) is the standard approach to find neighbors of the user or item, it is possible to add a filtering method to exclude from the neighbors those who are too weakly correlated to the target user or item. Other methods to find the neighborhood have also been investigated: for example, graph-based methods which aim at determining neighbors from user-item or user-user graphs have been discussed in (Aggarwal et al., 1999; Fouss et al., 2007).

Among the User-based and Item-based methods, the Item-based would often provide more relevant recommendations since the user's own ratings are used to predict the score of other items, rather than using other user's preferences. Indeed, other users might have overlapping interests but also different ones from the target user. On the other hand, in some cases, this aspect of User-Based methods can also bring some diversity by recommending items outside of the interests of the user.

Neighborhood-Based Recommendation has the advantage to be easily tuned as few parameters are used in this technique. They are also intuitive for the developer of the RS to implement, and a simple explanation can be given to the user at the time he receives the recommendation (such as showing statements like "Users who liked this item also liked this" or "We recommended these movies because you watched and liked [title of a movie]").

However, the Neighbor-Based approaches show some drawbacks: first, enough ratings need to be collected, otherwise the step of finding neighbors would be unsuccessful, and the coverage of rating prediction might be very small. Then, as the number of users or items grows to become very large, computing similarity becomes impractical. For these reasons, the second sort of CF approaches are often preferable in RS applications.

2.3 Collaborative Filtering: Matrix Factorization Techniques

Apart from the Memory-Based methods which find neighbors based on the previous ratings, another sort of Collaborative Filtering emerged, known as Model-Based approaches. Instead of using all previous ratings to make a prediction, they first build a model from theses ratings, and use this model to make further predictions and recommendations.

Various approaches can be used to build the model, such as clustering (Ungar and Foster, 1998; George and Merugu, 2005), Bayesian methods (Miyahara and Pazzani, 2000), or Neural Networks (Salakhutdinov et al., 2007). However, this section will only give on overview on **Matrix Factorization (MF)** methods for recommendation, which are part of Latent Factor Models. A more detailed presentation will be made compared to previous sections, as they are the most popular techniques, and our methods described in further chapters are based on MF.

2.3.1 Matrix Factorization and Low-Rank Approximation

The main idea behind MF is that overall preferences of each user can be decomposed into a set of features that represent and weight the interaction between the tastes of the user and the items. They are part of Collaborative Filtering approaches as they suppose that a large portion of rows and columns of the rating matrix are highly correlated.

The central hypothesis of Matrix Factorization is that a true rating $r_{u,i}$ made by a user u on an item v is the product between a user feature vector $\mathbf{U}^*{}_u$ and an item feature vector $\mathbf{V}^{*T}_i$. As a consequence, the data in the matrix can be well-estimated using a **low-rank approximation**, by projecting the users and items features into a low dimension space.

The low-rank approximation provides a lower dimensional representation of the original high-dimensional rating matrix R of size $N \times M$. If we consider a low-rank approximation of size k, the matrix $\mathbf{U}$ as the users latent features, of size $N \times k$ and the matrix $\mathbf{V}$ as the items latent features of size $M \times k$, then the rating of a user u on an item i is estimated as

$$\hat{r}_{u,i} = \mathbf{U}_u \mathbf{V}_i^T \,, \tag{2.2}$$

where $\mathbf{U}_u$ and $\mathbf{V}_i$ correspond respectively to the line of user u in $\mathbf{U}$ and the line of item i in $\mathbf{V}$. Here, each row of U corresponds to a user and represents the factors influencing this user's choice, while each row of $\mathbf{V}$ maps an item into the same k-dimensional space.

The Figure 2.3 represents how Matrix Factorization works. The blue-filled rectangles represent an estimated rating in the matrix and its corresponding feature vectors for the user u and item i in $\mathbf{U}$ and $\mathbf{V}$.

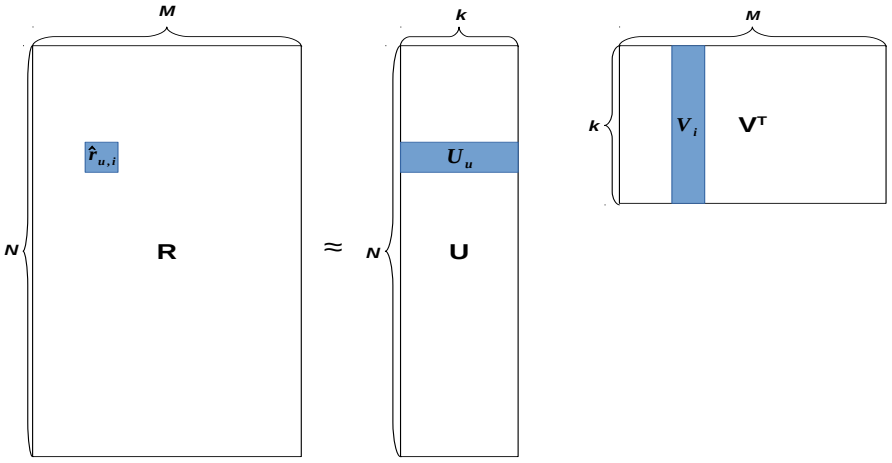


FIGURE 2.3: Representation of Matrix Factorization.

Let us take a concrete example to understand this concept: if each feature of a row in $\mathbf{V}$ represents a movie genre, then the values in a row of $\mathbf{V}$ would represent for a movie its affinity to each genre. Similarly, each value of $\mathbf{U}$ would give an affinity score representing how much a user likes a genre. Then, the rating of the user on this movie is estimated as the sum of all products between the affinity of the user to a given genre and the affinity of the item to this same genre. This example is given to understand the concept of latent user and item features, but in practice, it is often not possible to interpret the semantic of each latent factor.

Compared to Neighbor-based methods, MF techniques have a number of advantages:

- Speed: the model learned with MF is learned very quickly compared to neighborhood-based models, where the similarity computation is quadratic in number of users or items. Furthermore, the representation learned through MF is compact since the dimension is reduced, and the prediction can be made quickly.
- Space: the low-rank approximation allows to gain space to store the model.
- More generalization power: by using regularization during the learning phase of the model, Model-based approaches can reduce overfitting of the data.

In the Section 2.3.2, we describe the main models of Matrix Factorization, which mostly differ by the nature of the objective function, or by the constraint imposed on $\mathbf{U}$ and $\mathbf{V}$.

2.3.2 Matrix Factorization Methods

Several approaches have been used to obtain a low-rank representation of the matrix of ratings. We present here a few of them which are among the most popular to perform MF.

Objective

In Matrix Factorization, the users and items feature matrices $\mathbf{U}$ and $\mathbf{V}$ have to be found by minimizing the squared error over known ratings in the set $\mathcal{S}$:

$$(\mathbf{U}, \mathbf{V}) = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmin}} \sum_{\forall (u,i) \in \mathcal{S}} \left(r_{u,i} - \mathbf{U}_u \mathbf{V}_i^T \right)^2 . \tag{2.3}$$

However, since a large part of the ratings in the matrix are unknown, usual models built for RS are adding a *regularization* term to solve the following equation, *e.g.*:

$$(\mathbf{U}, \mathbf{V}) = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmin}} \sum_{\forall (u,i) \in \mathcal{S}} \left(r_{u,i} - \mathbf{U}_u \mathbf{V}_i^T \right)^2 + \lambda (||\mathbf{U}||_F^2 + ||\mathbf{V}||_F^2) , \tag{2.4}$$

where $\| \cdot \|_F^2$ denotes the squared Frobenius norm, and λ is a regularization parameter.

When trying to solve the minimization problem in Equation (2.4), the second term defining the regularization has received a particular attention in research (Srebro et al., 2004). Since only few entries of the ratings matrix are available for the model to learn, the model can easily overfit or be influenced by extreme values. The goal of the regularization is to reduce chances of large values of features for values in $\mathbf{U}$ and $\mathbf{V}$. Thus, choosing an adapted regularization is a crucial aspect in MF methods.

In the following, we describe the three main approaches which adopt the perspective of solving this minimization problem.

Optimization Methods

We present in this section three approaches that received a lot of attention in the field of RS to solve the minimization problem: Singular Value Decomposition, Stochastic Gradient Descent and Alternating Least Squares.

SVD The Singular Value Decomposition (SVD) of a full matrix $\mathbf{R}$ with N users and M items is defined as

$$\mathbf{R} = \mathbf{P}\Sigma\mathbf{Q}^T \,, \tag{2.5}$$

where $\mathbf{P}$ is a $N \times N$ orthogonal matrix, $\mathbf{Q}$ is a $M \times M$ orthogonal matrix, and Σ is a $N \times M$ rectangular diagonal matrix with non-increasing non-negative singular values on the diagonal.

The properties of SVD assure that solving the Equation (2.3) for the matrix $\mathbf{R}$ with SVD gives the best low-rank linear approximation (from the Eckart-Young theorem (Eckart and Young, 1936)): the approximation found is thus optimal in Frobenius norm. From the SVD decomposition, the matrix $\mathbf{R}$ can be approximated by choosing the first k singular values of Σ and reducing the matrices $\mathbf{P}$ and $\mathbf{Q}$ to their first k columns.

The application of standard SVD in the case of RS raises the issue of sparsity. The SVD method requires a full matrix to be used, while the matrix of ratings contains many (user-item) couples for which no preference has been specified. For this reason, some solutions have been applied in the RS literature, such as filling the entries in the matrix with some default values, for example zero or the average rating of each item or each user (Kurucz et al., 2007). The imputation in practice can be very expensive as it significantly increases the amount of data, and that inaccurate filling of the matrix might distort the data.

Some works done at the end of the 2000s have shown that it is more efficient to find the decomposition by only using observed ratings (Paterek, 2007; Takács et al., 2007; Koren, 2008), which is the focus of SGD and ALS methods described thereafter. Since the problem is solved only using observed ratings, regularization is always used, so we consider all approaches in the following to target the minimization problem described in Equation (2.4).

SGD Stochastic Gradient Descent (SGD) first initializes user and item latent features $\mathbf{U}$ and $\mathbf{V}$ randomly. Then, for a given number of training rounds or until convergence, the algorithm loops through all observed entries in the matrix. The prediction error is calculated for each entry:

$$e_{u,i} = r_{u,i} - \mathbf{U}_u\mathbf{V}_i^T \,, \tag{2.6}$$

and then, the latent features are updated with the following update rules:

$$\mathbf{U}_u \leftarrow \mathbf{U}_u + \alpha \cdot (e_{u,i} \cdot \mathbf{V}_i - \lambda \cdot \mathbf{U}_u) \,, \tag{2.7}$$

$$\mathbf{V}_i \leftarrow \mathbf{V}_i + \alpha \cdot (e_{u,i} \cdot \mathbf{U}_u - \lambda \cdot \mathbf{V}_i)\,, \tag{2.8}$$

where α is the learning rate of the gradient descent, and λ is a regularization parameter to avoid overfitting. The algorithm of SGD is described in Algorithm 2. The stopping criterion is satisfied after a given number of iterations, but the algorithm can also be stopped prematurely if the overall error (between the predicted ratings and the actual ratings) goes below a given level.

Algorithm 2: Stochastic Gradient Descent.

Input : Matrix of observed ratings: $\mathbf{R}$,
Set of known ratings: $\mathcal{S}$,
regularization parameter: λ,
Output: User features matrix: $\mathbf{U}$
Item features matrix: $\mathbf{V}$

Initialize $\mathbf{U}$ and $\mathbf{V}$ (with randomized values for example) ;
while *Stopping criterion not met* **do**
  for $(u, i) \in \mathcal{S}$ **do**
    $e_{u,i} = r_{u,i} - \mathbf{U}_u \mathbf{V}_i^T$;
    $\mathbf{U}_u \leftarrow \mathbf{U}_u + \alpha \cdot (e_{u,i} \cdot \mathbf{V}_i - \lambda \cdot \mathbf{U}_u)$;
    $\mathbf{V}_i \leftarrow \mathbf{V}_i + \alpha \cdot (e_{u,i} \cdot \mathbf{U}_u - \lambda \cdot \mathbf{V}_i)$;
  end
end

The complexity of SGD per iteration is $O(|\mathcal{S}|\, k)$, with k being the size of the approximation. The SGD method is efficient but is sensitive to the parameter chosen for the learning rate, and also to the initialization of $\mathbf{U}$ and $\mathbf{V}$. Some work has been done to select an adaptive α through each iteration to avoid local minima and make the convergence faster (Gemulla et al., 2011). Finally, SGD can be parallelized, and a variety of ways have been proposed such as (Gemulla et al., 2011; Recht et al., 2011). However, the parallelization remains challenging, especially compared to the one of ALS presented below.

ALS Alternating Least Squares (ALS) (Jain et al., 2013) also aim at solving the Equation (2.4). Because both $\mathbf{U}$ and $\mathbf{V}$ are unknown, the objective function to minimize in Equation (2.4) is non-convex. The idea behind ALS is to solve the problem in an iterative manner, by repeatedly executing the two following steps:

1. Holding the item latent matrix $\mathbf{V}$ fixed and solving the quadratic equation for the user matrix $\mathbf{U}$.
2. Holding the user latent matrix $\mathbf{U}$ fixed and solving the least-square problem for the item matrix $\mathbf{V}$.

When one of these two is taken as a constant, the optimization problem is indeed quadratic and can be optimally solved. The algorithm loops between these two steps until convergence or for a given number of iterations.

This approach is considered more stable than SGD, since ALS does not need a learning rate parameter and usually requires less iterations to obtain a good enough model. The reason behind that is that at each step of ALS, the exact minimum is found: for example, repeating step 1 two times in a row will have no effect on $\mathbf{U}$ as the minimum with the item latent matrix $\mathbf{V}$ fixed has already been found. It means that generally, a single iteration of ALS will move much further than a single iteration of SGD, and fewer iteration will be needed for convergence.

When $\mathbf{V}$ is considered fixed, then $\mathbf{U}$ is determined by solving a regularized linear least squares and the optimal value for $\mathbf{V}$ is given by

$$\mathbf{U} = (\mathbf{V}\mathbf{V}^T + \lambda\mathbf{I})^{-1}\mathbf{V}\mathbf{R}^T\,, \tag{2.9}$$

where $\mathbf{I}$ is the identity matrix. Similarly, the solution for $\mathbf{V}$ when $\mathbf{U}$ is fixed is given by

$$\mathbf{V} = (\mathbf{U}\mathbf{U}^T + \lambda\mathbf{I})^{-1}\mathbf{U}\mathbf{R}\,. \tag{2.10}$$

The analytical solution can be computed since it involves inverting a small $k\times k$ matrix and multiplication of sparse matrices. With $\mathcal{I}(u)$ denoting the set of items rated by user u and $\mathcal{J}(i)$ denoting the set of users who rated item i, the ALS is described in detail in Algorithm 3.

Algorithm 3: Alternating Least Squares.

Input : Matrix of observed ratings: $\mathbf{R}$,
Set of known ratings: $\mathcal{S}$,
regularization parameter: λ,
Input/Output: User features matrix: $\mathbf{U}$
Item features matrix: $\mathbf{V}$

Initialize $\mathbf{U}$ and $\mathbf{V}$ (with randomized values for example);
while *Stopping criterion not met* **do**
  for $u \in 1, \dots N$ **do**
    $\mathbf{U}_u \leftarrow (\sum_{i\in\mathcal{I}(u)} \mathbf{V}_i\mathbf{V}^T{}_i + \lambda\mathbf{I})^{-1} \sum_{i\in\mathcal{I}(u)} \mathbf{V}_i\mathbf{R}^T_{u,i}$;
  end
  for $i \in 1, \dots M$ **do**
    $\mathbf{V}_i \leftarrow (\sum_{u\in\mathcal{J}(i)} \mathbf{U}_u\mathbf{U}^T{}_u + \lambda\mathbf{I})^{-1} \sum_{u\in\mathcal{J}(i)} \mathbf{U}_u\mathbf{R}_{u,i}$;
  end
end

The stopping criterion can either be triggered when a given number of iterations is reached, or when a satisfying score on an evaluation measure (like RMSE) is reached on a probe dataset. Remark that it is almost always faster to solve the linear system $\sum_{i\in\mathcal{I}(u)} \mathbf{V}_i\mathbf{V}^T{}_i + \lambda\mathbf{I} = \sum_{i\in\mathcal{I}(u)} \mathbf{V}_i\mathbf{R}^T_{u,i}$ instead of inverting the matrix $(\sum_{i\in\mathcal{I}(u)} \mathbf{V}_i\mathbf{V}^T{}_i + \lambda\mathbf{I})$ as written in the algorithm (and similarly for the inversion in the update of $\mathbf{V}$).

Updating each user vector $\mathbf{U}_u$ costs $O(|\mathcal{I}(u)|k^2 + k^3)$ and updating each item vector $\mathbf{V}_i$ costs $O(|\mathcal{J}(i)|k^2 + k^3)$, which gives a total complexity for one iteration of ALS of $O(|\mathcal{S}|k^2 + (N+M)k^3)$. Although this is a high time complexity, ALS is well-suited for parallelization as shown in (Zhou et al., 2008).

There are several well-known extensions of ALS: one is called Alternating Least Squares with Weighted-Lambda-Regularization (ALS-WR), described in (Zhou et al., 2008), and which uses a specific regularization. The ALS-WR algorithm is given in detail in Chapter 3 as we use this variant of ALS in our approach. Another application of ALS with a weighted version suits particularly well in the setting where the feedback is implicit, with many zero values (Hu et al., 2008).

Incorporating Bias

The MF minimization problem as described in Equation (2.4) has many variations to make more complex models and add some terms to capture a specific effect. A popular variation of the MF is to add some term capturing the bias embedded in the ratings.

Suppose a RS where the rating scale ranges from 1 to 5 stars. Let us consider two users u_1 and u_2 whose respective average ratings on items are 1.5 and 4.5. Then, if u_1 gives a rating of 3 to an item, the meaning of this rating is really different than if the user u_2 was rating the same item with the same rating of 3. More generally, explicit feedback data is highly biased: some users rate more extremely than others, some items tend to get higher ratings than others... Regularization is then not enough to make good generalization and a solution is to explicitly model biases into the model (Koren et al., 2009).

A standard way to incorporate the biases is to incorporate the rating bias into the prediction. A first-order approximation of the bias b_{u_i} for a user u and an item i is defined as

$$b_{u_i} = \mu + b_i + b_u \,, \tag{2.11}$$

where μ represents the overall average rating in the whole dataset, b_u represent the observed deviation of user u and b_i is the observed deviation of the item i. For example, if the overall average rating in a dataset is 3.2 stars, if a user u rates in average 0.5 stars higher than other users, and if an item i receives in average 1.5 stars less than other items, then the estimate for the bias would be $b_{u_i} = 3.2 + 0.5 - 1.5$. The system then minimizes the following squared error function:

$$(\mathbf{U}, \mathbf{V}) = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmin}} \sum_{\forall (u,i) \in \mathcal{S}} \left(r_{u,i} - \mu - b_i - b_u - \mathbf{U}_u \mathbf{V}_i^T \right)^2 + \lambda (||\mathbf{U}||_F^2 + ||\mathbf{V}||_F^2) \,. \tag{2.12}$$

Apart from the bias on ratings, other aspects can also be added into the function to minimize: some examples are (Koren, 2010) which is tackling the temporal dynamics, or (Hu et al., 2008) which targets a problem with a dataset containing implicit feedback.

Other Models

Other particular models have been proposed to manage MF, which are specific due to their perspective to approach the problem, like PMF which is designed from a probabilistic view, or NMF which add constraints to the optimization problem. Another way to represent the problem is also to use Factorization Machine, which is a generalization of previous models and can possibly integrate extra information (cf. Section 2.5).

PMF Another possible approach for Matrix Factorization is the probabilistic one, such as the Probabilistic Matrix Factorization (PMF) (Salakhutdinov and Mnih, 2008b). In this case, Gaussian priors are put on user and item feature vectors, with respective hyperparameters σ_U^2 and σ_V^2:

$$\mathbf{U} \sim \mathcal{N}(0, \sigma_U^2 \mathbf{I}) \,, \tag{2.13}$$

$$\mathbf{V} \sim \mathcal{N}(0, \sigma_V^2 \mathbf{I}) \,, \tag{2.14}$$

and rating are generated according to a Gaussian with hyperparameter σ^2 as:

$$r_{i,j} | \mathbf{U}, \mathbf{V} \sim \mathcal{N}(\mathbf{U}_i^T \mathbf{V}_j, \sigma^2) \,. \tag{2.15}$$

If the matrix $\mathbf{R}$ is of size $N \times M$, k is the dimension of the latent features, and $I_{u,i}$ is defined as 1 if $r_{u,i}$ is a known rating and 0 otherwise. The log of the posterior distribution over user and movie features is given by:

$$\begin{aligned} \ln p(\mathbf{U}, \mathbf{V} | \mathbf{R}, \sigma^2, \sigma_V^2, \sigma_U^2) = &- \frac{1}{2\sigma^2} \sum_{u=1}^{N} \sum_{i=1}^{M} I_{u,i} (r_{u,i} - \mathbf{U}_u \mathbf{V}_i^T)^2 \\ &- \frac{1}{2\sigma_U^2} \sum_{u=1}^{N} \mathbf{U}_u^T \mathbf{U}_u - \frac{1}{2\sigma_V^2} \sum_{i=1}^{M} \mathbf{V}_i^T \mathbf{V}_i \\ &- \frac{1}{2} \left(\kappa \ln \sigma^2 + Nk \ln \sigma_U^2 + Mk \ln \sigma_V^2 \right) + C \,, \end{aligned} \tag{2.16}$$

where κ is the number of known entries, C is a constant independent of the parameters, . Maximizing the log-posterior over movie and user features is equivalent to minimizing the objective function:

$$E = \frac{1}{2} \left(\sum_{\forall (u,i) \in \mathcal{S}} I_{u,i} (r_{u,i} - \mathbf{U}_u \mathbf{V}_i^T)^2 + \lambda_U \sum_{u=1}^{N} \|\mathbf{U}_u\|_F^2 + \lambda_V \sum_{i=1}^{M} \|\mathbf{V}_i\|_F^2 \right) \,, \tag{2.17}$$

where $\lambda_U = \sigma^2 / \sigma_U^2$ and $\lambda_V = \sigma^2 / \sigma_V^2$. A gradient descent in $\mathbf{U}$ and $\mathbf{V}$ can then be used to find a local minimum of this objective function.

While we recover an objective function similar to the one in Equation (2.4) (the only difference is that there are specific regularizations λ_U and λ_V for the user and item features respectively, instead of a single regularization parameter), the probabilistic point of view allows for extensions of this formulation such as a Bayesian version BPMF (Salakhutdinov and Mnih, 2008a), or the use of side information (Ma et al., 2011).

NMF Non-negative Matrix Factorization (NMF) adds a constraint about **U** and **V** on the optimization formulation, where both matrices have to be non-negative (*i.e.* have no negative elements). NMF approaches are mostly used in RS in the case where the feedback stored in the rating matrix is an implicit feedback with unary ratings (for example, we only know the user listened to a song, or bought an item, which are represented by a rating of 1 in the matrix, but other preferences are all unknown).

NMF can be solved iteratively by using the following rules to update the matrices **U** and **V** (Lee and Seung, 2001):

$$u_{i,j} \leftarrow u_{i,j} \frac{(RV)_{i,j}}{(UV^T V)_{i,j}} \forall i \in 1 \dots N, \forall j \in 1 \dots k\,, \tag{2.18}$$

$$v_{i,j} \leftarrow v_{i,j} \frac{(R^T U)_{i,j}}{(VU^T U)_{i,j}} \forall i \in 1 \dots M, \forall j \in 1 \dots k\,. \tag{2.19}$$

In order to prevent divisions by 0, small values are sometimes added to the denominator of both update equations. The entries of **U** and **V** are initialized to random values in [0,1] and the update rules are executed until the convergence of both matrices.

As in other types of MF, a regularization can be used to improve the quality of the solution (Pauca et al., 2006). Finally, NMF can also be performed through ALS instead of using Gradient Descent, such as in (Kim and Park, 2008).

NMF methods do not show any particular improvement in accuracy compared to other methods, but they have the advantage to be more interpretable than standard MF models, as all features of **U** and **V** are non-negative. For specific applications of Non-Negative Matrix Factorization in the field of RS, see (Hofmann, 2004; Zhang et al., 2006).

Factorization Machine Factorization Machines (FM) (Rendle, 2012) are a generalization of latent factor approaches, and approaches such as SVD can be viewed as special cases of this approach. They have shown a lot of success in online competitions about Recommendation Systems or Machine Learning in general. The idea behind FM is to model the rating as a linear combination of interactions between input variables: as such the FM can be compared to Support Vector Machines (SVM) with a polynomial kernel, which allows the FM to perform well on sparse data.

For a set of N users and M items, the input for FM can be considered as a flatten version of users and items of dimension $P = N + M$, where each training row is a concatenation between a binary representation of the active user and a binary representation for the active item. The output target to learn is then the rating for this user on this item. Every training example represents a given rating for one user and one item. The power of FM is that additional information such as external information about the user, the item, or context information can be added really easily by simply concatenating it to the training example.

The Figure 2.4 gives an example of training data for FM, with some additional information. Refer to Section 2.5 for more information about integrating contextual information in models.

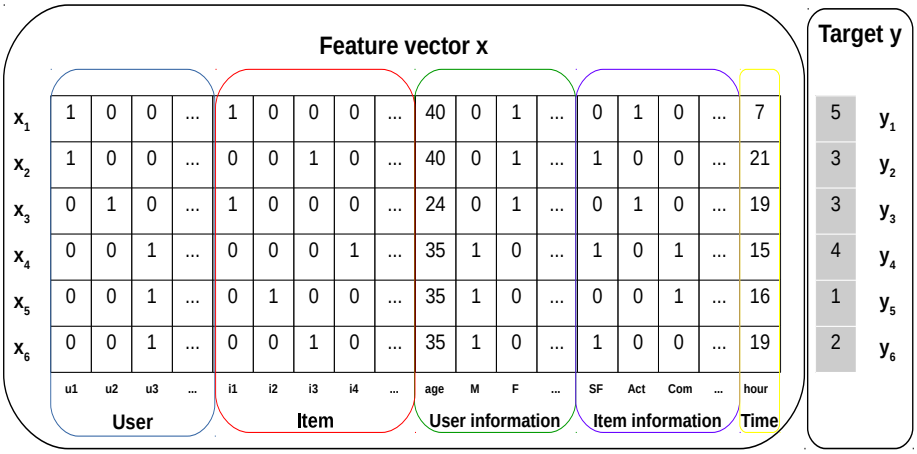


FIGURE 2.4: Example of Factorization Machine input (user and item binary representation, and external or contextual information) and output rating.

Factorization Machine have strong expressiveness power and efficiency, as they can take in input an arbitrary feature space, and make the rating prediction by using pairwise interactions between the factors. The model equation for FM of degree 2, where only the second order interaction are learned, is defined as

$$\hat{y}(x) = w_0 + \sum_{i=1}^{P} w_i x_i + \sum_{i=1}^{P} \sum_{j=i+1}^{P} \langle v_i, v_j \rangle x_i x_j \,, \tag{2.20}$$

where the model parameters are w_0 representing the global bias, w_i modeling the strength of the i-th variable, and $\hat{w}_{i,j} = \langle v_i, v_j \rangle$ which models the interaction between the i-th and the j-th variable. Even though the number of interaction terms seems large, most of them would be evaluated to zero in the sparse settings. This is why FM are sometimes considered not well adapted to dense setting, and mostly suit a sparse input.

Through feature engineering, the FM can mimic some of the best specialized MF models such as SVD++. Several methods of optimization can be used to solve the optimization problem, and Rendle introduced three of them in (Rendle, 2012): the Stochastic Gradient Descent, the Alternating Least-Squares, and the Markov Chain Monte Carlo inference, which are all implemented in the library libFM to use FM models [1].

[1] http://www.libfm.org/

2.3.3 Limitations

MF-based methods have been among the most popular models for RS since the Netflix prize (Bennett et al., 2007). However, these do not come without limitations, which we list here briefly.

1. Cold-start: The cold-start issue happens when new items or new users enter the system. The CF recommendation engine cannot provide suggestions to unknown users, or about unknown items in such cases because they use only the past feedback provided by users or about the items (a CF RS would have no past feedback for a user or an item which are unknown until now). The Section 2.4 gives more details about the cold-start issue, and introduce some of the solutions found to reduce it.

2. Item trend and obsolescence: In dynamical systems, items are possibly inserted and deleted quickly, or they see their trend grow or decrease quickly through time. A good example for this situation are RS targeting at news. The RS model needs to update and adapts very quickly, and the maintenance of the model via incremental update or recalculation is required. The use of the contextual information (time for example) as depicted in Section 2.5 can sometimes help to integrate such effects into the model.

3. Metrics to optimize: The CF methods presented in the previous section target at predicting unknown ratings, and aim at optimizing the quadratic error between the predicted rating and the real rating. Yet, in real recommendation, the user's satisfaction only depends on the RS being able to detect the top preferences of the user, as he bears interest only in the top items which are recommended to him. For this reason, in recent years, the interest of the RS community has shifted from trying to predict accurately ratings to being able to rank items correctly, and Learning to Rank methods have started to emerge. A good example of this shift is the Challenge organized by the Recsys conference in 2014, where the evaluation focused on optimizing a rank measure instead of a prediction metric, and where our approach mostly based on Learning To Rank algorithm obtained the best results (Guillou et al., 2014). Refer to Section 2.6.1 for a description of prediction and ranking metrics, and go to Section 2.7 for an overview of Learning to Rank approaches.

4. Security: In collaborative RS, some users may try to influence the system by creating fake profiles and affect the system by giving either high ratings to some items, making them more likely to be recommended, or by lowering the ratings of competitor's items, making them less likely to be suggested by the RS. The protection of the RS against attacks to build trustworthy systems is addressed for example in (Mobasher et al., 2007)

In the next sections, we focus on these limitations and solutions that have been proposed to solve them.

2.4 Cold Start Issue

The cold start issue appears due to the lack of information for the system about a user or an item. These two situations are often referred as *user cold-start* and *item cold-start* respectively. It is usually the most challenging situation for CF Matrix Factorization approaches, since these methods rely purely on the collected ratings about user and items. As a consequence, two fields of research have looked into this issue and propose some solution. The first one created some RS called Hybrid RS, by combining CF methods with Content-Based recommendation to reduce the cold-start by using external information. The second one is the field of Active Learning, which considers the problem from a different point of view and aims at learning the profile of the user as quickly as possible by requesting the user to rate at the beginning some items which would bring as much information as possible to learn the user features.

2.4.1 Hybrid Recommendation Systems

As CF approaches work with the preferences of users, and Content-Based methods rely on users or items information (or Demographic methods using user's demographic information), they all use different sources of input to make prediction, and have their own advantages or weaknesses. Hybrid RS (Burke, 2002) make the assumption that various sources of input are available at the same time, which allow to make use of different recommendation approaches inside one recommendation framework, and combine them to minimize their disadvantages.

The combination of models can either be designed as an ensemble (multiple predictions from different RS are combined into a single prediction) like in (Jahrer et al., 2010; Wu, 2007), or can integrate directly the different methods into one more complex RS structure to make the prediction (Claypool et al., 1999). Finally, in the situation where a list of items is recommended, the RS can simply make use of each model separately and combine their recommendation to form the list. A good example of this latter approach is the Amazon website, where different parts of the screen indicate a recommendation performed by one kind of algorithm (for example: "Similar products" based on Content-Based methods and "The client who bought this item also bought ..." based on Neighborhood CF approach).

2.4.2 Active Learning

Before being used in Recommendation Systems, Active Learning is originally a technique used in semi-supervised machine learning, where a learning algorithm has a limited budget about the number of training points to learn a model, and has to query a user about a limited number of training examples. These data points have to be chosen in order to maximize the prediction accuracy of the algorithm.

The problem is very similar in RS in the cold-start situation for a user: the system has to query the user to model his preferences, but cannot require him to give a rating to many items, as the process of acquiring rating is time-consuming and users are not willing to provide some ratings without benefit. Active learning approaches focus on finding which items to recommend at first to gather as much information as possible about the user's preferences. For more details about Active Learning, the reader can refer to (Rubens et al., 2011).

2.5 Incorporate Contextual Information

In a similar fashion to Hybrid RS, some RS have attempted to use as much as possible information available to them by integrating contextual information into their model (Adomavicius and Tuzhilin, 2011). The main examples of contextual information are:

- Social information: in the case of a movie RS, a user might want different recommendations depending on when he is with his friends or when he is with his family.
- Temporal information: it is a well-known phenomena that some sales depend on the season, especially in the clothing industry. Depending on the time of the day, on the day of the week, or on the season, users do not always have the same preferences concerning their recommendation, and including such information can help to greatly improve the recommendation accuracy. (Campos et al., 2014) (Koren, 2010)
- Location information: more and more users are installing mobile applications to listen to music, search for a restaurant, or buy items online. The location information is thus tightly related to the rise of recommendation on mobile platforms. Through the GPS, the location of user is available and can be used by the RS (Levandoski et al., 2012; Yang et al., 2008). For instance, some users might want to listen to a specific kind of music when resting at home, and a different one when being at the office.

Compared to usual MF approaches, the use of the context adds a dimension to the input space, and the rating data does not consists in a rating matrix anymore but is rather a 3-dimensional cuboid corresponding to user, item and context. If several contexts are used at the same time, then the space of the representation for ratings would become multidimensional (Adomavicius et al., 2005). There are several ways to solve the recommendation problem with this sort of input. The first one is the use of Factorization Machine presented in Section 2.3 as they can easily integrate context (Rendle et al., 2011). Other approaches often make use of tensor factorization, with the largest part of the research focusing on the integration of temporal dynamic. (Karatzoglou et al., 2010; Xiong et al., 2010).

2.6 Evaluation of Recommendation Systems

To evaluate the performance of RS, some criteria or metrics are necessary, as well as a proper evaluation methodology (Herlocker et al., 2004; Shani and Gunawardana, 2011). This is a crucial step in the recommendation process, as a misleading evaluation of the RS could lead to the loss of customers and profit for a company. In this section, we first go over the range of evaluation metrics, and then describe the evaluation methodologies and scenarios traditionally set up.

2.6.1 Metrics

A broad range of measures have been used throughout the research on RS. All these measures focus on a different utility function and goal. It is possible, and even recommended to evaluate a RS on different measures to test its robustness on different criteria.

Prediction Accuracy

As described before, the CF approaches have historically aimed at predicting ratings through Matrix Completion methods. As a consequence, prediction accuracy metrics have been massively used in the field for a long period to perform the evaluation of RS.

Given a set of hidden ratings in a test set $\mathcal{T}$, the RS is evaluated through its prediction for these hidden ratings, by comparing how close they are from the real ones. We denote the predicted rating on the item i by the user u as $\hat{r}_{u,i}$, and $r_{u,i}$ is the real rating. The most widely adopted measures are:

- The **Mean Absolute Error (MAE)** measures the average absolute deviation between a real and a predicted rating. It is easy to compute, but does not penalize large errors:

$$MAE = \frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} |r_{u,i} - \hat{r}_{u,i}| \,. \tag{2.21}$$

- The **Mean Squared Error (MSE)**. Compared to MAE, MSE emphasizes large errors.

$$MSE = \frac{1}{|\mathcal{T}|} \sum_{(u,i)\in|\mathcal{T}|} (r_{u,i} - \hat{r}_{u,i})^2 \,. \tag{2.22}$$

- The **Root Mean Squared Error (RMSE)** is the square-root of the MSE value, and it is often employed instead. It was the standard metric adopted for the Netflix Prize (Bennett et al., 2007), and has been used in a large number of Collaborative Filtering papers.

$$RMSE = \sqrt{MSE} \,. \tag{2.23}$$

For all these measures, the lower the score, the better the predictive power of the system is. However, such metrics put a strong emphasis on popular items, and items with few ratings would not have much influence on the final evaluation score. For example, a system in which almost all ratings are around 3 in a 1 to 5 stars scale would get a good evaluation score by predicting a 3 for every item, while it would be more important from the point of view of the user to put more weight on high ratings to be able to correctly predict them. For a more detailed review on the advantages or disadvantages of MAE and RMSE, see (Willmott and Matsuura, 2005; Chai and Draxler, 2014).

These prediction accuracy measures have been targeted for the evaluation in a high proportion of RS papers in the 2000s, but some researches have pointed the fact that focusing on accuracy can hurt the user's experience (McNee et al., 2006), and evaluation metrics have slowly shifted to other measures.

Information Retrieval measures

If prediction accuracy metrics measure how well the system can estimate ratings, they do not focus on the user's satisfaction (*i.e.* does the user receive recommendation on which he will give a high rating?). Information Retrieval measures evaluate on the other hand more precisely how the RS is capable of making relevant recommendations, by comparing the list of recommended items by the RS with the ground-truth of the user's preferences.

The relevancy of an item can be determined in different ways: if the feedback considered in the dataset is an implicit one, and represents the click on items, an item can be considered relevant if the user clicks on it. Otherwise, if the feedback is an explicit one represented by a rating in a numerical scale, a threshold is usually defined to judge whether an item is relevant or not (for instance, a rating strictly higher than 3 for a 1 to 5 stars rating scale).

Suppose the algorithm recommends a list of s items to a user, and the set of recommendation is denoted as $\mathcal{L}(s)$ (the size of the recommendation list can change). Let $\mathcal{R}$ denote the set of relevant items for this user among all items. Then, two metrics can be defined:

- the **Precision** measures the fraction of relevant items recommended in the list, as
$$Precision(l) = \frac{|\mathcal{L}(s) \cap \mathcal{R}|}{|\mathcal{L}(s)|}, \tag{2.24}$$
- the **Recall** measures which fraction of the relevant items have been recovered in the set of recommendation:
$$Recall(l) = \frac{|\mathcal{L}(s) \cap \mathcal{R}|}{|\mathcal{R}|}. \tag{2.25}$$

Let us consider a user u with a set of relevant items consisting in 5 items: $\mathcal{R}(u) = \{i_1, i_3, i_4, i_6, i_8\}$. If the list of size is $s = 3$, and $\mathcal{L}(3) = \{i_1, i_2, i_3\}$ is recommended to this user, then the Precision score is

$$\frac{|\{i_1, i_2, i_3\} \cap \{i_1, i_3, i_4, i_6, i_8\}|}{|\{i_1, i_2, i_3\}|} = \frac{2}{3},$$

and the Recall score is

$$\frac{|\{i_1, i_2, i_3\} \cap \{i_1, i_3, i_4, i_6, i_8\}|}{|\{i_1, i_3, i_4, i_6, i_8\}|} = \frac{2}{5}.$$

Scores for precision and recall are averaged over all users to measure the performance of the system.

The scores of Precision and Recall can be conflicting, as increasing the size of recommendation can increase the recall (a longer recommendation list gives more chance to recommend more relevant items), but decreases the precision at the same time. A solution to evaluate the balance between the two is to calculate the **$\mathbf{F_1}$ score**:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{2.26}$$

Finally, if $\mathcal{A}$ represents the set of all available items for the RS, the **Receiver Operating Characteristic (ROC)** curve (Fawcett, 2006) characterizes the trade-off between the False Positive Rate $FPR = \frac{|\mathcal{L}(s) \setminus \mathcal{R}|}{|\mathcal{A} \setminus \mathcal{R}|}$ (which represents the ratio between non-relevant items which are recommended to the user and all the non-relevant items), and the True Positive Rate $TPR = \frac{|\mathcal{L}(s) \cap \mathcal{R}|}{|\mathcal{R}|}$ (which is the same as the Recall: it is the ratio between items recommended and all the possible relevant items). By gradually varying the length s of the list recommended to users, we can obtain several values of these two values, and get the ROC curve by plotting the FPR on the horizontal axis and the TPR on the vertical axis.

Rank Accuracy

In most real-world RS, the user actually receives a list of recommendation containing the top-k items predicted by the RS. In general, items must have a ranking in accordance with the user's preference, with highly-rated items ranked higher in the list. To measure how well a RS is able to conform to this need, ranking metrics have been used for evaluation. They allow to define how well a RS can generate a personalized ordering of items that matches the true user's ordering of preferred items.

For all ranking measures defined below, let us consider a user u, to which the list of items $i_1, \ldots, i_\ell$ of size ℓ has been recommended. $r^*_{u,i}$ is the true rating of user u with regard to item i. We assume the ratings range from 0 to r_{max}.

The first two ranking measures are rank correlation coefficients, and are named the **Kendall coefficient** and the **Spearman coefficient** (Kendall, 1948).

To calculate the Kendall coefficient, we define first the coefficient $C(i,j)$ which is equal to +1 if items i and j are in the same relative order in the ranked list and in the ranking predicted by the RS (concordant pair), and equal to -1 if the items are in a different relative order (discordant pair). If items are tied, then the value of $C(i,j)$ is 0. The Kendall correlation for a user u is computed as the number of concordant pairs minus the number of discordant pairs, over all the pairs of items in the recommendation list:

$$\tau_u = \frac{\sum_{i<j} C(i,j)}{\ell(\ell-1)/2} \,. \tag{2.27}$$

The Spearman coefficient is equal to the Pearson correlation coefficient applied to the ranked variables (calculated between the ground truth ranking and the ranking predicted by the RS). The range of the computed value is from -1 to 1, where the best value is 1 when both ranking are equal.

Other measures have been defined, putting more emphasis on the top of the recommendation list, as the user pays much more attention to the top-ranked items. These measures are the **Normalized Discounted Cumulative Gain**, the **Mean Reciprocal Rank**, and the **Expected Reciprocal Rank**.

- The Discounted Cumulative Gain at ℓ (Burges et al., 2005) is first defined as

$$DCG@\ell = \sum_{r=1}^{\ell} \frac{2^{r^*_{u,i_r}} - 1}{\log_2(r+1)} \,. \tag{2.28}$$

  The Normalized Discounted Cumulative Gain (NDCG) at ℓ is

$$NDCG@\ell = \frac{DCG@\ell}{DCG^*@\ell} \,, \tag{2.29}$$

  where DCG^* is the best possible DCG (the DCG score obtained for the best possible ranking). For a theoretical analysis of the NDCG measure, the interested reader can refer to (Wang et al., 2013).

  Let us take an example where a list of items of size $l = 5$ is recommended to a user, and where relevance scores range from 1 to 5 stars. The user provides the following scores for the recommended items: $\{3, 5, 2, 3, 1\}$. The DCG score is calculated based on the Table 2.2.

TABLE 2.2: An example to calculate NDCG score.

r	r^*_{u,i_r}	$\log_2(r+1)$	$\frac{2^{r^*_{u,i_r}}-1}{\log_2(r+1)}$
1	3	1.0	7
2	5	1.585	19.56
3	2	2.0	1.5
4	3	2.322	3.01
5	1	2.585	0.39

The DCG is the sum of the last column of the Table:

$$DCG@5 = \sum_{r=1}^{5} \frac{2^{r^*_{u,i_r}} - 1}{\log_2(r+1)} = 31.46\,.$$

Let us now suppose that, for the same user, the higher possible relevance scores reached when a list of 5 items is recommended are the following scores: $\{5, 5, 4, 4, 3\}$. Using a similar table to Table 2.2, the best possible DCG would be $DCG^*@5 = 67.23$. The NDCG obtained for this user is then

$$NDCG@5 = \frac{DCG@5}{DCG^*@5} = \frac{31.46}{67.23} = 0.468\,.$$

- The notion of "reciprocal rank" in the Mean Reciprocal rank (MRR) is defined usually for binary relevancy (the rating on items is either 0 or 1), but can also be defined if the relevancy of an item is defined for a numerical scale (for example, in a 1 to 5 rating scale, any item with a score higher than 3 is considered to be relevant). The MRR score consists in the reciprocal of the rank at which the first relevant item is detected. The reciprocal rank is 1 if the first item is relevant in the list of recommended items. For a number N of users, it is defined as:

$$MRR = \frac{1}{N} \sum_{u=1}^{N} \frac{1}{rank_u}\,, \tag{2.30}$$

where $rank_u$ is the rank position of the first relevant item for user u. For example, let us consider a RS which contains 3 users and recommends lists of items of size $l = 5$. If the binary relevancy scores for each user are respectively: $\{0, 0, 1, 1, 0\}$, $\{0, 0, 0, 1, 0\}$ and $\{1, 1, 0, 1, 0\}$, the rank of the first relevant item in the lists of the users are respectively 3, 4 and 1. Thus, the MRR score is:

$$MRR = \frac{1}{3} \sum_{u=1}^{3} \frac{1}{rank_u} = \frac{1}{3} \cdot (\frac{1}{3} + \frac{1}{4} + \frac{1}{1}) = 0.527\,.$$

- The Expected Reciprocal Rank (ERR) (Chapelle et al., 2009) also uses the reciprocal rank, but is a bit more complex metric. It makes the assumption of a cascade model for the user behavior, meaning that the user will look at each item of the ranked list one after another until he picks one. The user sweeps through the recommended items from the first item to the one it chooses, but stops looking at the list once he picked an item. The ERR for a list of size l is defined as follow:

$$ERR@l = \sum_{r=1}^{\ell} \frac{1}{r} p(u, i_r) \prod_{s=1}^{r-1} (1 - p(u, i_s))\,, \tag{2.31}$$

with

$$p(u, i) = \frac{2^{r^*_{u,i}} - 1}{2^{r_{max}}}\,. \tag{2.32}$$

The probability $p(u,i)$ in Equation (2.32) defines a probability for the user to be satisfied by an item. The ERR in Equation (2.31) is the expectation of the reciprocal of the position of a result at which a user stops. The product in the equation represents the probability of not having been satisfied by any of the items seen until the item r.

Let us take an example where a list of items of size $l = 3$ is recommended to a user, and where relevance scores range from 1 to 5 stars ($r_{max} = 5$). The user provides the following scores for the recommended items: $\{3, 5, 2\}$.

The Table 2.3 provides the way to compute the ERR score, where $p(\text{stop at item } r)$ is the same as $p(u,i_r)\prod_{s=1}^{r-1}(1-p(u,i_s))$ in Equation (2.31).

TABLE 2.3: An example on how to calculate ERR score.

r	$1/r$	r^*_{u,i_r}	$p(u,i)$	$p(\text{stop at item } r)$
1	$1/1$	3	$7/32$	$7/32$
2	$1/2$	5	$31/32$	$31/32 \cdot (1 - 7/32)$
3	$1/3$	2	$3/32$	$3/32 \cdot (1 - 31/32) \cdot (1 - 7/32)$

The ERR score is obtained by multiplying the reciprocal rank with the stop probabilities:

$$ERR@3 = \frac{1}{1}\cdot\frac{7}{32} + \frac{1}{2}\cdot\frac{31}{32}\cdot(1-\frac{7}{32}) + \frac{1}{3}\cdot\frac{3}{32}\cdot(1-\frac{31}{32})\cdot(1-\frac{7}{32}) = 0.598\,.$$

Other Evaluation Measures

The metrics defined previously are important for evaluation to compare mathematically different recommendation techniques, but they are not focused on how the user perceives the recommendations produced by the RS (Vargas and Castells, 2011). Several key metrics allow to measure such an effect, and are gaining importance in the RS field. Since these criteria are more complex to evaluate and several formulas have been given in the literature, we refer to the main papers for each metric to get more details.

- **Novelty**. This is related to RS which should recommend items of which the user is not aware. For example, recommending a Star Wars movie to a Science-Fiction fan is not a novel suggestion, since the user is surely already aware of this movie. See (Celma and Herrera, 2008; Zhou et al., 2010; Vargas and Castells, 2011) for examples of how to measure novelty in RS.

- **Diversity**. This measures to which extent the RS is capable of suggesting items that are "different" from each other. Often, a RS would recommend very similar items to a user (for example, books by the same author), while a good RS should recommend items that span over the whole user's interest. The diversity has been measured with different definition: for example (Ziegler et al., 2005) measures the average pairwise dissimilarity between recommended items, while (Zhang and Hurley, 2008) addresses diversification by optimizing jointly two objective functions reflecting preference similarity and item diversity.

- **Serendipity**. It measures the capability of a RS to suggest items that would surprise the user while still matching his/her tastes. Indeed, a good RS should not recommend too obvious recommendations but also offer unexpected items to the user. Some measures to evaluate serendipity can be seen in (Ge et al., 2010; Murakami et al., 2007).
- **Coverage**. This measure focuses on the percentage of elements (users, items or ratings) for which the RS is able to provide recommendations. Several coverages can be defined: for example, the user coverage measures how large is the set of users for which the RS can perform recommendations, while the catalog coverage measure the percentage of (user-item) pairs for which prediction can be made over the set of all (user-item) pairs. Coverage can also help to notice if the RS is biased towards some specific subset of users or items (Ge et al., 2010).
- **Abandonment** The abandonment metric is used when a list of items is suggested to the user. A user is said to "abandon" the system when none of the items recommended in the list are relevant for him. The abandonment is the probability for a user to leave the system within a session, and should be minimized by the RS (Radlinski et al., 2008; Kohli et al., 2013).

2.6.2 Evaluation Scenarios

After choosing the evaluation measure, a proper evaluation scenario and experimental settings must be set to judge of the success of the designed recommendation method. There are three categories into which the evaluation methodology usually falls into: the online setting, the user studies and the offline setting.

Online Evaluation

The online evaluation is only possible when the RS implementation team has a direct access to a real system deployed online. In this case, the RS evaluation can use real users to gather feedback and be evaluated with little bias, as users are directly using the system. Usually, in case of successive refinements and implementations of the system, a RS prototype would be displayed and tested only on a subset of randomly selected users in the system while other users would still use the more stable and previous version of the RS. This allows to avoid having too much of an impact on the website reputation in case the new RS has some unseen flaws.

To compare several algorithms, most companies set up a method called A/B testing. Let us suppose there are two algorithms: users are first divided into two groups A and B, and each group see recommendations generated by one of the two RS algorithms. Then, at the end of the testing phase, both methods can be compared on some metrics, for example the conversion rate which evaluates how many user ended up choosing an item appearing in the recommendation list.

Another perspective is to use a Multi-Armed Bandit (MAB) algorithm, which aims to find the best algorithm among a set of RS algorithms. The basic idea is to select one of these algorithms each time a user requests a recommendation. The selection is either

- an exploration step, where the algorithm is chosen at random (or based on its uncertainty)
- or an exploitation step, where the knowledge accumulated until the current time step is used to choose the best recommendation algorithm until now

This approach allows to converge to the best RS after a certain number of recommendations.

However, the issue with online studies is that commercial RS are not publicly accessible and test on these systems can only be performed by people to which the access is granted. This implies online tests are not replicable by other researchers. Another issue is that a high number of users is necessary to compare several algorithms and have a significant evaluation. For these reasons, researchers usually turn to the two other possibilities described below.

User studies

If the online evaluation cannot be conducted because it is too risky or because no access is given to a real-world application, the user study is another possible choice to get results from real users. In this case, a small number of users are recruited and requested to interact with the RS being tested. A controlled experiment is set up to evaluate the system.

User studies allow researchers to collect information about the users' interaction with the system, and ask them about their experience afterwards. Hence, the system can be judged on more criteria compared to online or offline studies where no qualitative answer is given by the user. However, the small number of users can introduce a bias in the data being retrieved. The awareness of the user about being tested to evaluate a RS can also introduce another bias. Results of user study need to be carefully considered and trusted.

Offline Evaluation

At last, the offline evaluation consists in comparing several RS algorithms on some datasets to compare their performance, without needing users to interact with the RS. One or several of the metrics defined in Section 2.6.1 are used to judge the performance of each approach. Usually, the datasets are either made of private collected data or are publicly available datasets, the most popular example being the Movielens dataset [2]. Offline evaluation is among the most popular setting in research since evaluation frameworks,

[2] http://grouplens.org/datasets/movielens/

metrics and datasets have been developed and are easily accessible to replicate or compare approaches with other algorithms.

As there is no real interaction between the RS and users, using offline experiments will make the assumption that the behavior of the users at the time the data was collected and the future behavior of these users will be similar. The data is often divided into three distinct parts: the training set, used to build the model of the RS, the validation set which is used for the model selection and parameter tuning, and the testing set which is built to evaluate the chosen metric of the final model. The separation of the sets is usually done by selecting at random a given percentage of ratings for each set (for example 80% for the training set, 10% for the validation set and 10% for the testing set), using a hold-out method. This means a rating chosen in a set can only belong to this set. This is meant to avoid learning the model and evaluating the RS on the same data.

If the temporal information is available about the rating, it is also possible to divide the set by taking only ratings which were given before a time threshold in the train and validation set, and evaluate on ratings given after this threshold, to simulate more realistic conditions about the user's preferences.

Some experiments evaluate on a specific scheme about the user train set: for each user, they select a given number n of items to build the train set and learn the model, and then the evaluation is done on all the remaining items. This approach is usually described with the annotation "Given n". In the case where a low value is selected for n, this setting allows to test the RS robustness to a user cold-start situation where only little information is known about users.

Finally, an evaluation in the Multi-Armed Bandits perspective in the offline case is introduced in (Li et al., 2011).

Despite most researches using an offline protocol for evaluation, they have some serious drawbacks:

- Firstly, there is no certainty that results obtained from offline evaluation would always align with actual results in a real-world RS. The behavior of user in the real RS might be different from the one supposed in the offline setting.
- Secondly, there are various metrics which can be used in offline metrics, and all evaluate the performance based on past preferences of the users, but these metrics cannot measure the real impact of recommendations on real user activity.
- At last, the treatment given to missing ratings is problematic. How should they be considered? Does it mean the user does not like the items which he did not rate? Does it means he does not know them, or that he does not care?

The only solution to these drawbacks is to test algorithms online to confirm or reject results obtained offline.

2.7 Learning to Rank

All models described previously treat the Recommendation Problem using a prediction technique, where the squared error of the rating prediction is minimized. However, this approach has an important flaw, as in real-world Recommendation, only the top-k items are displayed to the user, and it is not important for the RS to have a good prediction accuracy on items that are not recommended.

A simple example to explain why prediction accuracy does not target the correct objective is to imagine a Recommendation System making very accurate predictions on the items with low or medium ranking, but bad predictions on items reaching the highest ratings. The average performance of such RS from a prediction accuracy point of view would not be too bad, as only the highest scores would obtain a high error and these top-rated items are not a majority among all items. However, from the point of view of the user, it would greatly hurt his experience of the system, as items which match the best his tastes would not be recommended.

For these reasons, some approaches have focused on optimizing the order of the list of recommendation instead of focusing on predicting correct ratings (Liu, 2009). Such methods are usually divided into three categories:

- Pointwise approaches: this is the simplest way to tackle the Learning to Rank problem. It consists in using techniques trying to optimize prediction accuracy metric (for example, using Matrix Factorization techniques presented in Section 2.3.2), and simply ranking items based on the rating predictions made by the algorithm. As a consequence, pointwise approaches do not target directly a ranking optimization, and other approaches are often preferred when solving a ranking problem.
- Pairwise approaches: these methods use pairs of items as input for training data instead of single rating. For a user u, a pair of item (i, i') is used as training data only if both ratings $r_{u,i}$ and $r_{u,i'}$ are known. Each pair of items brings information about whether the user showed more preference for item i compared to item i', or the opposite case. If the first case happens, then the pair is labeled with a +1, otherwise with a -1.
- Listwise approaches: finally, listwise rank methods make use of the entire list of known ratings and focus on optimizing a ranking-based objective function. Example of such functions are the NDCG and the MRR ranking measures. However, compared to prediction tasks in which the objective function is smooth, ranking functions are often non-smooth, which makes it hard to optimize. For this reason, a smooth approximation of the non-smooth objective function is often adopted to simplify the problem. The ranking accuracy of the algorithm will then depend on the quality of the approximation. Examples of listwise Learning to Rank algorithms applied for RS are (Weimer et al., 2008; Shi et al., 2010; Shi et al., 2012; Shi et al., 2013). These approaches are described in more details in Chapter 4.

If features are available for the RS to learn, other popular Machine Learning exist in the Learning To Rank field to solve the recommendation problem as a traditional Machine Learning regression problem (Burges et al., 2005). The most popular ones are approaches developed by C. Burges based on Gradient Boosted Trees and described in (Burges, 2010). These methods have shown especially a good success in Learning To Rank recommendation challenges where methods have access to attributes (Chapelle and Chang, 2011; Guillou et al., 2014).

2.8 Conclusion

This chapter aimed at giving some background knowledge about RS: the sort of data they use, their goals, the possible approaches and how they are evaluated. Among all techniques, Matrix Factorization approaches have been shown to be among the most efficient and widely used in the industrial world, and several algorithms have been proposed to incorporate the effect of social factor, time factor, location factor...

In the following parts of this thesis, we will focus on two of the current challenges in RS:

- **How to reduce the cold start effect and learn user or item's profile as quickly as possible?**

  Many solutions have been proposed to use external information about the user, the item, the context... or to do Active Learning to query the user before the recommendation process to model him. However, in practice, asking the user is expensive and it is sometimes not possible to get external information other than the past feedback.

- **How to evaluate properly the recommendation process?**

  The question of which evaluation metric to use has been a hot topic in recent years, with the metrics used in research slowly shifting from prediction metrics like RMSE to ranking metric such as NDCG. However, almost all of the approaches evaluated offline still use a split method between train and test data and evaluate their approaches in a fixed setting, which is not a close evaluation to the real process of recommendation in a real-world RS.

The reason behind the choice of these two topics is that both the challenge of cold-start and the challenge of evaluation appear from a different perspective in a sequential context. First, in such context, the cold-start situation becomes more integrated into the recommendation problem, where the model of the RS has to continuously gather information and provide accurate recommendation to the users. Second, when evaluated in a sequential manner, the evaluation metric needs to be optimized through time, to be maximized (or minimized) in the long-term instead of retrieving a single score for a fixed point in time.

Since recommendations are provided sequentially in a real-world RS, we argue in the following chapters that the problem should be viewed within the sequential context and also evaluated as such. Some intrinsic questions

and issues related to sequential learning, that are not usually taken into consideration in the RS field, are raised and studied. In the next chapter, we introduce one of these issues known as the exploration-exploitation dilemma, and propose a model aiming at tackling this dilemma for RS, based on Matrix Factorization and Multi-Armed Bandits.

Chapter 3

Recommendation as a Multi-Armed Bandit

In this chapter [1], we set up a framework to study recommendation as a sequential process, and we propose SeALS, an approach which is based on Matrix Factorization (MF) methods for Collaborative Filtering (CF), seen in Section 2.3 in the previous chapter, and on a Multi-Armed Bandits (MAB) algorithm. We introduce the first sequential CF-based RS which:

1. makes a good trade-off between exploration and exploitation through time, by incorporating the use of MAB algorithm in a simple way into the recommendation process;
2. is able to properly scale. As the potential number of user and items in RS is really high, a scalable method is a necessity.

We conduct extensive experiments on real-world datasets of millions of ratings to highlight the need for a trade-off between exploration and exploitation for RS, and to study how efficient is our approach time wise.

Contents

[1] This chapter is an extended version of a preliminary work presented at a NIPS workshop (Guillou et al., 2015) and two papers published in PACIS (Guillou et al., 2016b) and MOD (Guillou et al., 2016a)

3.1 Sequential Recommendation

A RS built with a CF algorithm recommends items to users based on each user's tastes as inferred from past user behavior and feedback. The past feedback is stored in a matrix of ratings $\mathbf{R}^*$, where only a few entries are known. In the context of CF, the RS recovers unknown values in $\mathbf{R}^*$ and the traditional evaluation is performed by splitting log data into two parts: the train-set is used to define the training matrix which is completed by the RS algorithm, while the test-set is used to measure the quality of the matrix returned by the RS. Common measures of that quality are prediction accuracy metrics like MAE and RMSE on the test-set.

While such a static batch evaluation makes sense to measure the quality of the matrix-completion step of Collaborative Filtering, it does not evaluate the quality of the final recommendation. A CF based RS works in reality in a sequential manner and loops through the following steps:

1. Build a model of the users' tastes from past feedback;
2. Recommend an item to a user using this model;
3. Gather feedback from user about recommended product.

This loop is represented in the Figure 3.1.

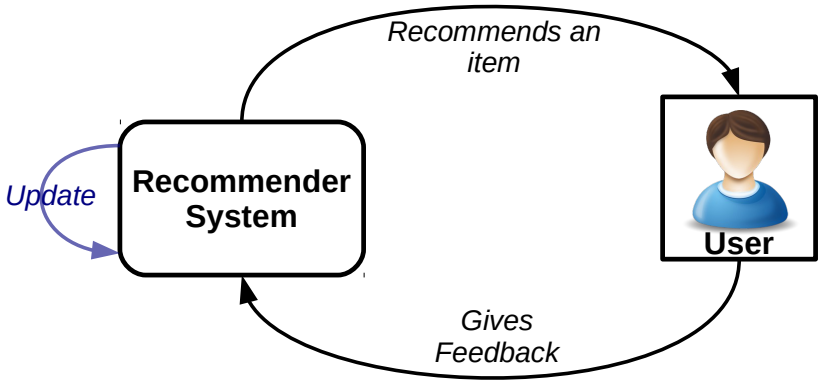


FIGURE 3.1: The sequential recommendation process. After the RS recommends an item, the user gives back a feedback, and the RS possibly update its model.

Note that the model built at step 1 heavily depends on the feedback gathered at previous iterations. This feedback only exists for items which were chosen by the model itself. As such, at step 2, one faces the exploration/exploitation dilemma, which means the RS can either:

- **Exploit**: recommend an item which led to the best feedback in the past.
- **Explore**: recommend an item which hopefully brings information on the user's tastes.

This dilemma is the core point of Multi-Armed Bandit Theory (Auer et al., 2002). It is already studied in a sub-field of RS which has access to a representation of the context (the user, the webpage ...) (Tang et al., 2014). Typical applications are the selection of news or ads to show on a web-page. The corresponding RS builds upon contextual bandits which are supported by strong theoretical results.

In contrast with these studies, we focus on the setting where these representations are unknown and have to be inferred solely from users' feedback. In particular, we want to emphasize that we do not use any side information, neither about users, nor items. That field of research is almost empty and the few attempts therein leave out computational complexity constraints (Kawale et al., 2015).

3.1.1 Formulation of the Sequential Recommendation Scenario

Let us focus on a particular recommendation scenario, which illustrates more accurately how typical Recommendation Systems work.

We consider a RS with N users, M items, and the unknown matrix $\mathbf{R}^*$ of size $N \times M$ such that $r^*_{u,i}$ is the tastes of user u with regards to item i. At each time step t,

1. a user u_t requests a recommendation from the RS,
2. the RS selects an item i_t among the set of available items,
3. user u_t returns a feedback $r_t \sim \mathcal{D}(r^*_{u_t,i_t})$ for item i_t.

In this chapter, we assume the mean of distribution $\mathcal{D}(r^*_{u_t,i_t})$ to be $r^*_{u_t,i_t}$. See (Koren and Sill, 2011) for an example of a more refined observation/noise model.

In the second step, it is mentioned that the RS can play only among available items. For one user, the set of items among which the RS will select the recommendations can indeed be restricted to items which are available in stock for a e-commerce website for example (cf. Chapter 5 for a more detailed discussion on this topic). In this chapter, we will not consider any stock or restriction about availability.

We refer to applications where the feedback r_t corresponds to the quantity that has to be optimized, aka. *the reward*. This denomination comes from the Reinforcement Learning, where the learning agent receives a reward to learn about future actions to be taken. Compared to a fixed setting where the evaluation metric is usually optimized for a situation corresponding to a precise and unique time step, the aim of the RS in a sequential context is to maximize the reward accumulated along time steps, which is denoted as *cumulative reward* and given by Equation (3.1):

$$\mathrm{CumRew}_T = \sum_{t=1}^{T} r_t \,. \tag{3.1}$$

Another measure used in a Multi-Armed Bandit problem is the *cumulative pseudo-regret* $\mathcal{R}_T$ described in Equation (3.2), which consists in a cumulative score, obtained by measuring how much the system loses by recommending a sub-optimal item at each time step:

$$\mathcal{R}_T = \sum_{t=1}^{T} \max_i r^*_{u_t,i} - \mathbb{E}[r_t] = \sum_{t=1}^{T} \max_i r^*_{u_t,i} - r^*_{u_t,i_t} \,. \tag{3.2}$$

We will propose in the following a solution which aims at minimizing this cumulative pseudo-regret.

Along this chapter, we use the following notations:

- We denote $\mathbf{R}_t$ the partially known $N \times M$ matrix such that $r_{u_s,i_s} = r_s$ for any $s \leqslant t$.
- We note $\mathcal{S}_t$ the set of known entries of $\mathbf{R}_t$:

$$\mathcal{S}_t = \{(u_s, i_s); s \leq t\} \,.$$

- $\mathcal{I}_t(u)$ the set of items i for which u gave his preferences until time t:

$$\mathcal{I}_t(u) = \{i; (u,i) \in \mathcal{S}_t\} \,.$$

- $\mathcal{J}_t(i)$ the set of users u which gave their preferences on item i until time t:

$$\mathcal{J}_t(i) = \{u; (u,i) \in \mathcal{S}_t\} \,.$$

For the sake of readability, the subscript t is omitted in the following. Finally, for any matrix $\mathbf{M}$, we denote $\mathbf{M}_i$ the i-th row of $\mathbf{M}$.

We introduce in Section 3.4 a Recommendation System which handles sequential recommendations and aim at maximizing the ratings received through time. This RS is composed of two main ingredients:

1. a model built using Matrix Factorization, to infer an estimate $\widehat{\mathbf{R}}^*$ of the matrix $\mathbf{R}^*$ from known values in $\mathbf{R}$,
2. a strategy to choose the item to recommend given $\widehat{\mathbf{R}}^*$

This strategy aims at balancing exploration and exploitation, and builds on Multi-Armed Bandits for this task. For the state of the art for Matrix Factorization, see the Section 2.3 in the previous chapter. Before reviewing state of the art approaches for Multi-Armed Bandits, described in the Section 3.3, we first review the work related to our approach in the next section.

3.2 Related Work

To the best of our knowledge, only few papers consider a RS setting similar to the one presented in this chapter, where exploration/exploitation dilemma is tackled with ratings only (Zhao et al., 2013; Xing et al., 2014; Nakamura, 2014; Kawale et al., 2015; Mary et al., 2015).

(Zhao et al., 2013) propose an approach called Interactive Collaborative Filtering (ICF), which combines Probabilistic Matrix Factorization with various bandits algorithms. However, they perform an evaluation where they split between train and test set, because they require for their approaches to learn in advance the item feature vectors (these feature vectors are considered as known and fixed during the test phase). In our work, we assume no prior knowledge about neither users nor items.

(Xing et al., 2014) combines Matrix Factorization with Bayes-UCB approach for Multi-Armed Bandit specifically for music recommendation, but only evaluate with a user study on eighteen people and 200 time steps of recommendation which is a very small window of time. (Nakamura, 2014) considers a UCB-like strategy to a specific problem (called the "direct mail problem") which is different from our setting. The direct mail problem appears when a systems has to select appropriately each day a set of (user-item) pairs to send a recommendation mail of item i to user u. In this problem, a user can only receive one recommendation per day, and the number of pairs selected every day is restricted to some number. Then, (Mary et al., 2015) also focus on a different recommendation setting where an item can be recommended one time only per user. Their approach builds upon ALS Matrix Factorization framework and extends linear bandits (Li et al., 2010) to handle the exploration/exploitation dilemma. The use of linear bandit framework prevents this approach from scaling to a high number of recommendations.

Finally, the approach PTS is the most recent one and is introduced in (Kawale et al., 2015), tackles the exact same problem as our approach. PTS is based on Thompson Sampling (Chapelle and Li, 2011) and Particle Filtering. However, their approach requires a huge computing time (cf. at the end of Section 3.4). As a consequence, (Kawale et al., 2015) only provides experiments on “small" datasets, with $k = 2$. SeALS is the first approach which both tackles the exploration/exploitation dilemma and scales up well to large datasets and high number of recommendations, while building an accurate representation of users and items ($k \gg 2$).

Apart from methods quoted above which tackle the non-contextual case, the exploration vs. exploitation dilemma is already well-studied in the field of RS which has access to a representation of the context (Tang et al., 2014). Compared to them, we focus on the setting where these representations are unknown and have to be inferred from users feedback.

Some papers focusing on the cold-start setting also focuses on the need for exploration (Agarwal et al., 2008; Bhagat et al., 2014): the goal in this case is to deal with new users or new items. While some approaches look at external information on the user (Agarwal et al., 2008), some papers rewrite the problem as an Active Learning problem (Bhagat et al., 2014): in this case, the goal is to perform recommendation in order to gather information on the user as quickly as possible. Targeted applications would first “explore” the user and then “exploit” him. Unlike Active Learning strategies, (i) we spread the cost of exploration along time, and (ii) we handle the need for a never-ending exploration to reach optimality (Auer et al., 2002).

Salah et al. (Salah et al., 2015) propose an incremental neighborhood-based CF RS based on weighted clustering approach suitable for incremental update of the RS, while other methods like the ones proposed by Rendle and Schmidt-Thieme (Rendle and Schmidt-Thieme, 2008) or Luo et al. (Luo et al., 2012) focus on online Matrix Factorization using a stochastic gradient descent to solve Equation (3.7), but they all omit the exploration/exploitation dilemma.

Finally, some researches focus on a ranking algorithm instead of trying to target a good RMSE score. Cremonesi et al. (Cremonesi et al., 2010) compare state of the art recommendation algorithms with respect to a rank-based scoring and shows that winning algorithms are not the ones which reach the smallest RMSE score. Following the same guideline, (Rendle et al., 2009; Shi et al., 2012; Weston et al., 2013) propose RS which directly target a good ranking of the top items instead of a full-completion of the matrix. During the training phase, they replace the L2 loss of Equation (3.7) by rank-based losses (AUC, MRR, NDCG...). While targeting a rank-based measure during the training phase could increase the accuracy of the RS, (Garcin et al., 2014) and (Said and Bellogin, 2014) show that the cumulative reward/regret is the unique good metric to evaluate the recommendation algorithm.

3.3 Multi-Armed Bandits

A RS works in a sequential context: as a consequence, while the recommendation made at time step t aims at collecting a good reward at the present time, it affects the information that is collected, and therefore also the future recommendations and rewards. Specifically, in the context of sequential decision under uncertainty problems, an algorithm which focuses only on short term reward loses w.r.t. expected long term reward. This Section recalls standard strategies to handle this short term vs. long term dilemma. For ease of understanding, the setting used in this Section is much simpler than the one faced in this thesis.

3.3.1 Setting

We consider the well-studied Multi-Armed Bandits (MAB) setting (Auer et al., 2002; Bubeck and Cesa-Bianchi, 2012): we face a bandit machine with M independent arms $\{i_1, \dots i_M\}$. At each time step, we pull an arm i and receive a feedback (aka. reward) drawn from $[0, 1]$ which follows a probability distribution ν_i. Let μ_i denote the mean of ν_i, $i^* = \operatorname{argmax}_i \mu_i$ be the best arm and $\mu^* = \max_i \mu_i = \mu_{i^*}$ be the best expected reward. The parameters $\{\nu_i\}$, $\{\mu_i\}$, i^* and μ^* are unknown.

We play T consecutive times and aim at maximizing the *cumulative reward*

$$\mathrm{CumRew}_T = \sum_{t=1}^{T} r_t \,, \tag{3.3}$$

where r_t denotes the reward collected at time step t while pulling arm i_t. As the parameters are unknown, at each time step (except the last one), we face the dilemma:

- either focus on short-term reward (aka. *exploit*) by pulling the arm which was the best at previous time steps;
- or focus on long-term reward (aka. *explore*) by pulling an arm to improve the estimation of its parameters.

Neither of these strategies is optimal. On one hand, an algorithm which acts greedily by only exploiting is likely to miss the optimal arm.[2] On the other hand, the pure-explore strategy only plays the optimal arm with probability $1/M$. To be optimal, a strategy has to balance exploration and exploitation. We review briefly in the following some of the well-known strategies tackling this issue.

3.3.2 Approaches

Upper Confidence Bound P. Auer (Auer et al., 2002) introduces an approach to handle this exploration vs. exploitation dilemma: the Upper Confidence Bound strategy (UCB). UCB balances exploration and exploitation by playing the arm i_t such that:

$$i_t = \underset{i}{\operatorname{argmax}}\, \hat{\mu}_i(t) + \sqrt{\frac{2 \ln t}{T_i(t)}}\,, \tag{3.4}$$

where $T_i(t)$ corresponds to the number of pulls of arm i since the first time step and $\hat{\mu}_i(t)$ denotes the empirical mean reward obtained from arm i up to time t.

This equation embodies the exploration vs. exploitation trade-off: while $\hat{\mu}_i(t)$ promotes exploitation of the arm which looks like optimal, the second term of the sum promotes exploration of less played arms.

At the beginning of the algorithm, since no arm was played and the number of pulls $T_i(t) = 0$ for all arms, each of the arm is play once to get initial values for the empirical mean reward. If the number M of arms is really high, it means that the number of times the algorithm will play needs to be far greater than M to converge to good results. Algorithm of UCB1 is described in Algorithm 4.

While UCB is optimal up to a constant, there exists several flavors of UCB-like algorithms (Audibert et al., 2009; Garivier and Cappé, 2011; Li et al., 2010) aiming at a strategy closer to the optimal one or aiming at a strategy which benefits from different constraints on the reward distribution.

[2] Remind that the rewards are random; as a consequence the first pulls of the optimal arm may yield rewards smaller than μ^*. In the worst case, these rewards are small enough so that a sub-optimal arm i to seem optimal and be chosen. As the optimal arm is not pulled, its empirical mean remains small, and this arm is no more pulled.

Algorithm 4: UCB1.

Input: Set of M arms

Play each of the M arms once, giving initial values for the empirical mean reward of each arm i and put $T_i = 1$ for each arm;

for $t = M, M+1 \ldots$ **do**

  Play $i_t = \text{argmax}_i \, \hat{\mu}_i(t) + \sqrt{\frac{2 \ln t}{T_i(t)}}$;

  Observe the reward r_t for arm i_t ;

  Update the empirical mean: $\hat{\mu}_i \leftarrow \hat{\mu}_i + r_t$;

  Update the number of times the arm i was played: $T_i \leftarrow T_i + 1$;

end

Epsilon-Greedy ε-greedy is an even simpler but efficient approach to balance exploration and exploitation (Auer et al., 2002). It consists in choosing the arm to play with the greedy strategy ($i_t = \text{argmax}_i \, \hat{\mu}_i(t)$) (exploitation) with a probability $1 - \varepsilon$ and in pulling an arm at random otherwise (exploration). Algorithm of ε-greedy is described in Algorithm 5.

Algorithm 5: ε-greedy.

Input: Set of M arms

for $t = 1, \ldots$ **do**

  if $p \sim U(0,1) < \varepsilon$ **then**

    Play the arm: $i_t \leftarrow random(i \in \{i_1, \ldots i_M\})$;

  else

    Play the arm $i_t \leftarrow \underset{i \in \{i_1, \ldots i_M\}}{\text{argmax}} \, \hat{\mu}_i(t)$;

  end

  Observe the reward r_t for arm i_t ;

  Update the empirical mean: $\hat{\mu}_i \leftarrow \hat{\mu}_i + r_t$;

end

If ε is set to a constant, exploration will be scattered across all iterations permanently, but usually it is preferable to explore more at the beginning when arms are unknown and exploit further after. Instead of a fixed ε, ε_n-greedy uses an ε_t in line 2 of Algorithm 5. ε_t is a function of t, for example α/t with α a constant which depends on N. This makes exploration stronger at the beginning, with more exploitation afterward.

Thompson Sampling Another well-known approach to solve the exploration vs. exploitation dilemma in a Bayesian way is Thompson Sampling (Thompson, 1933; Bubeck and Cesa-Bianchi, 2012).

With θ denoting the set of parameters of the distribution of the reward, and D being the past observations currently available, the system chooses the arm i with probability:

$$\int \mathbb{I}[\mathbb{E}[r|i,\theta] = max_{i'} \mathbb{E}[r|i',\theta]] P(\theta|D) d\theta \,, \tag{3.5}$$

where $\mathbb{I}[a = b]$ stands for 1 when $a = b$ and 0 otherwise. This is usually implemented by sampling θ from the posterior $P(\theta|D)$ and choosing the arm which maximizes the expected reward given the parameters and the action taken $i = \text{argmax}_{i'} \mathbb{E}[r|i', \theta]$. While there are few theoretical results related to these strategies (Kaufmann et al., 2012), they demonstrate good empirical results on some occasions (Chapelle and Li, 2011).

3.4 Explore-exploit Recommendation System

In this section, we introduce a RS which handles the sequential aspect of recommendation. More specifically, the proposed approach works in the context presented in Section 3.1 and aims at minimizing the pseudo cumulative regret $\mathcal{R}_T$ defined in Equation (3.2). As needed, the proposed approach balances exploration and exploitation.

We first define the model of our RS, making use of Matrix Factorization model (Koren et al., 2009): the unknown matrix $\mathbf{R}^*$ is assumed to be of low rank. Namely, there exist $\mathbf{U}$ and $\mathbf{V}$ such that

$$\mathbf{R}^* = \mathbf{U}\mathbf{V}^T , \tag{3.6}$$

where

- $\mathbf{U}$ is a matrix of size $N \times k$ representing users features,
- $\mathbf{V}$ is a matrix of size $M \times k$ representing items features,
- k is the rank of $\mathbf{R}^*$, and $k \ll \max(N, M)$.

Thereafter, the estimator of $\mathbf{R}^*$ is defined as $\widehat{\mathbf{R}}^* = \widehat{\mathbf{U}}\widehat{\mathbf{V}}^T$ where $(\widehat{\mathbf{U}}, \widehat{\mathbf{V}})$ is the solution of

$$\underset{\mathbf{U},\mathbf{V}}{\text{argmin}} \sum_{\forall (u,i) \in \mathcal{S}} \left(r_{u,i} - \mathbf{U}_u \mathbf{V}_i^T\right)^2 + \lambda \cdot \Omega(\mathbf{U}, \mathbf{V}) . \tag{3.7}$$

We choose in our approach to build the MF model by using ALS-WR (Zhou et al., 2008), which regularizes users and items according to their respective importance in the matrix of ratings:

$$\Omega(\mathbf{U}, \mathbf{V}) = \sum_u \#\mathcal{I}(u) ||\mathbf{U}_u||^2 + \sum_i \#\mathcal{J}(i) ||\mathbf{V}_i||^2 . \tag{3.8}$$

This regularization is known to have a good empirical behavior — which means a limited overfitting, an easy tuning of λ and k, and a low RMSE. This is also the default one in Mahout[3], an open-source platform about Recommendations Systems providing CF algorithms, which is widely used in real-world systems worldwide. The algorithm of ALS-WR is given in details in Algorithm 6.

[3] https://mahout.apache.org/

Algorithm 6: ALS-WR: finds a solution to Equation (3.7) with an alternating least square approach.

Input : Matrix of observed ratings: $\mathbf{R}$,
Set of known ratings: $\mathcal{S}$,
regularization parameter: λ,
Input/Output: User features matrix: $\widehat{\mathbf{U}}$
Item features matrix: $\widehat{\mathbf{V}}$

1 **for** $u \in 1, \dots N$ **do**
2 $\quad \widehat{\mathbf{U}}_u \leftarrow \text{argmin}_{\mathbf{U}} \sum_{i \in \mathcal{I}(u)} (r_{u,i} - \mathbf{U}\widehat{\mathbf{V}}_i^T)^2 + \lambda \cdot \#\mathcal{I}(u)||\mathbf{U}||$;
3 **end**
4 **for** $i \in 1, \dots M$ **do**
5 $\quad \widehat{\mathbf{V}}_i \leftarrow \text{argmin}_{\mathbf{V}} \sum_{u \in \mathcal{J}(i)} (r_{u,i} - \widehat{\mathbf{U}}_u\mathbf{V}^T)^2 + \lambda \cdot \#\mathcal{J}(i)||\mathbf{V}||$;
6 **end**

Aside from the model of our RS which is built upon ALS-WR Matrix Completion approach, we integrate an ε_n-greedy strategy to tackle the exploration/exploitation dilemma during the recommendation process. We name this approach **SeALS** (for *Sequential ALS-WR*) and describe it in Algorithm 7.

Algorithm 7: SeALS: recommends in a sequential context.

Input : period of the update: T_u,
proportion for the size of mini-batch update: p,
regularization parameter: λ,
exploration parameter: α
Input/Output: Matrix of observed ratings: $\mathbf{R}$,
Set of known ratings: $\mathcal{S}$

1 $(\widehat{\mathbf{U}}, \widehat{\mathbf{V}}) \leftarrow$ ALS-WR$(\mathbf{R}, \mathcal{S}, \lambda)$;
2 **for** $t = 1,\ 2,\ \dots$ **do**
3 $\quad$ get user u_t and set $\mathcal{L}_t$ of allowed items;
4 $\quad$ **if** $p \sim U(0,1) < \varepsilon$ **then**
5 $\quad\quad i_t \leftarrow random(i \in \mathcal{L}_t)$;
6 $\quad$ **else**
7 $\quad\quad i_t \leftarrow \underset{i \in \mathcal{L}_t}{\text{argmax}}\, \widehat{\mathbf{U}}_{u_t} \cdot \widehat{\mathbf{V}}_i^T$;
8 $\quad$ **end**
9 $\quad$ recommend item i_t and receive rating $r_t = r_{u_t,i_t}$;
10 $\quad$ update $\mathbf{R}$ and $\mathcal{S}$;
11 $\quad$ **if** $t \equiv 0 \mod T_u$ **then**
12 $\quad\quad (\widehat{\mathbf{U}}, \widehat{\mathbf{V}}) \leftarrow$ mBALS-WR$(\widehat{\mathbf{U}}, \widehat{\mathbf{V}}, \mathbf{R}, \mathcal{S}, \lambda, p)$;
13 $\quad$ **end**
14 **end**

The algorithm first creates $\widehat{\mathbf{U}}$ and $\widehat{\mathbf{V}}$ by using ALS-WR on the matrix of ratings $\mathbf{R}$ on line 1. At a time step t, for a given user u_t, ALS-WR associates an expected reward $\hat{r}_{u_t,i} = \widehat{\mathbf{U}}_{u_t} \cdot \widehat{\mathbf{V}}_i^T$ to each item i. Then the item to recommend i_t is chosen by an ε_n-greedy strategy, described in lines 4 - 8. Despite its simplicity, ε_n-greedy strategy is extremely efficient in such context.

Obviously, ALS-WR requires too large computation times to be run at each time step to recompute user and item features. A solution consists in running ALS-WR every T_u time steps as written in lines 11-13 of Algorithm 7. While such a strategy works well when T_u is small enough, $\mathcal{R}_T$ drastically increases otherwise (cf. Section 3.5.4). Taking inspiration from Stochastic Gradient Descent approaches (Bottou and Bousquet, 2007), we solve that problem by designing a mini-batch version of ALS-WR, denoted mBALS-WR (cf. Algorithm 8).

Algorithm 8: mBALS-WR: mini-batch version of ALS-WR.

Input : Matrix of observed ratings: $\mathbf{R}$ (of size $N \times M$),
Set of known ratings: $\mathcal{S}$
regularization parameter: λ,
exploration parameter: α
proportion for the size of mini-batch update: p

Input/Output: Matrix of users' features: $\widehat{\mathbf{U}}$,
Matrix of items' features: $\widehat{\mathbf{V}}$

1 Sample randomly $p \cdot N$ users among all users, to insert in a list l_{users};

2 Sample randomly $p \cdot M$ items among all items, to insert in a list l_{items};

3 $\forall u \in l_{users}$, $\widehat{\mathbf{U}}_u \leftarrow \text{argmin}_{\mathbf{U}} \sum_{i \in \mathcal{I}(u)} \left(r_{u,i} - \mathbf{U}\widehat{\mathbf{V}}_i^T \right)^2 + \lambda \cdot \#\mathcal{I}_t(u) \|\mathbf{U}\|$;

4 $\forall i \in l_{items}$, $\widehat{\mathbf{V}}_i \leftarrow \text{argmin}_{\mathbf{V}} \sum_{u \in \mathcal{J}(i)} \left(r_{u,i} - \widehat{\mathbf{U}}_u \mathbf{V}^T \right)^2 + \lambda \cdot \#\mathcal{J}_t(i) \|\mathbf{V}\|$;

mBALS-WR is designed to work in a sequential context where the matrix decomposition slightly changes between two consecutive calls. As a consequence, there are three main differences between ALS-WR and mBALS-WR. First, instead of computing $\widehat{\mathbf{U}}$ and $\widehat{\mathbf{V}}$ from scratch, mBALS-WR updates both matrices. Second, mBALS-WR performs only one pass on the data. And third, mBALS-WR updates only a fixed percentage of the line of $\widehat{\mathbf{U}}$ and $\widehat{\mathbf{V}}$, decided by the parameter p. This parameter is in the range from 0 to 1, and represents the proportion of the matrix which should be updated. When the parameter p is set to 1, mBALS-WR is a one-pass ALS-WR.

The main advantage of mBALS-WR is in spreading the computing budget along time steps which means $\widehat{\mathbf{U}}$ and $\widehat{\mathbf{V}}$ are more often up to date. On one hand, ALS-WR consumes a huge computing budget every thousands of time steps; in between two updates, it selects the items to recommend based on an outdated decomposition. On the other hand, mBALS-WR makes frequent updates of the decomposition. In the extreme case, updates can be done at each time step.

For mBALS-WR, updating each user vector $\mathbf{U}_u$ costs $O(|\mathcal{I}(u)|k^2 + k^3)$ and updating each item vector $\mathbf{V}_i$ costs $O(|\mathcal{J}(i)|k^2 + k^3)$, but we update only a given proportion of the total number of users and items and do a single iteration. For each call of mBALS-WR, the total complexity is $O(|\mathcal{S}|k^2 + p(N+M)k^3)$.

For a total number of recommendations T, the complexity for calling mBALS-WR in SeALS would be $O(\frac{T}{T_u}(|\mathcal{S}|k^2 + p(N+M)k^3))$, as mBALS-WR is called every T_u time steps, but either T_u is typically large with a high value of p (the update happens rarely, but for a large portion of the users

and items), or T_u is smaller but with a smaller value of p (the model is updated often for a small part of the users and items). In either case, this makes the overall complexity lower. In comparison, PTS has a complexity at each time step of $O(((\hat{N} + \hat{M})k^2 + k^3)D)$, where $\hat{N}$ and $\hat{M}$ are the maximum number of users who have rated the same item and the maximum number of items rated by the same user respectively, and where D is the number of particles of the particles filter. This complexity also has one term in k^2 and one in k^3, but compared to SeALS, the update is done at every time step which gives an overall high complexity.

Finally, no analysis about the regret for our algorithm is provided yet. However, we want to emphasize that this a difficult task and that, at current time, no existing work on bandits and RS manages to prove regret bounds on their algorithms. The only proof available for bandits applied to RS is the one of PTS (Kawale et al., 2015), but the regret bound only holds in a specific case when the rank of the matrix is supposed to be $k = 1$, which is a very simplistic and non-realistic setting for RS (for example, in this case, the highest-rated item in expectation is the same for all users).

3.5 Experimental Investigation

In this section, we empirically evaluate the algorithms in the sequential setting on large real-world datasets. Ideally, an online evaluation would be the best choice to observe the sequential aspect of RS, but since we do not have access to a real application, we build here our experimental setting for datasets offline. Two series of experiments emphasize two different aspects of the sequential RS: Section 3.5.3 shows that exploration improves the quality of the RS model and compares our model with several baselines, while Section 3.5.4 focuses on the influence of the method updating the matrix model on the pseudo cumulative regret and the running time.

3.5.1 Experimental Setting and Remarks

We use the same setting as the one used in the paper by Kawale et al. (Kawale et al., 2015). For each dataset, we start with an empty matrix $\mathbf{R}$ to simulate an extreme cold-start scenario where no information is available at all. Then, for a given number of time steps, we loop on the following procedure:

1. we select a user u_t uniformly at random,
2. the algorithm chooses an item i_t to recommend,
3. we reveal the value of $r_t = r^*_{u_t,i_t}$ and increment the pseudo-regret score as in Equation (3.2).

We assume that $\mathbf{R}^*$ corresponds to the values in the dataset. To compute the regret, the maximization term $\max_i r^*_{u_t,i}$ is taken w.r.t. the known values in the full matrix (selected among $i \in \mathcal{I}^*(u)$, where $\mathcal{I}^*(u)$ represents the set of all items for which the rating $r_{u,i}$ from user u is in the dataset). We do not put any constraint about items already recommended to a user: it is allowed

to play an arm several times, and an item already recommended in previous iterations is not discarded from the set of future possible recommendations. In other words, an item can be recommended several times to the same user. The goal for all algorithms is to minimize the cumulative regret $\mathcal{R}_T$ defined in Equation (3.2), for any time step t.

Datasets

We consider five real-world datasets for our experiments. Except Movielens1M, these datasets are among the largest ones available to evaluate RS, and items are as various as movies, artists or even people to date.

1. **Movielens1M** (Harper and Konstan, 2015): this dataset was collected by GroupLens Research and contains ratings on movies from MovieLens users who joined MovieLens in 2000. This dataset was released in February 2003 and has since been widely used in most of the research papers dealing with RS which use and offline evaluation. All users in the dataset have at least 20 ratings on movies. Note that some additional information is available, such as demographic data about the users, as well as the title and genre information about the movies. We do not integrate here any of this information and only use the ratings data.

2. **Movielens20M** (Harper and Konstan, 2015): these data were also created by Grouplens Research group, and extracted from ratings on movies by Movielens users between January 09, 1995 and March 31, 2015. This dataset was generated on March 31, 2015, and is the most recent and largest stable Movielens dataset. All selected users also have rated at least 20 movies. Contrary to Movielens1M, no demographic information about users is included, but tagging information made freely by users on movies is available. Notice that the rating scale of Movielens changed through time: the rating scale at the release was an integer scale of 1-5 and changed from a half-integer from 0.5 to 5. As the range of the time at which ratings were collected goes from 1995 to 2015, ratings from the old scale notation and the new ones are mixed in this dataset.

3. **LibimSeTi** (Brozovsky and Petricek, 2007): this dataset was extracted from the LibimSeTi dating website by Charles University. This is an interesting dataset as "items" in this case are other users of the website. Once again, only users who provided at least 20 ratings were included. Moreover, users who provided constant ratings were excluded. Note that ratings were originally on an integer 1-10 scale, but we scaled them to a (float) 1-5 scale. The gender information about users was included in the dataset but is not used in our experiments.

4. **Douban** (Ma et al., 2011): this is also a dataset about ratings on movies extracted from the Douban website, which is a Chinese Web 2.0 website providing user review and recommendation services for movies, books, and music. This dataset also contains the social friend network as this network was used in the paper made by researchers who released this dataset.

5. **Yahoo! Music user ratings of musicalartists**[4]: this dataset is provided by the Yahoo! Research Alliance Webscope program. It represents a sample of (anonymized) Yahoo! users' ratings of musical artists, gathered over a thirty-day period sometime prior to March 2004. The ratings are integers ranging from 0 to 100. There is an exception with the score of 255 which indicates the user noticed the system "never play again" about the recommended artist. Note that the Yahoo! dataset used in our experiments is taken from the originally larger dataset from which we extracted users with at least 20 ratings. We also rescaled ratings from a 0-100 integer scale to a 1-5 scale (with floating points), and assigned a score of 0.5 when the rating was the 255 value.

Some basic characteristics about the dataset as we use them in our experiments are reported in Table 3.1. The statistics about mean rating and standard deviation of the ratings are calculated from ratings after rescaling for LibimSeTi and Yahoo! dataset.

TABLE 3.1: Characteristics of the five datasets used for experiments on sequential recommendation using MAB.

	Movielens1M$^\circ$	Movielens20M$^\circ$	LibimSeTi $^{\circ *}$	Douban	Yahoo! $^{\circ *\dagger}$
#users	6,040	138,493	135,359	129,490	1,065,258
#items	3,706	26,744	168,791	58,541	98,209
#ratings	1,000,209	20,000,263	17,359,346	16,830,839	109,485,914
Density	4.47%	0.54%	0.076%	0.22%	0.1%
Rating scale	1-5	0.5-5	1-5	1-5	0.5-5
#ratings/user	165.6	144.4	128.2	130	102.8
#ratings/item	269.9	747.8	102.9	287.5	1114.8
Mean rating	3.58	3.53	3.19	3.84	3.03
STD of ratings	1.12	1.05	1.38	0.91	1.61

$\circ$ at least 20 ratings $*$ rescaled $\dagger$ filtered

Some difficulties arise when using real-world datasets: in most cases, the ground truth is unknown, and only a very small fraction of ratings is known, since users only gave ratings to items they have purchased (or listened to or watched... depending on the type of RS). This makes the evaluation of algorithms uneasy considering we need in advance the reward of items we include in the list of possible recommendations. This is the case in our experiments, as we do not have access to the full matrix $\mathbf{R}$ in all datasets.

This issue is solved in the case of contextual bandits by using reject sampling (Li et al., 2011): the algorithm chooses an arm (item), and if the arm does not appear in logged data, the choice is discarded, as if the algorithm had not been called at all. For a well collected dataset, this estimator has no bias and has a known bound on the decrease of the error rate (Langford et al., 2008). With our setting, we need no more to rely on reject sampling: we restrict the possible choices for a user at time step t to the items with a known rating in the dataset.

[4]https://webscope.sandbox.yahoo.com/

3.5.2 Baselines

The SeALS algorithm is compared to the following baselines:

Random At each iteration, a random item is recommended to the user (chosen only among the ones rated by the user in the full dataset).

Popular This approach assumes we know the most popular items based on the ground truth matrix. At each iteration, the most popular item (restricted to the items rated by the user on the dataset) is recommended. Remark that this is a strong "oracle" baseline as it knows beforehand which items have the highest average ratings. However, it is not an optimal strategy as it does not perform any personalization for users: recommended items are based only on the average popular preferences.

UCB1 This bandit approach considers each reward r_{u_t,i_t} as an independent realization of a distribution ν_{i_t}. In other words, it recommends an item without taking into account the identity of the user requesting the recommendation. The average rating $\mu_i(t)$ is computed from all ratings given on the item i by any user until t, and the number of times the item was seen represented by $T_i(t)$ also denotes how many times the item was recommended in total to any user.

PTS (Kawale et al., 2015) This approach builds upon a statistical model of the matrix $\mathbf{R}$, using the Probabilistic Matrix Factorization model (Salakhutdinov and Mnih, 2008b). The recommendations are done after a Thompson Sampling strategy (Chapelle and Li, 2011) which is implemented with a Particle Filter. We present the results obtained with the non-Bayesian version, as it obtains very similar results with the Bayesian one.

Greedy This approach is SeALS built with $\alpha = 0$ which implies no exploration is done during the recommendation process. This is the approach traditionally used in RS, where no Multi-Armed Bandits is integrated into the recommendation phase.

Remark that, despite what one could think at first sight, the UCB1 approach which eventually will recommend the items with the highest estimated average rating does not necessarily converges to the same solution as the Popular baseline, as the average rating it estimates can be different. The interested reader can see an example in Appendix A.

3.5.3 Impact of Exploration

The first set of experiments compares two strategies to recommend an item: SeALS with $\alpha > 0$, and SeALS with $\alpha = 0$ (denoted Greedy) which corresponds to the greedy strategy. As this experiment does not target the study of the update parameter, both strategies use the maximum possible value for this parameter p ($p = 1$), which means we update all the users and items every T_u time steps.

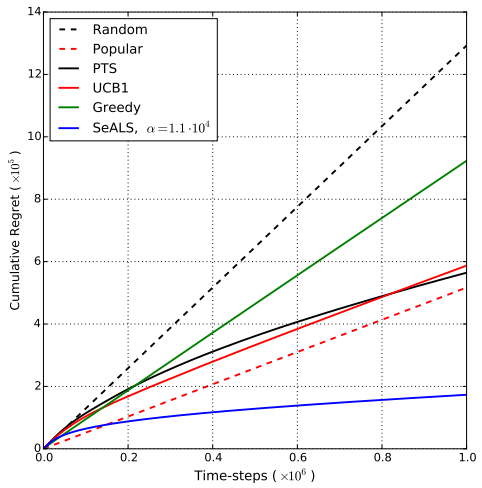
The values of T_u used are the same for SeALS and Greedy, and T_u was set to 1,000 for Movielens1M, 10,000 for Douban and Movielens20M and 250,000 for Yahoo!. These values of T_u are chosen since they allow the update to happen often enough to obtain good results, and since setting a higher period of update would not give much better results (cf. Section 3.5.4). We set the regularization parameter $\lambda = 0.1$ for Greedy and $\lambda = 0.15$ for SeALS. Parameter k is set to 15. These values are chosen from a grid search during experiments as they give high stable results on all datasets.

We also compare these two approaches with PTS. By fixing the value of the parameters of PTS as mentioned in (Kawale et al., 2015) (30 particles and $k = 2$), the experimental results we obtain are not as good as the ones presented in that paper. We increase k to 15 to be on par with the number of features used in SeALS, and we obtain results more similar to the ones presented in (Kawale et al., 2015). We use that value of k for all results of the PTS approach we are displaying.

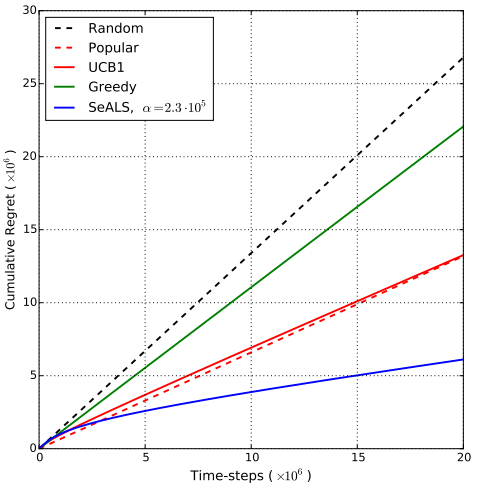
Figure 3.2 displays the pseudo-regret $\mathcal{R}_T$ obtained by the Recommendation System after a given number of iterations (all results are averaged over 100 runs). Results on all datasets demonstrate the need of exploration during the recommendation process: by properly fixing α, SeALS gets lower pseudo-regret value than Greedy. SeALS also obtains the best results on all datasets among algorithms which does not know the ground truth (contrary to Popular). Other methods aiming at tackling the exploration vs. exploitation dilemma, like PTS or UCB1 (which does not even provide personalized recommendations) reach a lower cumulative regret than Greedy, which highlights the importance of the exploration.

Notice that results on LibimSeti indicates a very low cumulative regret for all algorithms except Random. Popular approach in particular reaches the best results with an extremely low cumulative regret: this indicates that "items" (in this case, people to date) in the dataset are likely to match the tastes of every user, and can be recommended to anyone. Thus, detecting these users is sufficient to perform good recommendations. Exploration is still necessary to detect them more quickly: in a case where the regret of Popular is close to the optimal, then UCB1 is the most adapted algorithm as it also aims at finding the best items in average over all users by exploring them. However, note also that the goal of the RS in the dating site would be to find a couple of people who match well mutually (in our case, this would mean the "user" also have to match the tastes of the "item"), which is a more complex task than the one we tackle.

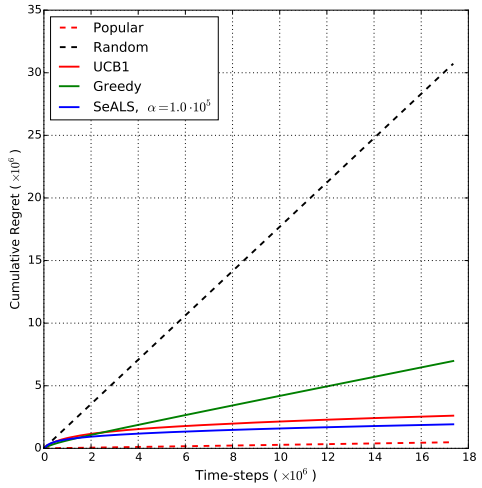
The PTS method which tackles the exploration/exploitation dilemma appears only on the Movielens1M evaluation as this method does not scale well on large datasets (cf. Section 3.5.4 for the running time of PTS on Movielens1M). However, it is important to note that on the original PTS paper, this approach performs only comparably or slightly better than the Popular baseline on the evaluation provided on all small datasets, while our approach consistently performs much better than this baseline. One can reasonably assume that SeALS would perform better than PTS even if the latter one didn't have a scaling issue.



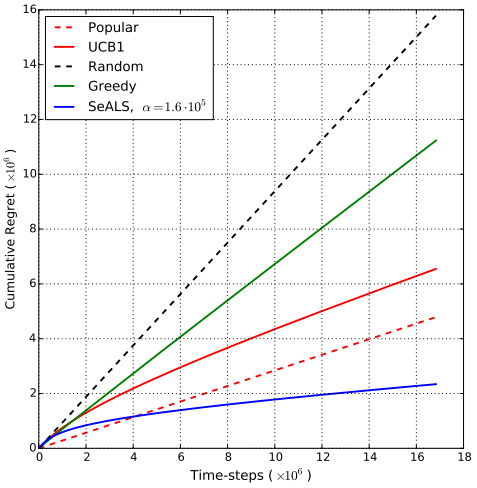
(A) Movielens1M



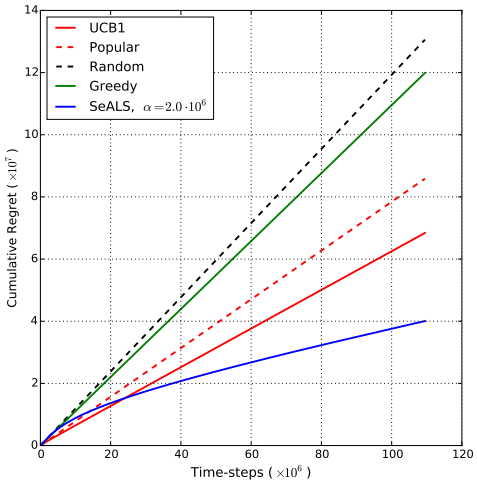
(B) Movielens20M



(C) LibimSeti



(D) Douban



(E) Yahoo

FIGURE 3.2: Impact of exploration on the cumulative regret score, evaluated on five datasets.

3.5.4 Impact of the Update Strategy

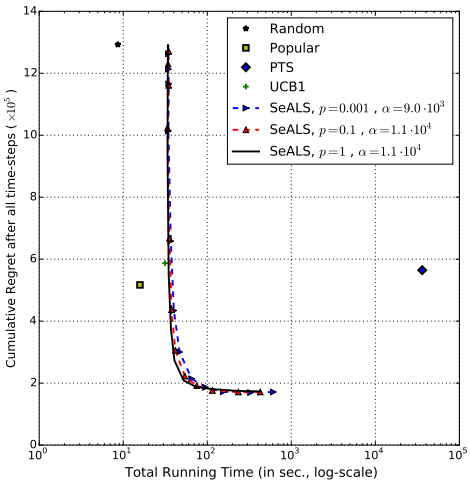
To evaluate the impact of the method used to update the model as well as the period at which updates take place, we set up a different evaluation display for this second experiment. For each RS, we run experiments as presented in Section 3.5.1, and we store the final running time as well as the pseudo-regret at the end of all iterations. Such evaluation methodology allows finding which size of mini-batch and update period leads to the best trade-off between the final pseudo-regret score and the total running time. This is an important point as a recommendation should both (i) be accurate and (ii) be quickly provided to the user in a real-word RS.

Figure 3.3 displays the results of this experiment (averaged over 100 runs). Each curve corresponds to a fixed size of mini-batch p, and every point of a same curve represents a specific value of the update period T_u. A point located at a high running time results from a small value of T_u (meaning the model was updated very often). Note that for SeALS, we display the results for each p (*i.e.* each curve) with the value of α reaching the lowest pseudo cumulative regret. Thus, each curve can have a different value of α. Note that in theory, each point of each curve could also potentially reach its best score with a different value of α, but it would make it hard to read the figure. We also display the results of other baselines for which we cannot control the running time such as Random or UCB1, and exhibit these results with a single point in the figure.

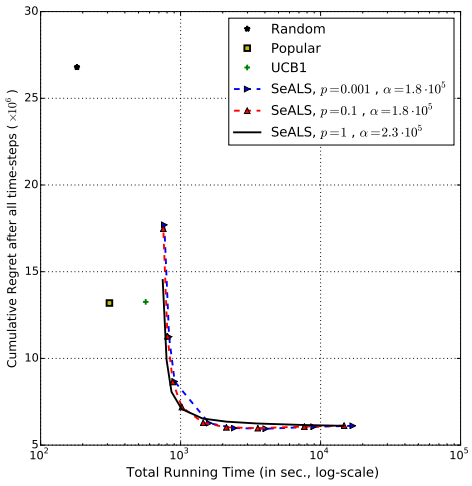
For SeALS with $p = 1$, the period T_u of the updates varies in the range $[1\,10^3; 2\,10^5]$ for Movielens1M, $[10^4; 5\,10^6]$ for Douban and Movielens20M, and $[2.5\,10^5; 10^7]$ for Yahoo!. For $p = 0.1$ and $p = 0.001$, the considered periods are the same ones as for $p = 1$, but respectively divided by 10 and 10^3 in order to obtain comparable running times. Indeed, since we update a smaller portion of the matrix, it is possible to run this update more often and choose a small period T_u. For each value of p, we display the curve with the value of α (for the exploration) which reaches the lowest pseudo-regret.

Three main conclusions are drawn from this experiment:

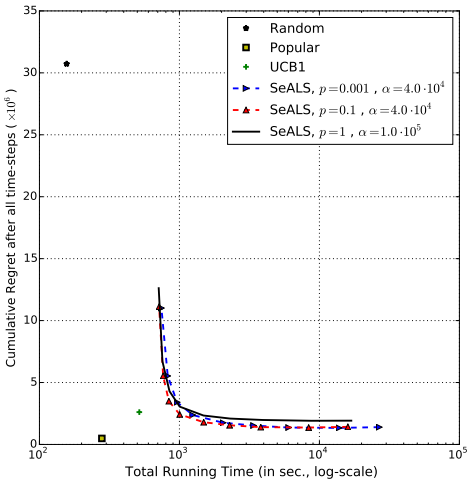
- The results on Movielens1M highlight the non-scalability of the PTS algorithm, which takes several hours to complete 1 million iterations while SeALS only takes a few minutes. PTS does not seem to be an appropriate algorithm to provide quick recommendations as it takes too long updating the model.
- On every dataset, the curve representing SeALS quickly decreases: there is a rapid transition from a poor score to a good pseudo-regret. This means finding the appropriate period of update is sufficient to obtain a good RS.
- Third, on the large datasets, decreasing the portion of the users and items to update with a smaller p results in a worse score when using large update periods, but leads to a slightly better trade-off between the pseudo-regret and the running time at some point, when the updates are happening more often. One has to notice the best results for smaller values of p are obtained with a slightly smaller value of α, which implies that less steps of exploration have been done during



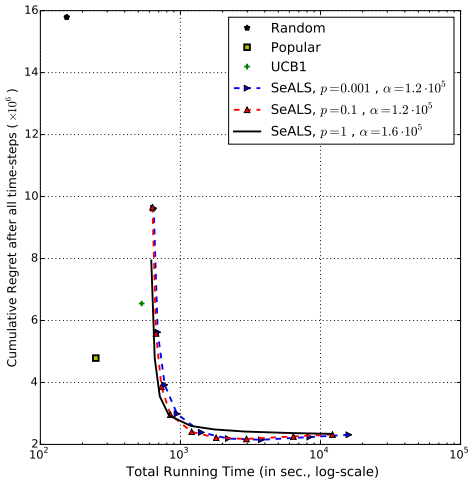
(A) Movielens1M



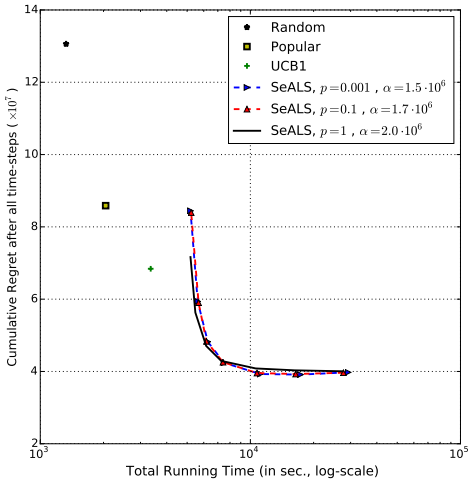
(B) Movielens20M



(C) LibimSeti



(D) Douban



(E) Yahoo

FIGURE 3.3: Impact of the update strategy on the final cumulative regret score, evaluated on five datasets.

the experiments. It is likely that some sort of exploration is added in the system by not updating every user and every item at each time step as it happens when $p = 1$ is chosen: while the whole model is not updated, it does not play optimally, which means it explores.

Finally, we want to highlight that UCB1 is a reasonable approach to apply in a RS if only a small time budget can be allowed, as it obtains good results on most datasets in a very small running time.

3.6 Concluding Remarks

In this chapter, we handle Recommender Systems based on Matrix Completion in the suitable context: an endless loop which alternates (i) learning of the model and (ii) recommendations given the model. Our proposed approach, SeALS, meets both challenges which arise in such a context. First, SeALS handles both short-term and long-term reward by balancing exploration and exploitation. Second, SeALS handles constraints on computing budget by adapting mini-batch strategy to alternating least square optimization. Experiments on real-life datasets show that (i) exploration is a necessary evil to acquire information and eventually improve the performance of the RS, and (ii) SeALS runs in an acceptable amount of time (less than a millisecond per iteration).

However, in this setting, we only focused on the recommendation of one item, where the feedback is directly given by the user. In a real-world RS, the user usually has a list of items proposed to him, where he can decide to pick (or not) an item and gives feedback on it. We study this setting in the next chapter of this thesis.

Chapter 4

Ranking Using (No-)Click Implicit Feedback

In this chapter [1], we study a different recommendation setting. The previous chapter introduced a RS aiming at handling sequential recommendation, where one item is recommended at a time and an explicit feedback is received in the form of a rating. We study in this chapter an extended setting where, at each time step, multiple items are recommended to the user in the form of an ordered list of items. In this case, the feedback received by the RS is both implicit and explicit. We describe a specific setting modeling the interaction between the user and the RS, and propose two approaches using the two sorts of feedback to improve faster the quality of the recommendation list. Our approach is evaluated on real-world datasets.

Contents

[1] This chapter is an extended version of a publication in ICONIP (Guillou et al., 2016c)

4.1 Sequential Recommendation of Multiple Items

In most real-world Recommendation Systems, the system does not want to recommend only one item as described in Chapter 3, since there is a high risk to disappoint the user if this item does not suit his tastes. Moreover, the explicit feedback on a recommended item is not always collected as the user has the choice to rate this item or not.

For these reasons, the system wants to let the user choose which item(s) he wants to buy/read/listen to..., and thus most commercial RS recommend an ordered list of items in which the user can pick one or several items.

In this chapter, we focus on such RS, which aim to recommend a list of items. Standard approaches in that setting discuss which loss is the most adapted and how to optimize that loss (Weimer et al., 2008; Balakrishnan and Chopra, 2012; Shi et al., 2013). In their attempts to do so, they omit one effect deriving from recommending a list of items: we could expect more feedback than a rating on a single item. For example, we could have access to the list of items which have been seen by the user, the list of items on which he clicked, etc. In a few words, we are confronted to a mix of explicit and implicit feedback. Despite its more acute noisiness compared to explicit feedback, implicit feedback can be useful for the system to learn about the user's preferences: for example the RS can use the implicit information when the user does not select any of the items recommended in the list to improve its model even if no rating is explicitly given by the user.

Several RS approaches have already been developed to handle implicit feedback (Rendle et al., 2009; Shi et al., 2012; Takács and Tikk, 2012), but surprisingly, none is considering negative implicit feedback: "the user u did not click on item j". Most of the time, the implicit feedback is limited to unary rating, such as "the user put the item in the purchasing basket" or "the user listened to one song", but not to "negative" implicit feedback.

The feedback gathered depends on the model of interaction for users. The first contribution in this chapter is to propose two approaches to handle the feedback involved by the interaction model discussed in (Chapelle et al., 2009). These two approaches lead to relevant recommendation lists for three real-life datasets.

The second contribution of our work relates to the experimental setting. The data used by a RS depend on previous recommendations. This means two different RS should recommend based on two different sets of feedback. As a consequence, the evaluation of the RS traditionally done in a fixed, batch setting does not make sense. In this part, we propose a proper experimental setting which mimics the interaction between the RS and the users. This evaluation setting extend the one used in Chapter 3, and we represent the sequential loop of recommendation for this chapter in Figure 4.1, which can be compared to Figure 3.1 from previous chapter to assess the difference.

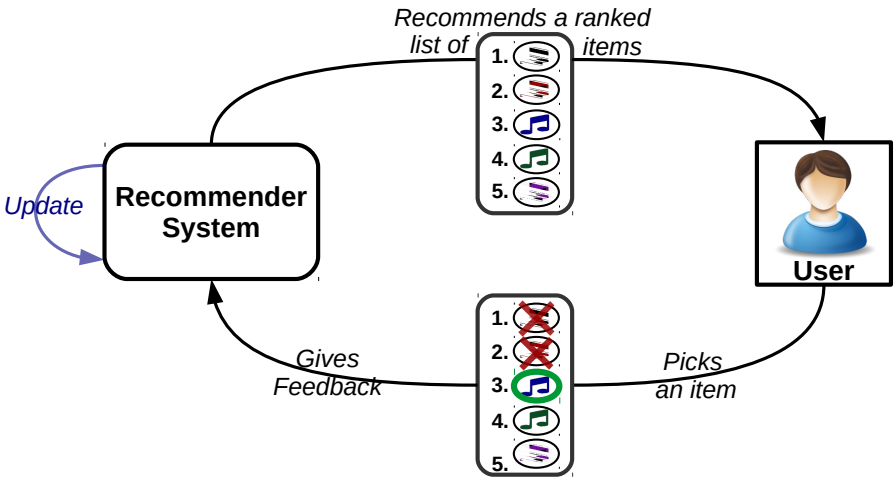


FIGURE 4.1: The sequential recommendation process with a list of items. After the RS recommends the item, the user looks at the items one by one and picks one, then gives back a feedback about the item picked, and the RS possibly update its model.

4.2 Related Work

In this section, we briefly review related work in RS about several aspects included in our approach. The first aspect is related to Online Ranking in Information Retrieval, the second one is about ranking methods applied in the context of RS, while the last aspect is the mixing of both explicit and implicit feedback.

4.2.1 Online Ranking in Information Retrieval

A field in Information Retrieval focuses on online Learning to Rank, where the goal is for a web search engine to provide a good ranking of documents and improve the performance of the system as the user chooses documents or not and gives feedback. In all these researches, only the implicit feedback given by the user's click (when he chooses one or several documents) is used in the model to improve the ranking, since in web search no explicit feedback is given by user about the web page or the document. In (Radlinski et al., 2008), a Multi-Armed Bandit (MAB) approach is used to provide diverse ranking, by running a MAB instance for each position in the ranked list. Each MAB instance is treated independently and only the MAB instance corresponding to the documents clicked by the user are updated.

The modelization of the problem as a Dueling Bandits Problem is introduced in (Yue and Joachims, 2009), where the only actions are comparisons between two points (in the case of web search, the comparison refers to the portion of users who preferred results provided by a retrieval function w over those of another function w'). The authors define a regret accordingly

and propose a Gradient Descent method with theoretical guarantees about this regret.

Finally, (Hofmann et al., 2013b) extends pairwise or listwise learning by adding mechanism to balance exploration-exploitation. They both use an exploitative list, predicted by a Stochastic Gradient Descent approach, and an exploratory list with elements selected at random, and make the final ranking of documents by using an idea derived from ε-greedy where each element of the final list is picked with a probability ε from the exploratory list or otherwise from the exploitative list.

The field of Information Retrieval and the field of RS is tightly related, and we also study here the problem as a sequential process with click made on items of the list. However, compared to these methods, the RS used in our setting receives both explicit and implicit feedback, and we do not integrate the exploration-exploitation aspect in our approaches.

4.2.2 Recommendation with Ranking Approaches

Approaches recommending a list of items are looking at a the best loss function. These approaches replace the squared loss on ratings by a loss measuring the ranking ability of the model learned.

Some methods of ranking focus only a rating matrix filled with implicit binary feedback, such as CliMF in (Shi et al., 2012) which is a variant of latent factor Collorative Filtering, optimizing a lower bound of the smoothed reciprocal rank of "relevant" items in ranked recommendation lists. Bayesian Personalized Ranking (BPR) in (Rendle et al., 2009) provides an optimization criterion BPR-Opt which is the maximum posterior estimator derived from a Bayesian analysis of the pairwise ranking problem, and proposes an algorithm based on Stochastic Gradient Descent to optimize this criterion. Finally, RankALS presented in (Takács and Tikk, 2012) introduces an Alternating Least Square approach where the objective function is a ranking function using pairwise preferences.

Other approaches have been built to handle the explicit feedback with multiple level of relevancy. CoFiRank is one of the first approach handling this type of ratings and was introduced in (Weimer et al., 2008). It is based on Maximum Margin Matrix Factorization (a matrix factorization technique with a trace norm regularization on the factors), and it optimizes various losses including a smooth approximation of the *Normalized Discounted Cumulative Gain* (NDCG).

Collaborative ranking approach described in (Balakrishnan and Chopra, 2012) first learns user and item features using Probabilistic Matrix Factorization. From these features, the authors can apply Learning to Rank pointwise and pairwise methods such as the ones described in (Burges, 2010) which needs input features. A very similar approach is used in (Volkovs and Zemel, 2012), where a Learning To Rank algorithm is applied after a PMF matrix factorization step, but this paper adds one more step of feature construction by using neighborhood approach to reduce the parameter

space. (Shi et al., 2013) is an extension of CliMF to handle ratings with multiple level of relevance and optimizes a smooth approximations of the *Expected Reciprocal Rank* (ERR). Finally, more recent work in (Liu and Aberer, 2014), and (Lee et al., 2014) respectively optimize a pairwise learning to rank loss, and a structured output loss.

These approaches only handle either explicit or implicit feedback. However, when recommending a list of items, a mix of both kinds of feedback is often to be expected: the rating of selected items (explicit), the list of items which have not been clicked despite having been shown to the user (implicit), the list of items which have been selected but not purchased, etc.

4.2.3 Mixing Explicit and Implicit Data

Only few approaches have attempted to mix explicit and implicit feedback for Collaborative Filtering approaches. The most famous approach among these approach is SVD++ (Koren, 2008). It integrates implicit feedback in the representation of a user and limits itself to a restrictive kind of implicit feedback: a list of clicked items. The model of the SVD++ is described by Equation (4.1), giving the rating $\hat{r}_{u,i}$ given by user u on item i:

$$\hat{r}_{u,i} = \mu + b_u + b_i + \left(p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j \right)^T q_i \,, \tag{4.1}$$

where μ, b_u and b_i are respectively the general, the user and the item bias, $|N(u)|$ is the set of implicit information (items rated by the user u), y_j is a factor vector to represent the implicit part, and p_u and q_i represent the user and item features.

Another approach, Co-Rating (Liu et al., 2010), considers implicit feedback as complementary values to fit. As such, Co-Rating manages any kind of implicit data. Still, it assumes that explicit and implicit feedback equal themselves as soon as they are scaled to $[0, 1]$. SVD++ and Co-Rating both learn by optimizing a squared loss.

In the remainder of the chapter, we develop an approach which properly handles any kind of feedback, and therefore leads to better experimental results.

4.3 Ranking Recommender System Using Click Feedback

Before building the model to handle both explicit and implicit feedback, we have to express and define the model of interaction we will use in what follows. This model governs the feedback which is gathered, and is used to build our second approach (in Section 4.3.3).

4.3.1 Setting

The recommendation setting we consider is the one described in the paper from Chapelle et al. (Chapelle et al., 2009). Let us consider N users and M items and the unknown matrix $\mathbf{R}^*$ of size $N \times M$ such that $r^*_{u,i}$ is the rating of user u with regard to item i. We assume the ratings range from 0 to R.

From $\mathbf{R}^*$, a relevance probability $p(u,i)$ is derived, and represents how much a user u is eager to select an item i:

$$p(u,i) = \frac{2^{r^*_{u,i}} - 1}{2^R}\,. \tag{4.2}$$

Then we set up the interaction between the RS, the users and the items. At each time-step t, a user u_t requests ℓ items, and the RS provides the ranked list $L^{(t)} = (i_1^{(t)}, \ldots, i_\ell^{(t)})$. As defined in the Cascading model, the interaction model supposes several characteristics:

- The user observes the list from top to bottom, one item after another;
- While observing the s-ith item, the user has a chance to pick it with probability $p(u_t, i_s^{(t)})$;
- Once an item is picked (denoted i_t), the user stops observing the list. Thus, note that the user does not observe items ranked after the item picked.

In our interaction model, the explicit feedback is collected at the time t for the item i_t which is picked by the user u_t in the list, as the rating $r^*_{u_t,i_t}$

We define the function $ERRClickModel(u_t, L^{(t)})$ which takes as input a user and a ranked list of items, and is described in Algorithm 9. Following the setting specified above, it returns the position of the item clicked, or $\ell + 1$ if no item was clicked.

Algorithm 9: ERRClickModel: models the interaction between a user and the RS presenting a ranked list of items.

Input : A user: u,
A list of items of size ℓ: $L = (i_1, \ldots, i_\ell)$,
The maximum possible rating R
Output: The position of the item clicked: $p_{clicked}$

$p_{clicked} \leftarrow \ell + 1$;
$s \leftarrow 1$;
while $s \leq \ell$ and $p_{clicked} = \ell + 1$ **do**
  $p(u, i_s) = \frac{2^{r^*_{u,i_s}} - 1}{2^R}$;
  if $P \sim U(0,1) < p(u, i_s)$ **then**
    $p_{clicked} \leftarrow s$;
  else
    $s \leftarrow s + 1$;
  end
end

This setting implies two types of feedback are received at every recommendation list displayed:

1. the **implicit feedback**: it consists in the list of skipped items (eventually empty if the first item in the list was picked), and the clicked item (eventually none if no click was made by the user during the recommendation step)
2. the **explicit feedback**: this is the rating of the clicked item. If no item was clicked, then this part of the feedback is retrieved.

Note that the rank (in the recommendation list) of the clicked item brings only information about items ranked above in the list: we know these items were skipped. On the contrary, no information is gathered about items placed below the clicked one, as the user did not observe them according to our setting.

The two following sections describe two different approaches attempting to use these types of feedback to improve recommendations.

In the following, we use shared notations, where:

- We denote $\mathbf{R}_t$ the partially known $N \times M$ matrix such that $r_{u_s,i_s} = r_s$ for any $s \leqslant t$.
- We note $\mathcal{S}_t$ the set of known entries of $\mathbf{R}_t$:

$$\mathcal{S}_t = \{(u_s, i_s); s \leq t\}\,.$$

- $\mathbf{A}_t$ denote the matrix of size $N \times M$ which stores for each user u and item i the number of clicks received by item i when presented to u, up to time t.
- $\mathbf{S}_t$ denote the matrices of size $N \times M$ which stores for each user u and item i the number of times i was skipped when shown to u on a recommendation list, up to time t.
- Finally, for any matrix $\mathbf{M}$, we denote $\mathbf{M}_i$ the vector corresponding to the i-th row of $\mathbf{M}$, and $\mathbf{M}_{i,j}$ correspond to the entry in the i-th row and j-th column.

For the sake of readability, the subscript t is omitted in the following.

4.3.2 Feature Engineering

The first method denoted **SVD+-** assumes that we can embed the implicit feedback into features of the model. Following the steps of SVD++, where the ratings is represented as in Equation (4.1) and the implicit feedback is modeled through the sum $\frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j$, we extend this implicit part as $\mathbf{C} \sum_{j \in N(u)} y_j$, where $\mathbf{C}$ consists in:

$$\mathbf{C}_{u,s} = \frac{\mathbf{A}_{u,s} - \mathbf{S}_{u,s}}{\sum_{s'=1}^{M} \mathbf{A}_{u,s'} + \mathbf{S}_{u,s'}}\,. \tag{4.3}$$

$\mathbf{C}$ gives a value to every item based on how many times it was clicked or skipped out of all interactions for a user. This value ranges from -1 to 1. The score in the numerator represents how much the item was clicked or skipped, while the denominator ponder this score by all the interactions made until now by the user. The denominator would grow for all items in $\mathbf{C}$. This helps to not penalize too much an item s which has been recommended in a list only one time and skipped at this first recommendation. The score $\mathbf{C}_{u,s}$ of this item would increase as time goes by, and it will eventually make its chance to be recommended high again.

A rating $r_{u,i}$ given by user u on item i is modeled by SVD+- as:

$$\hat{r}_{u,i} = \mu + b_u + b_i + \left(p_u + \mathbf{C} \sum_{j \in N(u)} y_j \right)^T q_i \,. \tag{4.4}$$

Notice that this representation does not generalize SVD++.

To learn our model, we build upon **Factorization Machine (FM)** in this way: following the data representation used for FM and described in the paper (Rendle, 2010), we associate to any user-item couple (u, i) a representation $\phi_1(u, i)$ and we look at a function f *s.t.* $f(\phi_1(u, i)) = r^*_{u,i}$. The representation of the data is the following:

$$\phi_1(u,i) = (\underbrace{0,\ldots,1,0,\ldots}_{N},\underbrace{0,\ldots,1,0,\ldots}_{M},\underbrace{\mathbf{C}_{u,1},\ldots,\mathbf{C}_{u,s},\ldots,\mathbf{C}_{u,M}}_{M}) \,, \tag{4.5}$$

where the first section indexes u, the second section indexes i, and the last one embed the implicit information gathered on items for the user u.

Our RS algorithm is called SVD+- as it uses both click and no-click feedback, and it takes as input $\mathbf{R}$ filled with known values, $\mathbf{A}$, $\mathbf{S}$, the set of items $\mathcal{S}$, and a size of recommendation list ℓ.

From that feature model, the Factorization Machine learns the function

$$\begin{aligned}\hat{f}_1(\phi_1(u,i)) = w_0 + w_u + w_i + v_u.v_i^T + \left(\sum_{s=1}^{M} C_{u,s} v_s\right) \cdot v_i^T \\ + v_u \cdot \left(\sum_{s=1}^{M} C_{u,i_s} v_s\right)^T + \sum_{s=1}^{M} C_{u,i_s} w_s \\ + \left(\sum_{s=1}^{M} C_{u,s} v_s\right) \cdot \left(\sum_{s=1}^{M} C_{u,s} v_s\right)^T ,\end{aligned}$$

where w_0, w_i, w_j and $(w_s)_{1 \leqslant s \leqslant M}$ are real values, and v_0, v_u, v_i and $(v_s)_{1 \leqslant s \leqslant M}$ are feature vectors of size k. These parameters are chosen to optimize a trade-off between (i) the average square loss with respect to known values in $\mathbf{R}^*$ and (ii) the L_2 norm of the parameters.

Note that optimizing the objective function $\hat{f}_1$ would not necessarily require to use Factorization Machine, but we choose this tool as it is both efficient and has shown good results in RS applications.

SVD+- is described in Algorithm 10. It takes as input a size of recommendation for the list and a period of update for the model, and the matrix of observed ratings. It provides recommendations sequentially to users of the RS.

Algorithm 10: SVD+-: sequentially recommends a list of items using click feedback.

Input : size of recommendation list: ℓ
period of update for the MF model: t_{update}
Input/Output: Matrix of observed ratings: $\mathbf{R}$,
Matrix containing the number of clicks: $\mathbf{A}$,
Matrix containing the number of no-clicks: $\mathbf{S}$,
Set of items rated: $\mathcal{S}$

1 **for** $t = 1,\ 2,\ \ldots$ **do**
2 get user u_t and set $\mathcal{A}_t$ of allowed items;
3 $\mathbf{C}_{u_t} = \frac{\mathbf{A}_{u_t} - \mathbf{S}_{u_t}}{\sum_{i \in \{1..M\}} \mathbf{A}_{u_t,i} + \mathbf{S}_{u_t,i}}$;
4 **for** $p = 1 \ldots \ell$ **do**
5 fill the list of items at position p:
 $L^{(t)}[p] \leftarrow \text{argmax}_{i \in \mathcal{A}_t} \hat{f}_1(\phi_1(u_t, i))$;
6 $\mathcal{A}_t \leftarrow \mathcal{A}_t \setminus \{L^{(t)}[p]\}$;
7 **end**
8 Simulate user click: $p_{clicked} \leftarrow ClickModel(u_t, L^{(t)})$;
9 **for** $p = 1 \ldots (p_{clicked} - 1)$ **do**
10 $\mathbf{S}_{u_t, L^{(t)}[p]} \leftarrow \mathbf{S}_{u_t, L^{(t)}[p]} + 1$;
11 **end**
12 **if** $p\_clicked \leq \ell$ **then**
13 the item clicked is: $i_t \leftarrow L^{(t)}[p_{clicked}]$;
14 $\mathbf{A}_{u_t,i_t} \leftarrow \mathbf{A}_{u_t,i_t} + 1$;
15 recommend i_t and receive rating $r_t = r_{u_t,i_t}$;
16 update $\mathbf{R}$ and $\mathcal{S}$;
17 **end**
18 **if** $t \equiv 0 \pmod{t_{update}}$ **then**
19 update model based on $\mathbf{R}$ (input is triplet $(u, i, \mathbf{C}_u)$);
20 **end**
21 **end**

From the lines 4 to 7, the algorithm fills the list of items to recommend according to the predictions of the FM. Then, in line 8, the user clicks on one item and we get the position of this item in the list.

Lines 9 to 11 represent the update of the matrix $\mathbf{S}$ for items which were ranked higher than the clicked item, while lines 12 to 17 represent the update of the matrix $\mathbf{A}$ for the clicked item, its recommendation to the user and the reception of the explicit rating. Finally, lines 18 to 20 show the update of the model at some specific time.

4.3.3 Dual Matrix Factorization

From another perspective, we design a second approach called **DualMF**, which considers both types of feedback as values to fit. More specifically, we look at a low rank approximation $\widehat{\mathbf{R}} = \mathbf{U}.\mathbf{V}^T$ of $\mathbf{R}^*$, where $\mathbf{U}$ and $\mathbf{V}$ are matrices of respective sizes $N \times k$ and $M \times k$. Obviously, we want $\widehat{\mathbf{R}}$ to fit known values in $\mathbf{R}$ (aka. explicit feedback).

But we also make use of implicit feedback. Based on the model of the probability of click $p(u,i)$, we can build a biased estimator $\hat{r}_{u,i}^{imp}$ of $r_{u,i}^*$ for any user-item couple (u,i) *s.t.* item i has been clicked or skipped at least one time by user u:

$$\hat{r}_{u,i}^{imp} = \log_2 \left(1 + 2^R \frac{\mathbf{A}_{u,i} + 0.5}{\mathbf{A}_{u,i} + \mathbf{S}_{u,i} + 1}\right) . \tag{4.6}$$

The 0.5 added at the numerator and the 1 added at the denominator act similarly to a prior and help the model to not penalize too much items for which only little information has yet been collected. Implicit estimated ratings are stored in $\mathbf{R}_{imp}$ distinctively from explicit ratings.

Overall, the approach DualMF looks for a matrix $\widehat{\mathbf{R}}$ fitting both known values in $\mathbf{R}$, and $\mathbf{R}_{imp}$ values, and the objective function $\hat{f}_2$ of this problem is learned by solving the minimization problem:

$$\min_{\widehat{\mathbf{R}}=\mathbf{U}.\mathbf{V}^T} \mu \sum_{(u,i)\in\mathcal{S}} (\hat{r}_{u,i} - r_{u,i}^*)^2 + \sum_{(u,i):A_{u,i}+S_{u,i}\neq 0} (\hat{r}_{u,i} - \hat{r}_{u,i}^{imp})^2 + \lambda(\|\mathbf{U}\|^2 + \|\mathbf{V}\|^2) , \tag{4.7}$$

where μ and λ are non-negative real values. μ controls the impact of explicit data compared to implicit one, and λ weights the regularization term.

We choose again to use a Factorization Machine model to solve Equation (4.7). Compared to SVD+-, the representation of the data $\phi_2(u,i)$ does not include the information about implicit feedback, and the input is simply:

$$\phi_2(u,i) = (\underbrace{0,\ldots,1,0,\ldots,}_{N}\underbrace{0,\ldots,1,0,\ldots,}_{M}) . \tag{4.8}$$

The representation of the input is simpler, as input vectors do not contain any additional features based on $\mathbf{C}$. However more data points are used for training, as the model is trying to fit both the explicit rating $r(u,i)$ and the estimated rating from implicit feedback $\hat{r}_{u,i}^{imp}$.

The approach DualMF is described in Algorithm 11. Note that we use Factorization Machine algorithm to solve Equation (4.8) and learn the objective function $\hat{f}_2$, but it is only one possible way and other models could have been used instead. The algorithm is not too different from Algorithm 10 of SVD+-: the biggest difference is in line 11, where the implicit rating of $\mathbf{R}_{imp}$ is estimated, and on line 21 where the model uses the implicit estimated ratings to update.

Algorithm 11: DualMF: sequentially recommends a list of items using click feedback.

Input : size of recommendation list: ℓ
period of update for the MF model: t_{update}
Input/Output: Matrix of observed ratings: $\mathbf{R}$,
Matrix of implicit ratings: $\mathbf{R}_{imp}$,
Matrix containing the number of clicks: $\mathbf{A}$,
Matrix containing the number of no-clicks: $\mathbf{S}$,
Set of items rated: $\mathcal{S}$

for $t = 1,\ 2,\ \ldots$ **do**
  get user u_t and set $\mathcal{A}_t$ of allowed items;
  for $p = 1 \ldots \ell$ **do**
    fill the list of items at position p:
    $L^{(t)}[p] \leftarrow \text{argmax}_{i \in \mathcal{A}_t} \hat{f}_2(\phi(u_t, i))$;
    $\mathcal{A}_t \leftarrow \mathcal{A}_t \setminus \{L^{(t)}[p]\}$;
  end
  Simulate user click: $p_{clicked} \leftarrow ClickModel(u_t, L^{(t)})$;
  for $p = 1 \ldots (p_{clicked} - 1)$ **do**
    $i_p \leftarrow L^{(t)}[p]$;
    $\mathbf{S}_{u_t,i_p} \leftarrow \mathbf{S}_{u_t,L^{(t)}[p]} + 1$;
    $\hat{r}^{imp}_{u_t,i_p} = \log_2\left(1 + 2^R \frac{\mathbf{A}_{u_t,i_p}+0.5}{\mathbf{A}_{u_t,i_p}+\mathbf{S}_{u_t,i_p}+1}\right)$;
    update $\mathbf{R}_{imp}$;
  end
  if $p\_clicked \leq \ell$ **then**
    the item clicked is: $i_t \leftarrow L^{(t)}[p_{clicked}]$;
    $\mathbf{A}_{u_t,i_t} \leftarrow \mathbf{A}_{u_t,i_t} + 1$;
    recommend i_t and receive rating $r_t = r_{u_t,i_t}$;
    update $\mathbf{R}$ and $\mathcal{S}$;
  end
  if $t \equiv 0 \pmod{t_{update}}$ **then**
    update model based on $\mathbf{R}$ and $\mathbf{R}_{imp}$;
  end
end

4.4 Experimental Investigation with ERR Click Model

We empirically evaluate the two proposed algorithms in a sequential setting on three real-world datasets. For each dataset, we start with an empty matrix $\mathbf{R}$ to simulate an extreme cold-start scenario where no information is available at all. Then, a list of items is recommended for each user according to the following procedure:

1. we select a user i_t uniformly at random among possible users (the ones to which no recommendation list was displayed yet),
2. the algorithm chooses a list of 5 items to recommend,
3. the user observes the list, clicks or not on an item i_t according to the setting described in Section 3.1. The value of r_{u_t,i_t} is revealed if there was a click. The user is then discarded from possible users.

When all users have been shown a recommendation list once, we reintegrate all of them in the list of possible users, and loop on this procedure again, while keeping all feedback gathered until now. These steps are done up to 50 recommendations shown for every user. As the ground truth is unknown for every item (users only rated a small portion of all the items in the dataset), we restrict the possible choices for a user at each time-step to the items with a known rating in the dataset.

Note also that it is allowed to include an item in the list of recommendations for a user even if it has already been rated in the past by him. This is actually an essential point to our algorithms as value in $\mathbf{A}$ and $\mathbf{S}$ are updated depending on the number of times the item was accepted or rejected. This situation where items can be recommended several times is more likely to happen in music RS, where a song can be recommended several times, or in a video RS with short videos such as Youtube.

4.4.1 Evaluation Metrics

Metrics used for the evaluation are abandonment, ERR@5 and NDCG@5, which we recall now. Let us consider a user i, to which the list of items $i_1, \ldots, i_\ell$ has been recommended.

The abandonment metric represents the probability

$$\prod_{r=1}^{\ell} (1 - p(u, i_r)) \,,$$

for a user not to be satisfied by any item in the list (no click), and the RS should attempt to minimize it, since as few users as possible should end up with a non-satisfying list of items.

While the abandonment assesses that the algorithm suggests at least one item satisfying the user in the recommended list, the ERR and NDCG depict how adequate is the full list of items suggested by the RS regarding the user's tastes.

The ERR is based on the click probability defined by Eq. (4.2) and should be maximized:

$$ERR@\ell(u,(i_1,\ldots,i_\ell)) = \sum_{r=1}^{\ell} \frac{1}{r} p(u,i_r) \prod_{s=1}^{r-1} (1 - p(u,i_s)) \,. \tag{4.9}$$

The formula used for the Discounted Cumulative Gain at ℓ (Burges et al., 2005) is:

$$DCG@\ell(u,(i_1,\ldots,i_\ell)) = \sum_{r=1}^{\ell} \frac{2^{r^*_{u,i_r}} - 1}{\log_2(r+1)} \,. \tag{4.10}$$

The NDCG is the DCG divided by the best possible DCG and should be maximized. As we have no access to the ground-truth matrix $\mathbf{R}^*$ in our experiments, the NDCG is computed *w.r.t.* the known values in $\mathbf{R}$. This means that the ideal DCG for a user u is calculated based on items i such that $(u,i) \in \mathcal{S}$: the best l ratings of the user u in the full dataset are retrieved and ordered to find the best possible DCG.

4.4.2 Datasets

We consider three real-world datasets for our experiments:

1. **Movielens1M** (Harper and Konstan, 2015): this dataset is the same as the one described and used in the Chapter 3.
2. **Yahoo! Music user ratings of musical artists**[2]: the original dataset is the same as the one used in previous chapter, but we build this time a different subset. To build this subset, we first select the 10,000 most rated artists, and then the 50,000 users who rated the highest number of artists. We also rescale the ratings from the range 0-100 to 1-5 and assign the ratings with 255 score (meaning "do not recommend this ever again") to 0.5.
3. **Yahoo! Music ratings for User Selected and Randomly Selected songs**[3]: this dataset differs from the previous one as it contains ratings for songs (instead of artists), with data collected from two different sources. The first source consists of ratings supplied by users during normal interaction with Yahoo! Music services, and the second source consists of ratings for randomly selected songs collected during an online survey conducted by Yahoo! Research (participants were asked to rate 10 songs selected at random from a fixed set of 1,000 songs). 5,400 survey participants were randomly selected for inclusion in this data set on the condition that they had at least 10 existing ratings in the Yahoo! Music rating database restricted to the fixed set of 1,000 songs used in the survey. An additional set of 10,000 users was randomly selected for inclusion in the data set from among all non-survey participants with at least 10 existing ratings in the Yahoo! Music rating database. The dataset has around 300,000 user supplied ratings and exactly 54,000 ratings for randomly selected songs.

[2] https://webscope.sandbox.yahoo.com/
[3] https://webscope.sandbox.yahoo.com/

Characteristics of these three datasets are reported in Table 4.1, with the mean rating and the standard deviation being calculated for the rescaled version of the dataset for the Yahoo! artists dataset.

TABLE 4.1: Characteristics of the three datasets used for experiments on sequential ranking with (no)-click feedback.

	Movielens1M °	Yahoo! artists *†	Yahoo! songs
#users	6,040	50,000	15,400
#items	3,706	10,000	1,000
#ratings	1,000,209	32,997,016	365,703
Density	4.47%	6.60%	2.37%
Rating scale	1-5	0.5-5	1-5
#ratings / user	165.6	659.94	23.7
#ratings / item	269.9	3299.7	365.7
Mean rating	3.58	2.15	2.73
STD of ratings	1.12	1.46	1.56

∘ at least 20 ratings ∗ rescaled † filtered

Movielens1M is a standard dataset to compare RS approaches, so we included it in our experiments. However, it contains mostly high ratings and has a lower standard deviation. Since items to recommend are chosen only among the ones for which we have ratings, the Movielens1M dataset is less interesting from a ranking point of view because it is harder to distinguish between algorithms. On the other hand, the Yahoo! datasets contain a lot more low ratings, which allows for more realistic experiments, especially the Yahoo! songs dataset where some ratings for randomly selected songs are included in the matrix. Note that the two Yahoo! datasets also suit more to the experimental setting we use, as the replay is allowed (an item can be recommended to a user even if it has been rated before). This setting is closer to a real-world music RS where songs or artists can be recommended to a user several times, but it is not really desirable effect in a movie RS, as it is rarer users want to watch again movie they have already seen and rated.

4.4.3 Baselines

Our two approaches relate to two main aspects of a RS, evaluated in a sequential context: the ranking aspect and the combined use of implicit and explicit feedback. Thus, a lot of methods can possibly be compared with our algorithm. We split the baselines in several categories.

Basic baselines These baselines are here to assess the worst or best scores reachable for a RS algorithm:

- **Oracle**: this strategy knows the ground truth matrix and makes recommendation accordingly, with the list of best possible items for each user. Note that the Oracle will always reach a NDCG score of 1 since it achieves the ideal "reachable" score, but perfect abandonment or ERR score are not necessarily reached, as some users in the dataset might have only low scores (this implies that the probability of a click in the list of recommendations is not guaranteed).

- **Random**: at each iteration, a random list of items is recommended to the user. This method allows to judge the bias of ratings observed in the dataset: if the random method achieves a good NDCG, ERR or abandonment score, it is due to high proportion of high ratings in the dataset, since we can only perform recommendation among these known ratings.
- **Popular**: it is assumed we know the most popular items based on the ground-truth matrix, where the popularity is calculated as the mean rating given by users on this item. At each iteration, the 5 most popular items (restricted to the items rated by the user on the dataset) are recommended. This approach does not integrate any personalization as it only recommends items based on their average popularity among all users.
- **ALS**: Alternating Least Squares method (Zhou et al., 2008) does not use implicit feedback, but performs a Matrix Factorization based on observed ratings in order to optimize the prediction accuracy. We use a Factorization Machine implementation to mimic ALS.

Ranking methods These methods aim at optimizing the order of the recommended list only using one kind of feedback (in this case, both methods use only explicit feedback):

- **xCliMF** (Shi et al., 2013): this algorithm is a listwise Learning to Rank approach, built by optimizing the ERR on explicit ratings. It is the generalization of CliMF (Shi et al., 2012) from binary ratings to ratings with multiple levels of relevance. xCliMF does not use implicit features but targets an appropriate ranking of items. The algorithm decomposes the matrix with known ratings into a user features and item features matrices, where the predicted score does not attempt at optimizing the prediction accuracy, but only serve the purpose of sorting the items by predicted preference for each user.
- **CoFiRank** (Weimer et al., 2008): CoFiRank is based on Maximum Margin Matrix Factorization (Rennie and Srebro, 2005) and optimizes a ranking measure using explicit features. In the original paper, CoFiRank can optimize in three different ways: the first one is based on a convex upper bound of the NDCG, the second one is optimizing the RMSE for Regression, and the last one is the Ordinal Regression. In our experiments, we use the version optimizing the ordinal regression as it displays the best results.

Methods using both types of feedback These methods integrate both implicit and explicit feedback to perform recommendations

- **SVD++** (Koren, 2008): this approach also consists in a FM with data arranged to make the model mimic SVD++ (while adding pairwise interactions between features), as described in (Rendle, 2010). It also incorporates both explicit data and implicit data, but only implicit data about the item clicked.

- **Co-Rating** (Liu et al., 2010): this approach tries to unify explicit and implicit feedback. To do so, it first normalizes both explicit and implicit scores between 0 and 1 and combines them when solving the minimization problem, using ALS. The main difference with DualMF is that it does not assume any model behind the observed clicks. In order to compare fairly with DualMF, we also use a FM to learn the model, and choose the implicit score to be

$$\hat{r}_{u,i}^{imp} = \frac{\mathbf{A}_{u,i} + 0.5}{\mathbf{A}_{u,i} + \mathbf{S}_{u,i} + 1}. \tag{4.11}$$

Exploration-Exploitation approach Since we use a sequential setting, we also add a baseline tackling the problem from an exploration/exploitation point of view. The baseline approach is **PTS** (Kawale et al., 2015), the method presented in the Chapter 3 (once again, we use the non-Bayesian version). Compared to SeALS also introduced in the previous chapter, PTS can predict a list of items where each prediction embed the exploration/exploitation dilemma. It would be harder to decide how to include the exploration step with SeALS in the case of the recommendation of multiple items: at a random step of the algorithm, should the whole list of items be selected at random? Or only the first item in the list? For this reason, we only include PTS in the comparison. Note that this approach updates after every recommendation done compared to other approaches (see below).

Since most state of the art algorithms are not designed to handle the sequential aspect which implies frequent updates, we makes use of a batch update, where the model of each approach is updated each time 50% of the users have been recommended a list of items. If N is the number of users in the RS, T is the number of recommendations that should be done for every user ($T = 50$ in our experiments), and t denotes the time related to one step recommendation, then according to our setting t ranges from 1 to $T \cdot N$, and the update is done every time $t \equiv 0 \pmod{\frac{N}{2}}$. In our case, each algorithm will have 100 updates spread along the 50 recommendations.

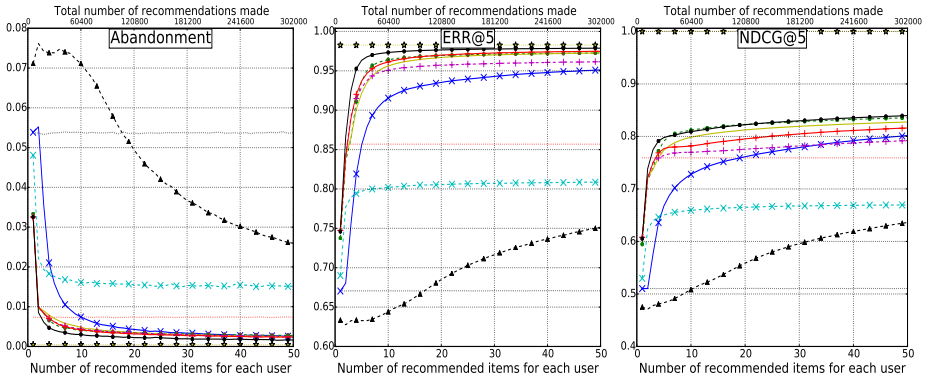
4.4.4 Results and Discussion

Results of all algorithms are averaged over 200 runs, and are shown on Figure 4.2. We use existing implementation called fastFM[4] for models using Factorization Machines (ALS, SVD++, SVD+-, Co-Rating, DualMF).

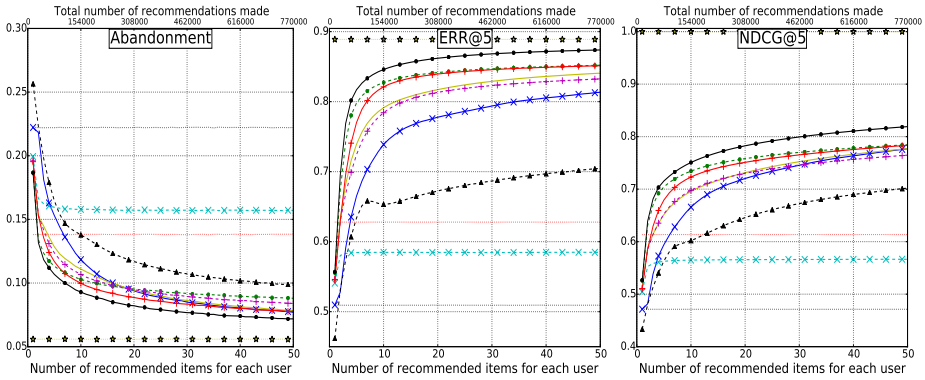
For Co-Rating and DualMF, we give different weights to explicit and implicit feedback when training the model, and best results are obtained by giving twice more weight to explicit feedback for DualMF and five times more weight to explicit feedback for Co-Rating.

Plotted values for each metric correspond to the average score for a user, obtained while recommending a list of items. In other words, each point in the curve correspond to one iteration of recommendations over all users: a number N of recommendations has been made (one per user), and the value plotted is the metric score obtained during this one iteration, averaged over N.

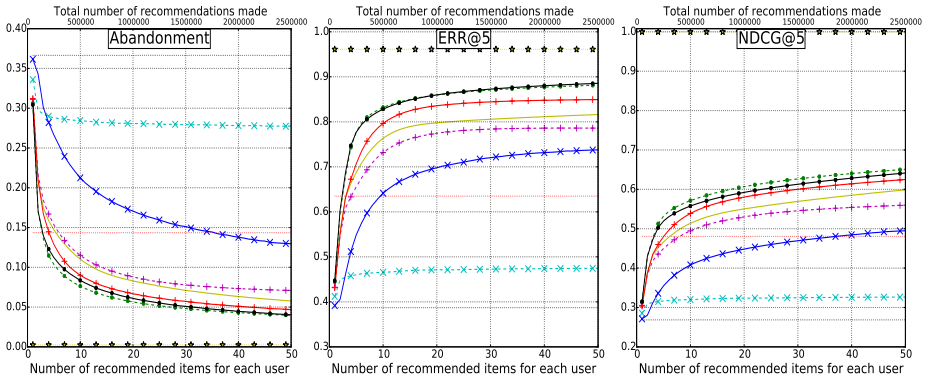
[4] http://ibayer.github.io/fastFM/



(A) Movielens1M



(B) Yahoo! songs



(C) Yahoo! artists

FIGURE 4.2: Evaluation of algorithms on three datasets and three metrics with the ERR click model (from the left column to the right one: Abandonment, ERR@5, NDCG@5).

For all algorithms learning matrices of features to represent users and items, we fix the number of columns of these matrices to 15. All algorithms using FM use a L2 penalty weight of $\lambda = 5.0$ for both pairwise coefficients and linear coefficients, and do one more step of learning at each update: they update their model learned previously by doing one pass on all data. CoFiRank and xCliMF do from 20 to 50 steps of learning at each update depending on the dataset, but relearn from scratch due to implementation. All these parameters are chosen following a grid-search on the datasets, and are the ones giving the most stable results on the three datasets. For PTS, we use a number of particles of 30 as in the original paper from (Kawale et al., 2015). We also do not apply PTS approach on the Yahoo! artists dataset due to the scalability issue already mentioned in Chapter 3.

From the results of experiments, we can draw three main conclusions:

About the use of both explicit and implicit feedback The three methods learning from both explicit and implicit feedback (which are DualMF, Co-Rating and SVD+-) perform significantly better than all other approaches on all datasets. In particular, our approach DualMF reaches the best performance on Movielens1M and Yahoo! songs, and is on par with Co-Rating on Yahoo! artists except on the NDCG@5 metric.

Note that the implicit score given during the learning phase by DualMF comes directly from the click model we defined: we know $p(u, i)$ defined in Equation (4.2) and we build directly a consistent estimator $\hat{r}_{u,i}^{imp}$. In practice, it would not be possible to access such knowledge and it would have to be inferred. Results of these three methods emphasizes the importance of using any feedback given by the user.

About ranking approaches The two approaches targeting specifically a good ranking by optimizing ranking loss (xCliMF and CoFiRank) surprisingly performs the worst, even compared to those using only explicit feedback like ALS. Aiming at the solving the learning to rank aspect does not seem to be a priority to reach a good performance.

About exploration/exploitation The approach PTS trying to tackle the exploration vs. exploitation dilemma does not reach good performance. On Movielens1M dataset, the ERR and NDCG scores are not increasing for the first recommendations, and the Abandonment score is even getting worse. The algorithm is even worse than Random at the first dozen of iterations.

The reason behind this behavior is that PTS encourages the recommendation of items with high uncertainty at the top of the list, since the algorithm tries to "explore" these items and gather information about them. At the beginning of the experiment, no information at all is available about any items, but the algorithm will quickly get feedback about items with high average ratings. However, users might never click on items with low average rating if they do not match their tastes, because the ERR click model is very punitive for items with low relevance. The algorithm fails to gather feedback about them, and since PTS does not consider the implicit feedback,

the algorithm will keep recommending these items at the top of the list until feedback is gathered, which might take long time given the probability of click is very low.

The reason why the curve of PTS is increasing more quickly in Yahoo! songs is due to the two datasets' characteristics: there are fewer items in this dataset and also more users. At each point of the curve, there are twice more ratings received in the matrix of ratings for Yahoo! songs dataset, on items which are more than three times less numerous. This means information is gathered far more quickly about items in Yahoo! songs and this helps greatly PTS to increase its performance from the beginning.

4.5 Experimentation with Other Click Models

In the experimental setting described previously, we make the assumption about the click model and consider that it follows the model defined as in the ERR equation. However, in real-world, the click model followed by users is not precisely known and probabilities of click on items on the list might differ noticeably from the ERR model. It is even likely that some users would follow one click model while other users would follow another one inside the same system, or that a user would behave differently depending on his mood or on the context he is in. In this section, we only consider that all users follow the same click model, but we apply click models different from the ERR click model to simulate the user interaction with the system, and we evaluate results of the same approaches as in previous section.

To define the other click models, we take inspiration from K. Hoffmann et al. (Hofmann et al., 2013a), who specified several models derived from the Dependent Click Model (DCM), which is a generalization of the Cascade Click Model. These models are themselves inspired from the analysis in (Guo et al., 2009a; Guo et al., 2009b). These click models were defined originally for the field of web-search and the recommendation of a list of documents which are ordered to their presupposed relevance, but they are also relevant to be applied in our experiments.

In the DCM, the user can click on several items on the list, and there is both a click probability and a stop probability for each level of relevance for the rating. The stop probability defines the chance of stopping to observe the list after having looked at the item. However, here we only study the Cascade Click Model with one click, so we only define the probability of clicking on an item, not the probability to stop observing the list, as the user stops as soon as he clicked on one item.

We evaluate algorithms on three more click models, referred as the *navigational model*, the *informational model* and the *almost random model*.

We first give in Table 4.2 the click probability of all models depending on the relevance rating of the item (the rating $r^*_{u,i}$ given by the user u on the item i in $\mathbf{R}^*$), with a rating scale from 1 to 5.

TABLE 4.2: The four different click models used in experiments.

Relevance rating	1	2	3	4	5
ERR model	$1/32$	$3/32$	$7/32$	$15/32$	$31/32$
navigational model	0.1	0.3	0.5	0.7	0.9
informational model	0.5	0.6	0.7	0.8	0.9
almost random model	0.4	0.45	0.5	0.55	0.6

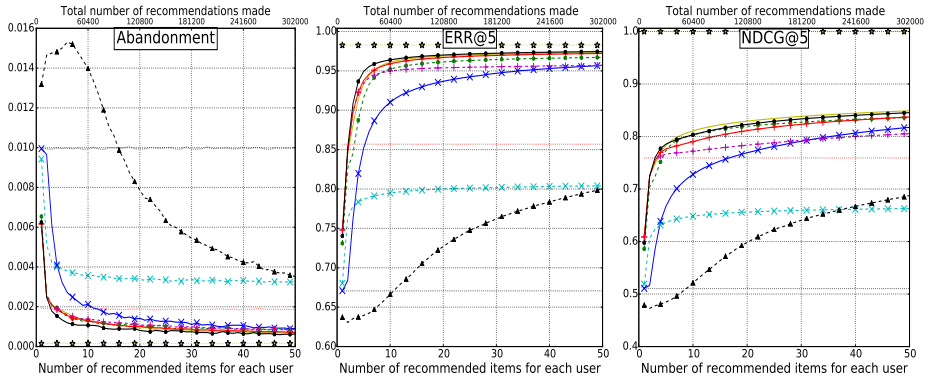
Note that compared to the ERR model, where the probability of click decreases exponentially as the rating decreases, the other three models probability decreases linearly. We give an explicit description of each model in the next sections, as well as results obtain from experiments performed following the same procedure as the one described in Section 4.4 (except for the click model representing the interaction between the user and the RS), with also same parameters for all algorithms.

4.5.1 The navigational click model

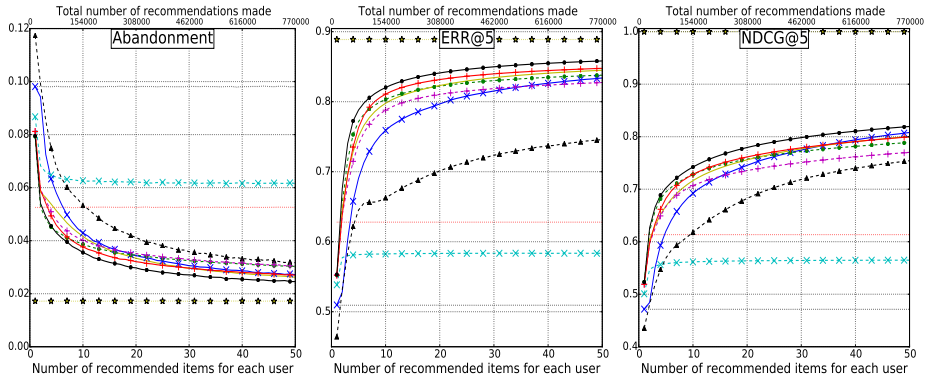
Description The *navigational model* simulates a user who is navigating on the RS application and who has a good knowledge of his own preferences. The user is focusing on items which match his tastes and interests. Thus, the probability of click is high for relevant items, and decreases quickly as the relevance level decreases. In this case, it is easy to make the difference between interactions (clicks) on items matching the user's tastes and items in which the user has no interest. The navigational model is the closest to the ERR click model out of the three click models used in this section, as the ERR model also gives a high chance of click on relevant items and a low probability of click on irrelevant items.

Results and discussion Results of all algorithms on the datasets for the navigational model are shown on Figure 4.3.

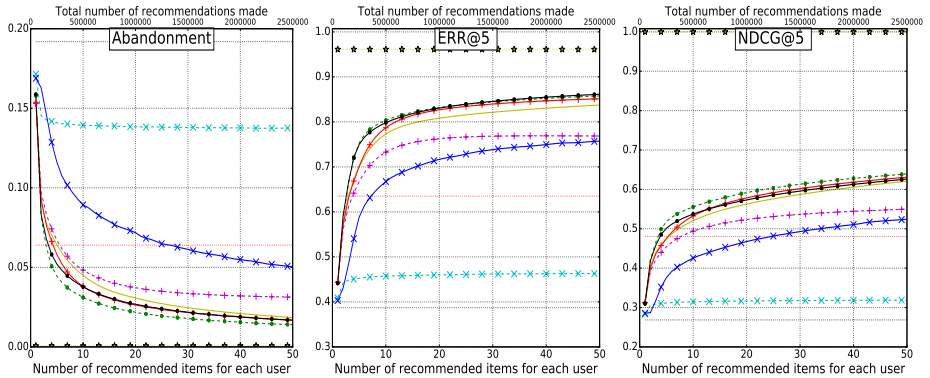
The navigational click model differs from the ERR model by the linear decrease of the probability of click depending on the relevance of the item, but it still strongly emphasizes the top items. As such, results are not too different, and DualMF is still performing well especially in general on the ERR metric and on the Yahoo! songs dataset, but it decreases on the two other ones since the model learned based on $\mathbf{R}_{imp}$ does not match the click model. Note also that the performance of CoFiRank increases, probably because items with lower rating have more chance to be clicked on than compared to the ERR model (the model receives more diverse levels of relevance of ratings to learn the pairwise relation). This also causes an increase in the performance of PTS: the algorithm still performs below others, but it is visible that information is gathered more quickly by the algorithm, especially on Movielens1M, as the algorithm surpasses Random more quickly than with the ERR click model.



(A) Movielens1M



(B) Yahoo! songs



(C) Yahoo! artists

FIGURE 4.3: Evaluation of algorithms on three datasets and three metrics with the navigational click model (from the left column to the right one: Abandonment, ERR@5, NDCG@5).

4.5.2 The informational click model

Relevance rating	1	2	3	4	5
informational model	0.5	0.6	0.7	0.8	0.9

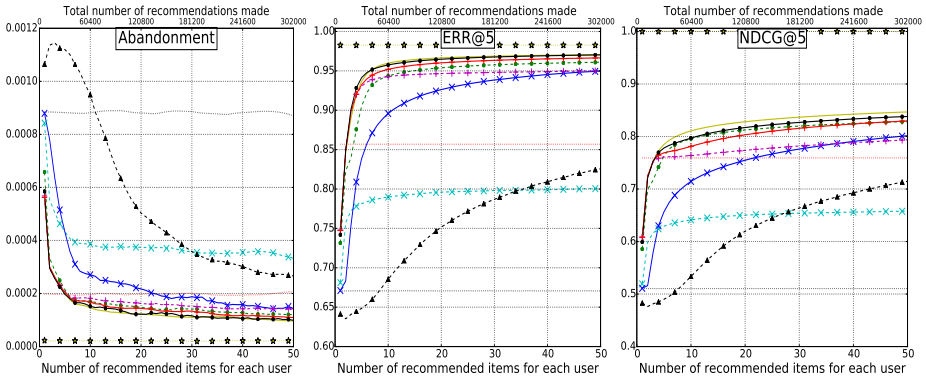
Description The *informational model* simulates a user who is less aware of what he is exactly looking for compared to the navigational click model, or who is searching for to gain some information in other items which may be less relevant. Considering the example of a music Recommendation System, it would signify that the user knows which artists or songs he likes, but is also willing to widen his interests and accept the recommendations about other artists which might not suit his tastes.

This characteristic results in a high probability of click on highly relevant items in the list, but the probability of click decreases slowly, as the user is not really sure if items suit his tastes or not. For example, an item who does not match the user's tastes at all (relevance rating of 1) still has one chance out of two to be clicked on.

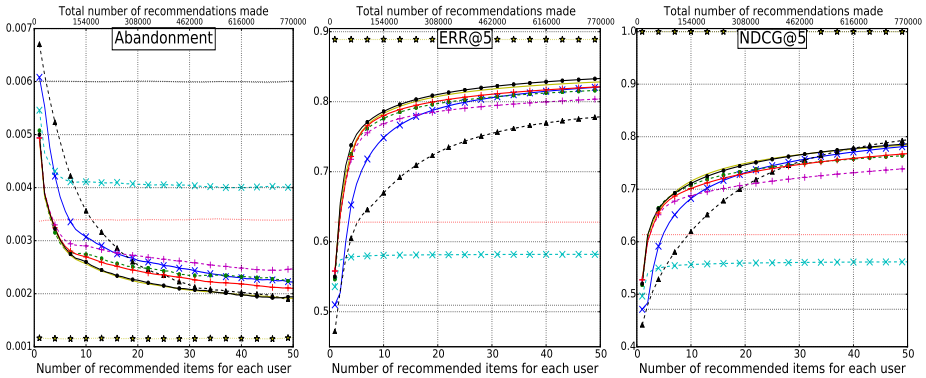
Results and discussion Results of all algorithms on the datasets for the informational click model are shown on Figure 4.4.

With this model, there is a higher probability of clicking on items with low relevance. It is harder for the algorithms including the implicit feedback into their model to capture the information embodied in the implicit feedback, since an item which has been clicked by the user does not necessarily implies the item is matching the user's tastes. The models of algorithms using implicit feedback have integrated this feedback optimistically, as a click is considered as a sign of preference for the user toward the item, which is not necessarily the case here. For this reason, all models using implicit feedback have a decreasing performance on ERR and NDCG compared to the navigational click model. DualMF on the Yahoo! songs dataset and Co-Rating on the Yahoo! artists dataset still reach the best performance, but their performances are on par with ALS which does not use implicit feedback.

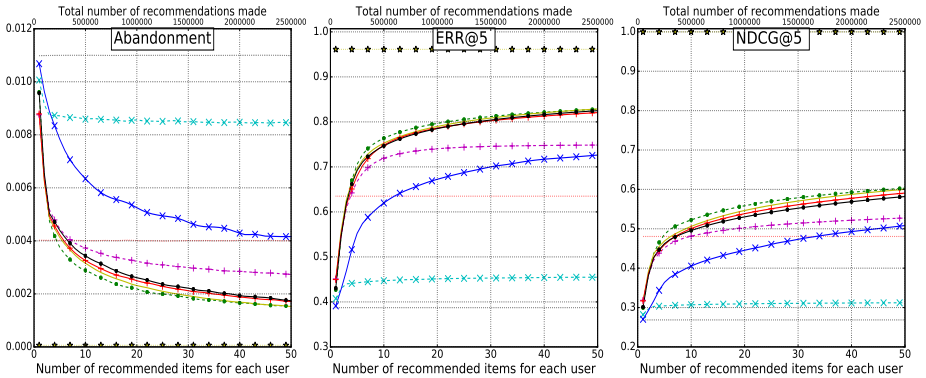
Finally, due to the higher chance to click on items with lower relevance compared to the ERR or the navigational click model, even more information is gathered about all items (even about items which have lower relevance). For this reason, the performance of CoFiRank is also high like in the results of the navigational click model. PTS also sees its performance increase even more compared to the navigational model. It even eventually reaches the best performance on the NDCG measure on Yahoo! songs dataset.



(A) Movielens1M



(B) Yahoo! songs



(C) Yahoo! artists

FIGURE 4.4: Evaluation of algorithms on three datasets and three metrics with the informational click model (from the left column to the right one: Abandonment, ERR@5, NDCG@5).

4.5.3 The almost random click model

Relevance rating	1	2	3	4	5
almost random model	0.4	0.45	0.5	0.55	0.6

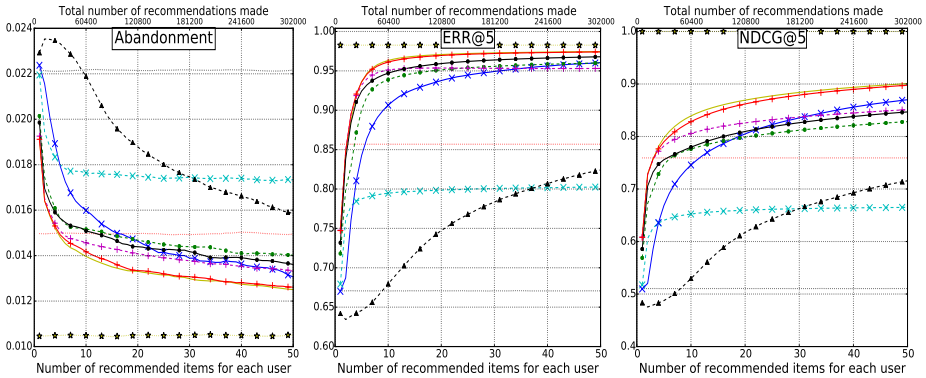
Description Finally, the *almost random model* supposes that the user barely has any knowledge about what he is exactly looking for, or does not look for anything in particular: it could mean he is willing to expand his tastes. Taking the example of a music RS again, we could imagine a user who does not really seek for any particular artists, and is willing to click on any suggestion in the recommendation list in order to discover new artists, but is not really more inclined to click on items which are more relevant to his tastes.

Therefore, the click model assumes the difference of click probability between a relevant and a non-relevant item is small, and the probabilities of clicking on an item are very close to each other no matter what the level of relevance of the item is.

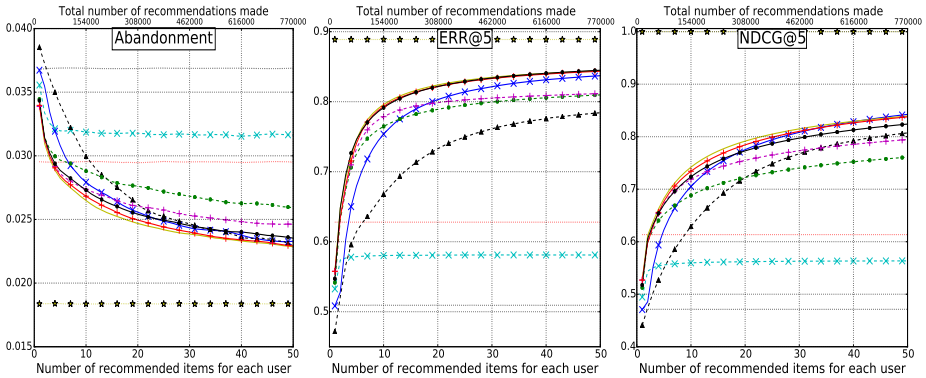
Results and discussion Results of all algorithms on the datasets for the almost random click model are shown on Figure 4.5.

With this last click model, the average abandonment score is higher for every dataset since the chance of clicking on an item is at best 60% for an item with the highest relevance. However, the ERR and NDCG can still get a high score as more information is received about user's tastes on all levels of relevance, and this can help to model accurately the user (this is especially visible with ALS). Our approach DualMF is still better on Yahoo! song, but its performance on the two other datasets decreases strongly, especially on Movielens1M. ALS reaches the best performance on Movielens1M and Yahoo! artists.

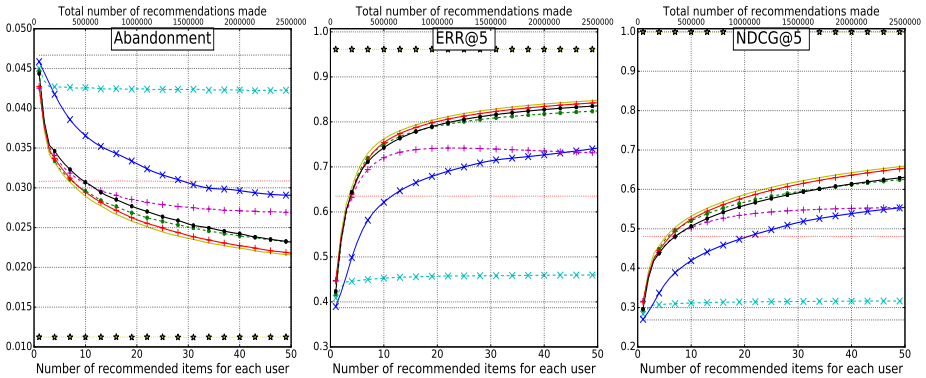
The only approach incorporating feedback which reaches a good performance is SVD+-, since it uses a more careful way to integrate feedback by including it in the feature input, and does not put any prior. The reason why algorithms incorporating implicit feedback still succeed on Yahoo! songs is maybe due to the higher standard deviation among ratings, which helps the differentiation between items during the ranking (the difference of picking an item with highest rating and one with lowest rating is still 20%). It is also possible the weight given to implicit and explicit rating in Co-Rating and DualMF may need some adjustments depending on dataset. Finally, PTS reaches a performance similar to the one in the informational click model, since in this case, information is once again received about items with low average relevance.



(A) Movielens1M



(B) Yahoo! songs



(C) Yahoo! artists

FIGURE 4.5: Evaluation of algorithms on three datasets and three metrics with the almost random click model (from the left column to the right one: Abandonment, ERR@5, NDCG@5).

Discussion from all results

One observation from results on the three click models is that the more the click model supposes that the user makes a clear distinction between items with high and low relevance, the more the implicit feedback brings information to the model. Indeed, in this case, the click on the item implies that the item has a high relevance, and that non-clicked items which had a higher position in the ranked list do not have a high relevance (which is a useful information for the system), and a clear separation can be made by the model between good or bad items to recommend.

However, even though they are less realistic, noisy click models like the almost random one highlight the caution needed when incorporating implicit feedback. In this case, it may seem better to only use explicit feedback as incorporating implicit feedback brings a lot of noise. But keep in mind that the implicit feedback is not incorporated correctly in the case of an Almost random click model by DualMF or Co-Rating, as our prior on the information brought by a click or no-click is optimistic (a click implies that the implicit rating $\hat{r}_{u,i}^{imp}$ is high). A more careful tune of the way the implicit feedback is integrated could still improve slightly the model compared to using only explicit feedback.

The exploration-exploitation approach with PTS brings interesting results, and another perspective from Chapter 3. The success of exploration-exploitation strategies also relies on the click behavior of the user: if the user is willing to give feedback on items which does not suit his tastes (an informational behavior for example), then only in this way the algorithm can gather information and reduce the exploration to exploit more. This highlights the necessity to use the implicit feedback in addition to only the explicit feedback, to avoid the algorithm being stuck in a exploratory phase.

4.6 Concluding Remarks

We study in this chapter Recommendation Systems from a novel point of view, where at every step a list of recommendation is provided to the user, who looks at items one by one following their order and pick one item (or none). For this setting, the system receives explicit feedback from the picked items and implicit feedback from the (no-)clicks. This model of interaction impacts the learning algorithm but also the experimental setting and results.

We provide a case study for a specific interaction model based on the ERR metric for which we propose two approaches, tackling the problem from two distinct perspectives. We evaluate various state of the art methods on several metrics and also several click models, and results on experiments display how considering both implicit and explicit feedback can significantly improve the performance, but also raises the issue of carefully incorporating them into the RS model.

Chapter 5

About Challenges in Real Recommendation Systems

On the previous chapters, experiments were made by simulating the sequential context of the RS: in Chapter 3, we supposed that the user was giving his explicit preference on the recommended item at each recommendation made, while in Chapter 4 the behavior of the user was simulated through his (no-)click on items in a list of recommendations. The second experimental section with various simulations of user click's behavior gave a first glimpse about the difficulty of modeling the behavior of users to make the evaluation similar to a live RS: offline simulations cannot grasp for all aspects of a real-world RS. In this chapter, after introducing several aspects of real RS, we discuss about offline evaluation for sequential RS. Finally, we give a broader discussion about building a real RS, from our results in a Challenge organized by the main conference on RS[1].

Contents

[1]This part of the chapter is extracted from our paper at the Recsys Challenge workshop in 2014 (Guillou et al., 2014)

5.1 Some Aspects of Real-world RS

In Chapter 3, we described a simple sequential evaluation setting also used in (Kawale et al., 2015). However, several aspects of real-world RS are not integrated into this setting and should be taken into consideration when building a real Recommendation System. In this section, we describe several aspects of real-world RS and emphasize their importance especially in the context of a sequential evaluation.

5.1.1 Power-law Distributions

The first aspect in a real-world RS is that users do not request recommendations nor rate items uniformly. Some users are more active than others and will use the RS more often. This also usually means that they provide more feedback for the RS to use, and their tastes can be modeled better. The distribution of ratings per user for RS is represented by a power-law distribution, with a long tail meaning that most users only provide little to no feedback to the RS.

Likewise for items, few items receive most of the attention from users: some are very popular and collect a lot of ratings, while others receive barely no ratings. The number of ratings received by each item also follows a power-law distribution.

An example of power-law distributions for users and items is given in Figure 5.1, which displays for the Movielens1M dataset the number of items rated per user, as well as the number of ratings received per item. Remark that power-law distributions of other datasets used in this thesis are even more severe. As such, the sequential offline settings described in Chapter 3 and Chapter 4 are not realistic since the selection of the user to which the recommendation is provided is made uniformly at random.

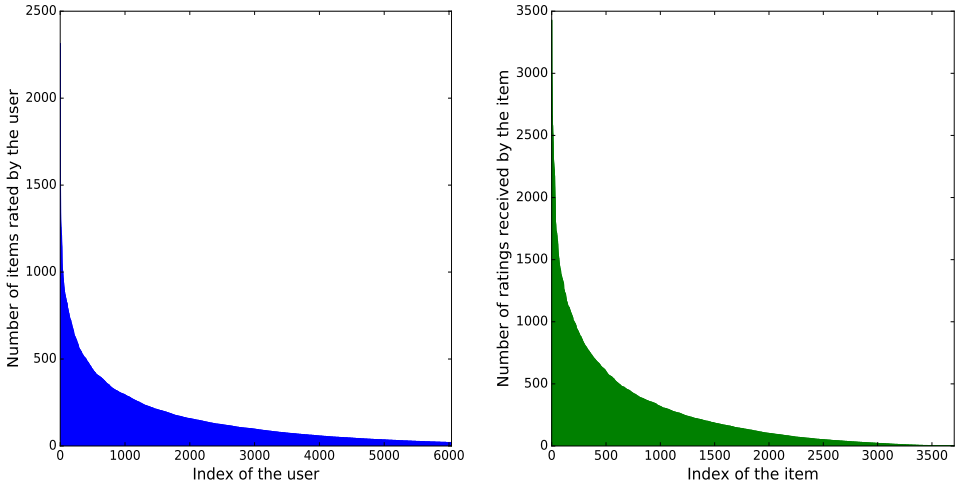


FIGURE 5.1: The power law distributions for the ratings from users and on items in the Movielens1M dataset.

5.1.2 "Replay" Aspect

We denote by "replay" the action for the RS to recommend the same item several times to one user after feedback is received. Except for some specific domains of recommendation like music or short videos (such as in Youtube), the recommendation of the same item to the same user would usually lead the user to abandon the system and is not advised inside a RS. Indeed, a user usually does not wish to be recommended repeatedly the same movie or computer to buy for instance, especially if he already watched the movie or bought the computer. For this reason, items for which feedback have already been given, or items which have been ignored several times by the user (when being recommended) are often discarded from the set of possible future recommendations.

As mentioned above, the only exception relates to music or short videos, since consumers are likely to listen to a song or watch a short video several times. However, even in these cases, the RS should carefully be tuned so that the item is not being recommended continuously.

In this regard, the setting described in Chapter 3 is also not realistic as items recommended are not discarded from the set of possible future recommendations. This is due to the Multi-Armed Bandit view of the problem, where usually, the replay of arms is allowed in the evaluation phase. The setting used in Chapter 4 also allows items to be recommended several times, but the approaches proposed in this setting are built more appropriately for RS which allow such behavior like music RS.

5.1.3 Large Set of Possible Recommendations

In a real-world RS, the number of items can be extremely large, and usually there is no restriction to the set of items among which the model can choose the recommendations for any user. The set of items that may be recommended to the user at a time step t (denoted $\mathcal{A}_t(u)$) is usually equal to the set $\mathcal{M}_t$ of all items in the system at this time step. Items already recommended to the user, in the set $\mathcal{I}_t(u)$, are eventually discarded, such that $\mathcal{A}_t(u) = \mathcal{M}_t \setminus \mathcal{I}_t(u)$.

As the set of possible items to recommend is very large in some domains like e-commerce websites, the domain of possible recommendations can be restricted using items previously rated by the user: for example, a RS can select items to recommend among the items which are similar to the ones already rated by the user, based on some content information or on a clustering technique. Eventually, some exploration can be done to explore beyond these similar items.

However, a problem occurs in offline sequential evaluation, as not all items have been rated by the user in the dataset. The situation is even opposite to this case as only a very small fraction of ratings in the whole matrix are known. Since the real preferences for most of the (user-item) pairs are unknown, we have to restrict greatly the set of possible recommendations for a given user u, by only allowing the RS to pick recommendations among items i for which the rating $r_{u,i}$ is known.

Three ideas to circumvent this issue would be:

- Build an artificial dataset with a full matrix of preferences. In this case, all preferences are known and any item can be recommended to any user. The problem with this solution is that the artificial dataset built does not match real world, and results obtained by recommendation techniques on such dataset offer less guarantees for real-world applications.
- Select a subset of users and items to constitute a more dense matrix of ratings, by selecting the users who have rated the highest number of items, and items that have been the most rated. This is a simple solution, and this is also the procedure we followed in the Chapter 4 to build Yahoo! artists dataset. However this introduces a bias toward users who are very active or items which receive a lot of attention.

  The filtering on Yahoo! artists dataset is a good example of this bias: we added an additional filtering between the Chapter 3 and 4 by selecting the most 50,000 active users out of 1,065,258 users, and the most 10,000 popular items out of the 98,209 items. This made the average rating in the matrix drop from 3.03 to 2.15: the reason in this case is probably that users who rated a lot of items were very active users of the Yahoo application, and they received of lot of suggestions about artists which they dislike. This leaded to a lot of really low ratings in the set of items rated by these active users. It is also possible that restricting to only very popular artists made the average rating of the dataset drop, as some very "popular" artists divide the public opinion and are disliked highly from a portion of the users.
- Use a sparse real-world dataset and fill the empty entries artificially. For example, a Matrix Factorization model could be learnt on all the known entries in the dataset, and this model could be used to fill all empty entries with its predictions. The issue in such a case is to find an appropriate model to perform this task. Even then, the entries introduced artificially into the matrix of preferences may introduce a strong bias, and lead to misleading results for the sequential evaluation.

As such, no proper solution for this aspect can be found for the offline sequential evaluation. This section also brings to light the problem of bias in RS datasets: as all datasets contains items which have been rated freely by the user (or following a recommendation), most datasets are highly biased toward high ratings. However, in an offline evaluation, we restrict the set of possible recommendations to known ratings. If all known ratings for a user are only high ratings, then it is harder to evaluate realistically: our algorithm would have to choose only items liked by the user, while in a real-life setting, the RS should choose among a set of items where preferences are highly different. To our knowledge, the only dataset bringing more various ratings is Yahoo! songs used in Chapter 4, as it introduces some ratings on items presented randomly to the user.

5.1.4 Stock Availability

The stock availability refers to the fact that not all items are always available for recommendation. This problem arises especially in e-commerce websites, where an inventory of items is available to the RS. The quantity of items left in the inventory also affects the recommendation, as the RS should decide to which user it would be more appropriate to recommend an item, especially when the supply of this item is getting low. Another example of this aspects is a DVD rental website, where items come and go into the RS as users rent them.

This aspect concerns mostly the RS side, to improve the model providing recommendations: for example, when only a small stock of an item is left, it is preferable to recommend this item to users who are demanding and not so flexible about their choices. It is also interesting for the RS to recommend this item to users who are likely to give a lot of information about it, as the RS will also benefit more from this feedback to improve future recommendations.

As no inventory is available with common RS datasets, this aspect can only be simulated but cannot be targeted realistically. Note that the "Replay" aspect described in Section 5.1.2 is somehow related to the stock availability, as the setting where the RS discards an item from the possible future recommendations after it is recommended could be considered as a specific "stock" setting where the item would be available in the inventory only one time for each user.

5.1.5 Past and New Users/Items

In a real-world RS, new items can be released at a specific time, or old ones can disappear from a RS. In a similar way, new users can enter or leave the system. From this point of view, the sequential offline evaluation setting we set in Chapter 3 made several hypotheses:

1. At the beginning of the evaluation, every user or item is considered "new", as the situation is a complete cold-start.
2. Since the user to which the recommendation is made at time t is selected uniformly at random, the phenomena of "a new user enters the RS" is considered. However, as users are selected uniformly, and it is allowed to recommend continuously the same items to a user, the phenomena "a user leaves the system" would never happen if t tends to infinity (because there will always be items to recommend to any user, and the probability to select a user is non-zero and fixed).
3. We did not consider any temporal information about when items (like movies for example) were released, and allowed the recommendation to be made in the whole set of items (restricted to the ones for which the preferences are known for a given user). For this reason, the effect of a "new item" or "disappearing item" never appears.

The temporal information about when item were released could be used to recreate the arrival of new items in the dataset and simulate the recommendation process at a specific timestamp: in this case, the set of possible items would be restricted to items which are available at this timestamp. We did not consider this in our setting, as this would reduce even more the set of possible recommendations which is already small for most of the users.

5.1.6 The Influence of the Recommendation System

Finally, the last aspect relates to the RS action: by recommending items to users in a specific sequential order, it also potentially influences them and their preferences. This aspect is linked to Reinforcement Learning (RL): if a Recommendation System is considered as a learning agent in a RL problem modeled as a Markov Decision Process (MDP), then the action of recommending an item to a user would be an action of the agent in the environment. This action gives back a reward to the RS and also results potentially in a change of state.

In our setting with a Multi-Armed Bandit formulation, the action of the RS has no effect on the user: his tastes are not changing depending on the action of the RS. We did not consider the problem of recommendation as a MDP, and almost no research has been done yet on this topic except (Shani et al., 2005; Mahmood and Ricci, 2009), as it is a more complex task to formulate and evaluate offline RS as a MDP.

5.2 Realistic Offline Sequential Recommendation

We described in the previous section several aspects of real-world RS. We incorporate some of these aspects in our offline evaluation setting, to make it more realistic. We perform experiments similar to the ones done in Chapter 3, but in a setting where items cannot be recommended several times and where the arrival of users in the RS follow a power-law distribution. These experiments highlight some issues of offline sequential evaluation.

5.2.1 Setting

We carry out again the two experiments done in the Chapter 3 in the section 3.5, about both the impact of exploration and the update of the model for SeALS algorithm.

However, we modify the setting for the evaluation in order to make for a more realistic evaluation scenario, by adding the power-law distribution of users described in Section 5.1.1 and by forbidding the "Replay" aspect explained in 5.1.2. More precisely, the setting is the following: for each dataset, we start with an empty matrix $\mathbf{R}$ to simulate an extreme cold-start scenario where no information is available at all.

Then, for a given number of time-steps, we loop on the following procedure:

1. we select a user u_t following the power-law distribution,
2. the algorithm chooses an item i_t to recommend,
3. we reveal the value of $r_t = r^*_{u_t,i_t}$ and increment the pseudo-regret score as in Equation (3.2),
4. we discard the item i_t from the set of possible items to recommend for user u_t.
5. If the set of possible items for the user u_t is empty, we delete this user from the possible users to recommend to.

To represent the power-law effect in step 1, we give to each user a probability to be picked, depending on the number of ratings from this user in the full true ratings matrix $\mathbf{R}^*$. The probability is simply the proportion $\frac{|\mathcal{I}^*(u)|}{|S^*|}$ normalized between 0 and 1, where $\mathcal{I}^*(u)$ is the set of items rated by u in $\mathbf{R}^*$, and S^* is the set of known (user-item) pairs in $\mathbf{R}^*$.

At step 4, we take care of the "Replay" aspect, by deleting the recommended item from the set of "available" items for future recommendations to this user. Note that deleting items from the set of future recommendation also leads to another effect: as we delete items from the set of possible items, the set for some users will eventually become empty. In such a case, the RS cannot provide any recommendation and we decide to discard the user from the future iterations, as described in step 5. By some means, it creates the effect of past/new users described in Section 5.1.5, as some users will disappear from the system.

Note that the formula used for the cumulative pseudo regret is also different from the one defined in Chapter 3. As items are deleted from the set of possible recommendations, the best possible choice for a given user might change through time. If we denote $\mathcal{A}(u)$ the set of items available for recommendation for user u, the regret is defined as

$$\mathcal{R}_T = \sum_{t=1}^{T} \max_{i \in \mathcal{A}(u_t)} r^*_{u_t,i_t} - \mathbb{E}[r_t] = \sum_{t=1}^{T} \max_{i \in \mathcal{A}(u_t)} r^*_{u_t,i_t} - r^*_{u_t,i_t} \,. \tag{5.1}$$

5.2.2 Results and Discussion

We carry experiments on the same datasets as in Chapter 3, and compare with the same baselines (cf. Section 3.5.2 for details). However, note that we do not perform as many iterations of recommendation: we perform $500,000$ iterations for Movielens1M, 10^7 iterations for Movielens20M, LibimSeti and Douban, and $25 \cdot 10^6$ iterations for Yahoo. Indeed, if the RS was to perform a high number of iterations, the set of possible recommendations would decrease too much for any user in the RS, and results would not be significant anymore as the choice for the RS would be too restricted (the RS would have to choose among a set of very few items for each user as most possible items would have been recommended before).

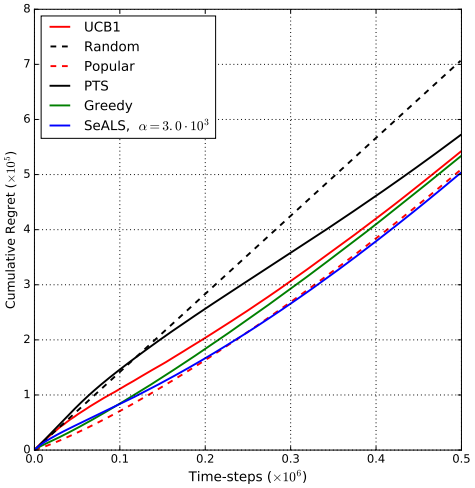
Impact of Exploration

We first study the results about the impact of exploration, given in Figure 5.2, to compare to Figure 3.2 in Chapter 3. The values of T_u used for the period of the update are the same for SeALS and Greedy, and T_u was set to 250 for Movielens1M, 10,000 for Douban and Movielens20M and 50,000 for Yahoo! dataset. We set the regularization parameter $\lambda = 0.1$ for Greedy and $\lambda = 0.15$ for SeALS, as well as the parameter k set to 15, which are the same values as in Chapter 3.

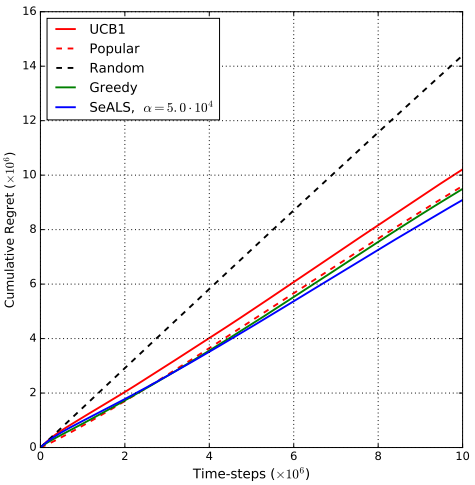
The experiments display very different results from the ones in Chapter 3. Except for Yahoo, all curves exhibits either a linear or increasing regret through time, except at the very beginning of the iterations (this is particularly visible on LibimSeti). Moreover, even if SeALS still appears as the best approach, the difference between algorithms is far less visible than in Chapter 3. This is due to the high sparsity of the matrix of ratings: if a user has very few ratings and it is forbidden to play items more than once, the task of learning users' tastes becomes more complex, and it becomes harder to make the difference between a good or a bad algorithm as both have little information and strong restrictions to perform their recommendation. Even if it eventually reaches a good model, PTS on Movielens1M also takes a long time to converge to this model: previous experiments showed that it does not reach good performance as quickly as SeALS, and forbidding the "replay" seem to affect it even more.

Notice that the value of α for the exploration in SeALS to get the best results is lower than the one used previously. There are several reasons behind this decrease:

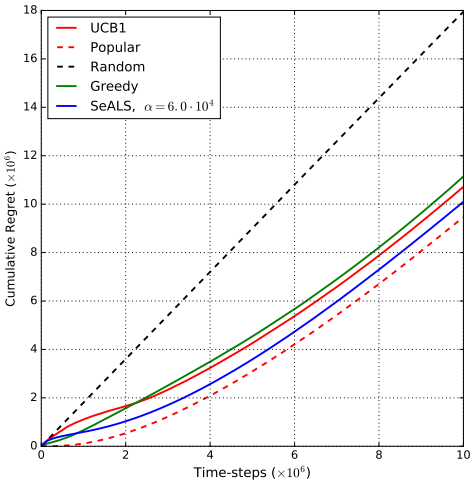
- We do not perform as many iterations of recommendation. The optimal score of α depends on how many recommendations are performed: if a high number of recommendations are made by the RS, then a stronger exploration is necessary to keep exploring longer and minimize the regret.
- The choice of the user following a power-law distribution also has an impact about how exploration should be done. By choosing an ε_n-greedy strategy, a lot of exploration is done at the beginning, and the RS can benefit a lot from this choice when users are chosen uniformly at random, as exploration is performed on a large set of users and information is gathered quickly. In the case of a power-law distribution, the exploration at the beginning will be done on a smaller number of users, who will rate a high number of items. By emphasizing exploration too much at the start, even if the preferences of this small set of users have been explored enough, there is a risk that the RS will miss the target to recommend accurate items to these users.
- the last reason is that items are deleted from possible recommendations after being recommended to a user. The exploration in the case where the "replay" is allowed brings more information about which item to play as all items are always available to be recommended in the future. In the current setting, the exploration will still bring information, but this information has to be used only on a different set of items, which reduce slightly its importance.



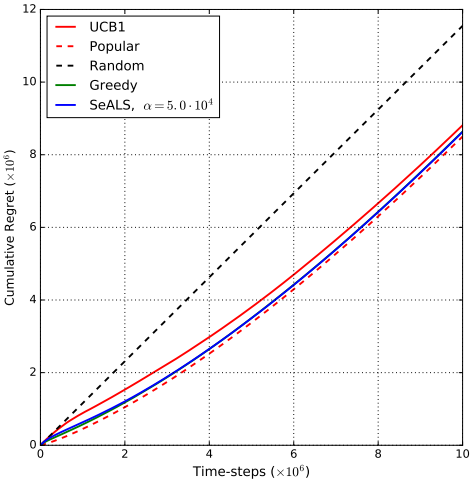
(A) Movielens1M



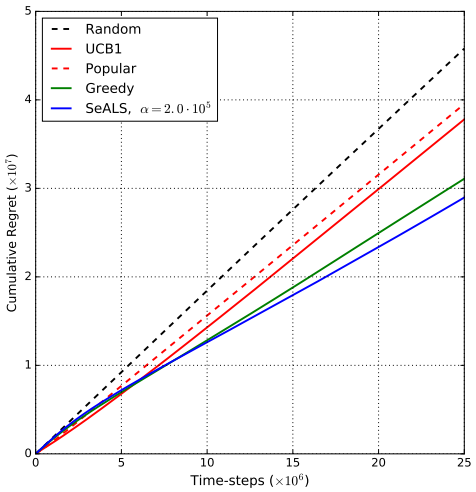
(B) Movielens20M



(C) LibimSeti



(D) Douban

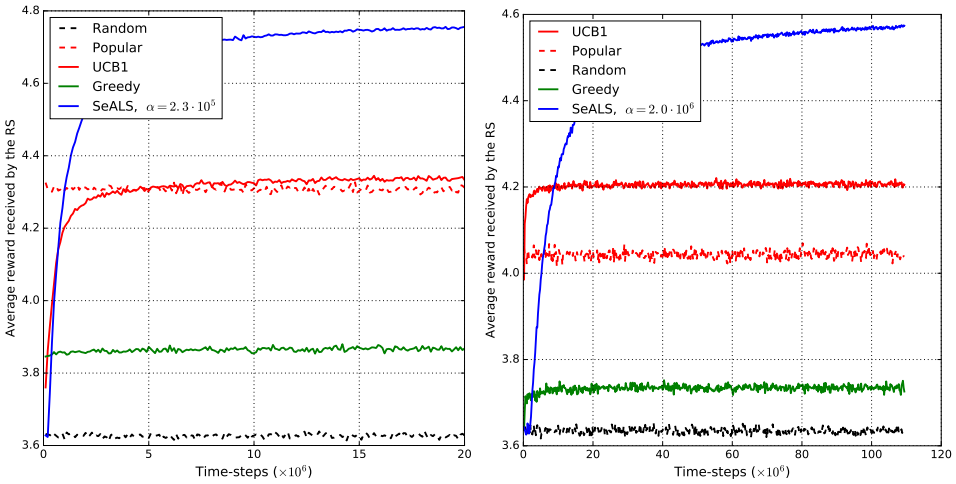


(E) Yahoo

FIGURE 5.2: Impact of exploration on the cumulative regret score when replay is not allowed, evaluated on five datasets.

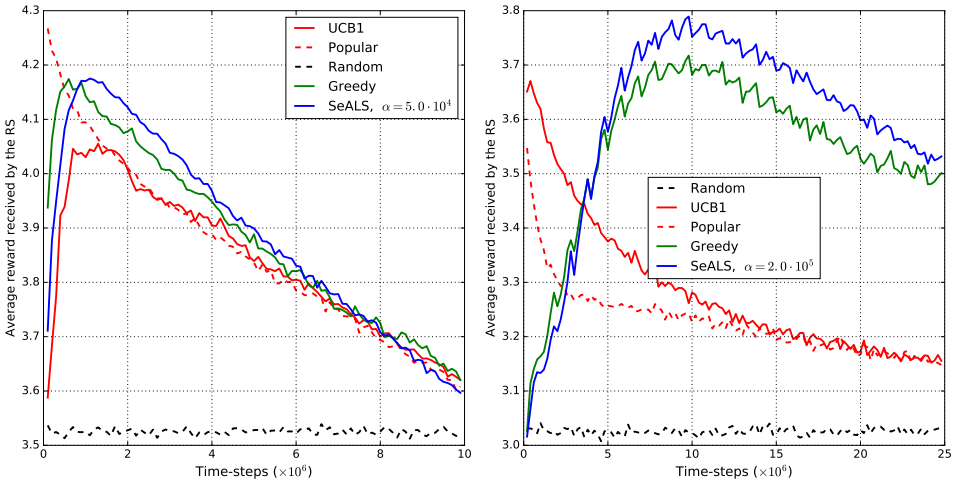
Then, we also display in Figure 5.3 and Figure 5.4 the curve of the average reward received by the RS, respectively in the setting of Chapter 3 and for the current setting. This allow to compare from another perspective the two offline settings.

In both Figures, each point in time is the average rating of the last 200,000 and 100,000 iterations done before this point, for Yahoo and Movielens20M respectively.



(A) Movielens20M (B) Yahoo

FIGURE 5.3: Average reward received by the RS through time, when replay is allowed.



(A) Movielens20M (B) Yahoo

FIGURE 5.4: Average reward received by the RS through time, when replay is not allowed.

As one can see, the learning curve of the algorithms is obvious in the case where the "replay" is allowed as all techniques except Random and Popular baselines see their average reward growing through time. On the opposite, the curves in the current setting display a stranger behavior which needs to be explained.

The comparison between the two figures lead to several conclusions:

- In the Figure 5.3, we observe that Greedy performance grows slightly at the beginning and remains stuck in a solution bringing low reward afterward. It reaches even worse performance than Popular or UCB1 approaches. On the contrary, forbidding the replay of the same item somehow forces the Greedy algorithm to "explore" and play other items, and it is able to reach a better score.
- For Popular or UCB1 approaches: in Figure 5.3, they both reach a good performance, and UCB1 is even able to reach better performance on both datasets. Then, in Figure 5.4, we observe that they both perform well initially and gather high feedback. This is because they either know in advance (Popular), or discover (UCB1) which items are popular for most users, and recommending these items brings high average reward. However, both scores decrease quickly because the two algorithms lacks personalization to perform good recommendation afterward.
- The most important effect is the decrease of the reward received by Greedy or SeALS in Figure 5.4 (which also happens for Popular and UCB1). It happens very quickly on Movielens20M dataset, and after about 10 millions recommendations for the Yahoo dataset. The meaning of this phenomena is that good items for some users have already almost all been recommended, and the RS is forced to pick an item to recommend from a set of items bringing lower reward. There are still some items bringing high reward among the ones to pick for the RS, since the regret is still increasing in Figure 5.2, but it is likely that the RS has already recommended all the "obvious" good items for the users (the items for which the preference of the user is obvious). The RS then has to recommend good items which are harder to detect, which explains why the average reward received by the RS decreases and why the regret increases even further in Figure 5.2.

  Remark that this effect also applies to UCB1 and Popular to explain the decrease of their results (since they recommended good items at the start), but the decrease appears more quickly in the recommendation process, and is stronger due to the lack of personalization.

At last, we want to emphasize the fact that results are sequential, and as such, a curve of an algorithm "crossing" another one and going above does not mean it would be preferable to change the stategy from the first algorithm to the second one. For example, SeALS is going under Popular, UCB1, and Greedy at the end of the experiment on Movielens20M, but this does not signify that it is advised to switch from the SeALS strategy to a UCB1 one at this point: at a specific iteration, for each user the set of possible items to recommend is different for every algorithms as this set results from the choices made before this iteration.

Update of the Model

We also study the update of the model, and display the results in Figure 5.5. Results are this time more similar to the ones obtained in Figure 3.3 in Chapter 3. Three observations can be made on this figure:

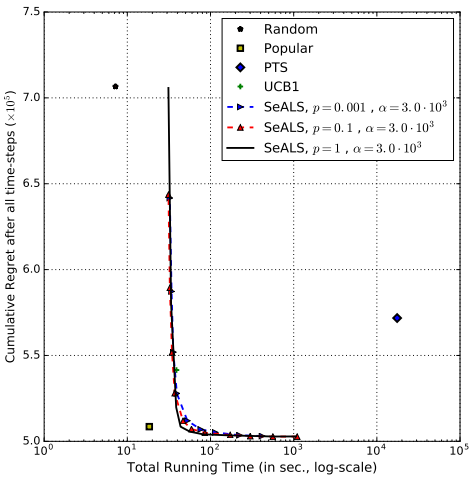
1. Updating the model more often benefits as much to the RS as in the case when the replay is allowed, as the cumulative regret is decreasing strongly if the model is updated more often. It even seems that it decreases slightly faster than in the experiments in Chapter 3. This might be due to the power-law distribution: a set of users receives recommendations more often and updating slightly more frequently this small set of users is sufficient to get good results.
2. There is little to no difference at changing the proportion of user to update p. Only the two larger datasets Movielens20M and Yahoo display a small improvement when choosing to update a smaller portion of the users and items. We only investigated the update of a proportion of users and items chosen at random uniformly in the Chapter 3 and in the current setting, but it might be more efficient both for the score and running time to choose which users or items to update in a smarter way.
3. While some results for different values of p were using different values of α in Chapter 3, it is not necessary here to tune α and reducing it does not bring any visible change. The reason may be that the value of exploration is already small, so reducing the exploration for smaller value of p would only lead the system to do almost no exploration at all.

5.2.3 Final Remarks

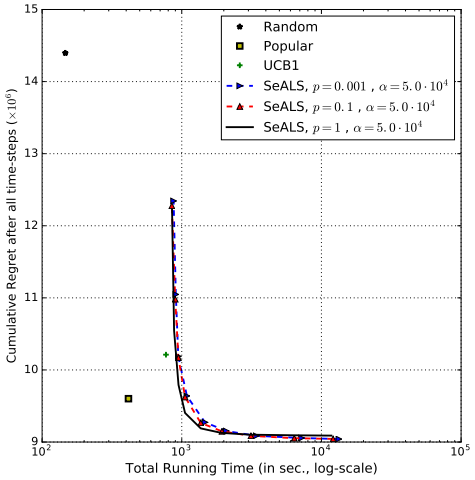
Results on this sections highlight the difficulty of carrying on an offline evaluation of RS in a more realistic sequential setting. The main issue is the sparsity and lack of ratings to simulate the no-"Replay" aspect where the RS is not allowed to recommend an item to a user more than one time.

This section brings to light the need for real-world RS datasets adapted to sequential recommendation. Most of the datasets available to public research were built during the years where prediction accuracy mattered the most to researchers in the field of RS, but they are not well adapted to simulate offline a sequential evaluation, which requires a high number of preferences known for the user on items randomly selected. Having some temporal information available could also benefit to perform more realistic offline evaluation, by simulating the arrival of new users or new items into the RS according to these information.

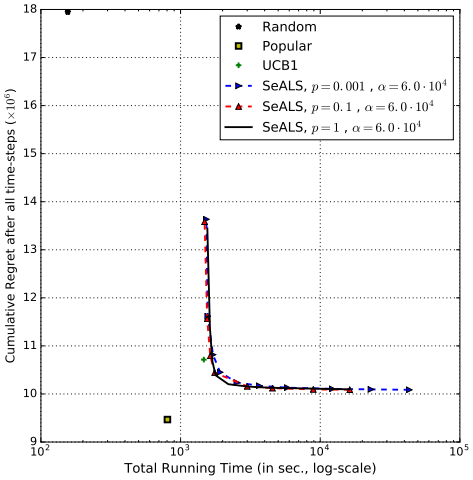
In the next section, we give some more general thoughts about real RS, which do not only apply to the sequential evaluation.



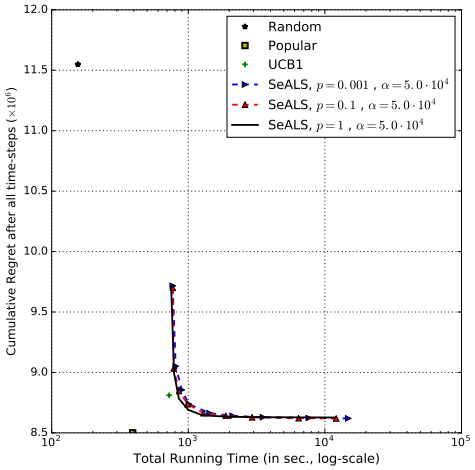
(A) Movielens1M



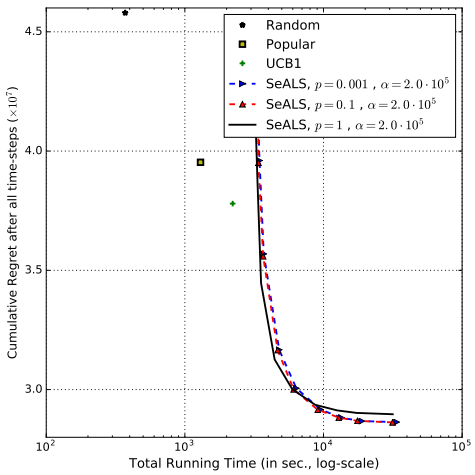
(B) Movielens20M



(C) LibimSeti



(D) Douban



(E) Yahoo

FIGURE 5.5: Impact of the update strategy on the final cumulative regret score when replay is not allowed, evaluated on five datasets.

5.3 Some Lessons from a Real Case RS Challenge

This section is built after the results on real data from the RecSys Challenge 2014 (Said et al., 2014) on which we reached the highest score. Conclusions drawn from the method used to win this challenge highlight some interesting aspects of building a real RS. Note that this Challenge did not target any sequential evaluation, but investigated a specific recommendation task: find items with the highest user engagement. Specifically, the objective was to rank a set of tweets after the (unknown) number of times they will be favorited or retweeted (Dooms et al., 2013). This problem came within the scope of a Learning To Rank point of view (cf. Section 2.7 in Chapter 2), since each user could be considered as a query, and each tweet associated to each user as an item.

We describe first in the following section the goal of the Challenge.

5.3.1 RecSys Challenge 2014: Data and Protocol

This challenge uses a traditional evaluation of Recommendation Systems, by splitting a large dataset chronologically in three subsets: a training set, a test set, and an evaluation set. The percentage for each of these subsets is respectively around 80%, 10% and 10%. Contestants of the Recsys Challenge were provided on the training and test sets in order to build their models and algorithms, while the evaluation set was kept for the final evaluation in order to decide on the winner of the challenge.

The dataset is an extended version of MovieTweetings dataset (Dooms et al., 2013), so data originated from users of the IMDb app, where they can rate movies and share the rating on Twitter. Tweets and information related to them were collected by querying the Twitter API on a daily basis for tweets containing the keywords 'I rated IMDb'.

As input features for each tweet, the challenge dataset contains metadata of the tweets as provided by the Twitter API. These metadata include information about the user, the movie or the tweet itself. The retweets and favorites counts are also included in the metadata, in order to evaluate tweets by their engagement and rank them.

Data characteristics and statistics Each dataset contains tweets, which are represented as follow: (twitter user id, IMDb item id, rating given by the user to the movie, scraping timestamp, tweet data). We present in Table 5.1 some statistics about the training dataset. We denote a tweet which obtains a non-zero engagement as a *successful* tweet, and a user who have had at least one successful tweet as a *successful* user.

These statistics give us some important information: first, the number of successful tweets is really low, i.e. most tweets have no success and do not get any retweet or are put as favorite by other users. As a consequence, most users in the dataset only have tweets with an engagement of 0.

TABLE 5.1: Training data statistics

Metric	Value
Tweets	170,285
Unique users	22,079
Unique items	13,618
Tweets with zero engagement	162,107 (95.2%)
Unsuccessful users	17,502 (79.27%)

Successful tweets receive an average engagement of 4.41 but 80% of these tweets have only a user engagement of 1 (cf. Figure 5.6). Overall, most of the tweets have an engagement of either 0 or 1, which means the challenge is almost a binary classification problem. In fact, as shown in (Loiacono et al., 2014), the binary oracle, i.e. the classifier that only gives the score 1 to tweets with positive engagement and the score 0 to remaining tweets, gets an overall result of 0.9877 on the test set, where the maximum possible score is 1.

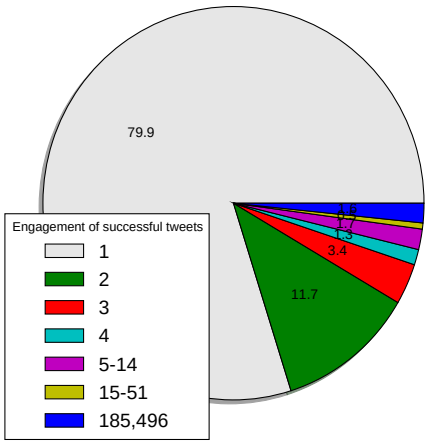


FIGURE 5.6: Distribution of the user engagement of successful tweets in the training dataset.

Secondly, part of these tweets are retweets (cf. Table 5.2). The engagement of a retweeted tweet is a specific case in the dataset, since such tweets share their retweet count with the original tweet (but do not share their favorite count). Every retweet necessarily has a strictly positive engagement. Given that both the retweet and favorite counts of the original tweet are available on the metadata, this distinctive feature has to be taken into account in the model. These tweets have not been originally posted by the user that we evaluate, so the user engagement of retweets can cause a distorted evaluation. On the 4,577 users who are considered successful, only 3,321 have successful tweets that have been posted by themselves, the rest of these users can be considered as "artificial" successful users since they only received a positive user engagement by retweeting, but not on their own tweets.

TABLE 5.2: About retweets in the training dataset

Metric	Value
Number of retweets	1,808
Percentage among successful tweets	22.10%
"Artificial" successful users	28.44%

About User Engagement as Evaluation This challenge focuses on a different way of evaluating models and recommendations. The goal of the algorithm is to determine the best ranking for each user after the engagement that each of his tweets receives. The user engagement for one tweet is calculated as the sum of the number of times it has been retweeted and the number of times it has been marked as favorite. In order to measure the performance of this ranking, the information retrieval measure used is the Normalized Discounted Cumulative Gain (NDCG), computed on the top 10 elements for each user. The overall evaluation is obtained by averaging the NDCG@10 of each user.

However, following the previous section which described characteristics of the dataset, some details have to be mentioned concerning the use of the NDCG measure. Firstly, most of the users don't have any successful tweet among their tweets. For these users, any ranking of the tweets is equivalent since all items have the same relevance.

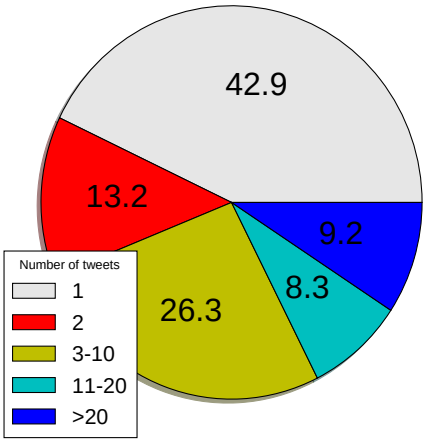


FIGURE 5.7: Distribution of the number of tweets per user in the training dataset.

Secondly, if we restrict ourselves to remaining users who have at least one successful tweets, it is also possible that any ranking will give the maximum NDCG score. This case can happen if the user only has one tweet in total (cf. Figure 5.7), or if all of his tweets have the same positive user engagement, for example a user with three tweets with an engagement of 1. Then the ranking provided by the model would not have any consequence on the NDCG score for this user, and the NDCG would always be 1.0 since any ranking would be considered perfect.

Overall, 2,792 users provide a clean ranking (after removal of the retweet effect). While these users represent only 13% of the total number of users,

they still gather 44% of the whole tweets, and 8% of their tweets are true successful tweets.

Input features for the model We use several features that seem relevant to predict the ranking or the user engagement of a tweet. Each of these features is contained in the dataset or is extracted separately to enhance the model. Each of these features enters into one of three categories: user features, movie features, and tweet features.

User features are given in the original dataset. These features include the number of followers, the number of friends, the number of tweet put as favorites, the number of statuses posted and the number of lists in which the user is included. User features can change through time, since a user might follow more people or get more followers, and feature values for each tweet are the one extracted at the exact time the tweet was posted. As for the movie features, we extract some features from IMDb website such as the IMDb rating, the IMDb votes, the budget of the movie or its release date.

Finally, we include also the features that are related to the tweet itself, such as the rating given to movie by the user, the date the tweet was posted, the time difference between the release date of the movie and the post of the tweet, or other information about hashtags, lang, retweet, image inside the tweet...

5.3.2 Method

After a brief description of each method used, we discuss how to combine different rankers, and then explain our approach.

LambdaMART model LambdaMART (Burges, 2010) uses a listwise approach: it considers the whole lists of items as instances in learning, and tries to optimize directly a performance measure.
LambdaMART has been created by combining two previous algorithms, MART and LambdaRank. MART is a pointwise ranking approach, based on a boosted tree model in which the output of the model is a linear combination of the output of a set of regression trees. It can be viewed as performing gradient descent in function space using regression trees. LambdaRank is a method based on neural networks, which expresses gradients based on the ranks of the documents, and modifies the weight in the neural net according to these gradients. The λ terms in LambdaRank can be seen as rules defining how to change the ranks of items in a ranked list in order to optimize the performance. Gradients of costs to optimize directly a performance measure are hard to compute since these measure are non-differentiable. Instead, the λ terms are considered to be gradients with contributions from all other items that have a different relevance label. LambdaMART combines both approaches by using the idea of λ terms from LambdaRank and MART's boosted regression trees. LambdaMART models have shown great efficiency in ranking problems and won the Yahoo! Learning to Rank Challenge (Burges et al., 2011).

Random Forests Random Forest (Breiman, 2001) is a kind of ensemble learning algorithm which combines predictions from an ensemble of random trees. Bagging is used to reduce the correlation between each pair of random trees in the ensemble. Compared to LambdaMART, the Random Forest method belongs to pointwise methods as it is a regression model. Each of the trees in the ensemble forest votes for the output value, and the predicted output is then determined by all the trees in the ensemble. This method has shown high performance and has been applied successfully in various different fields, including LTR competitions (Chapelle and Chang, 2011).

Description of the Approach Here we describe the models we use in our overall approach, and how we combined them. At first we built a LambdaMART model on a modified train dataset, in which we removed users who would have the same NDCG without regard to the ranking given by the algorithm. As a consequence, the training dataset for LambdaMART is very small. We tried to artificially augment the dataset as showed in (Burges et al., 2011) by sampling some percentage of tweets for each user and inserting these data in the training set. For example, instead of learning the ordering A>B>C for three tweets, sampling from these tweets and removing B help the model not to overfit and manage to learn that A>C without the condition of B lying between them. However, training on augmented models did not show any improvement on the evaluation. This is due to the characteristics of the data where only very few tweets, are successful for a given user, i.e. in most cases, the ranking problem is reduced to identify that one or two tweets will generate higher engagement than other tweets. These characteristics are highlighted by the almost perfect score of the binary oracle.

Such result encourages to also consider simpler algorithms, such as regression models. In Section 5.3.3 we present the results obtained with Linear Regression, and with Random Forests. One advantage of regression models is that they allow to easily correct the effect of retweeted tweets: (i) remove the retweet count of the original tweet from the features, (ii) while learning the model, modify the user engagement by subtracting the retweet count of the original tweet, and (iii) add the retweet count of the original tweet to the result returned by the learned model. In other words, the user engagement of any tweet that was not originally posted by the user, is reduced to its "true" user engagement, by dropping the retweet count that actually belongs to the user who posted the original tweet. At the contrary, the effect of such cleaning is less clear while using LambdaMART. In fact, LambdaMART model returns a value that does not target the engagement score as it focuses on the ranking of tweets. Hence, adding the retweet count of the original tweet to that value is meaningless.

Finally, we also explore various ways to combine these models. It is usually not an easy task to decide which method to keep in the final model if several methods perform similarly well, and combining rankers can be a way to minimize the chance of achieving poor results on an eventual new test dataset, because the diversification of methods will provide more robustness to the final model. Since the values returned by LambdaMART

and regression models are not at the same scale, we standardize both predictions. After this step, the combination of rankers can be done in various ways. We apply both a simple averaging method, and a method to perform an optimal linear combination of rankers described in (Wu et al., 2010). Given a pair of rankers who respectively gave the score s_1^i and s_2^i for a item i, the idea is to combined them convexly as:

$$s_{comb}^i = \alpha s_1^i + (1-\alpha) s_2^i \,, \tag{5.2}$$

where a parameter α is sweeping from 0 to 1. By enumerating all values of α for which the NDCG will change, it is possible to identify the value of this parameter for which the two rankers are linearly optimally combined.

5.3.3 Experiments and Discussion

In this section we partially answer two questions:

- While the challenge is a Learn To Rank problem, should we learn a ranking model or not?
- Is there an efficient way to combine ranking models with other models?

These questions are answered in the light of the results obtained on the data of the challenge. We consider both (i) the NDCG score obtained by the models discussed in the paper and (ii) the importance of features in the learned forests.

Hyper-parameters of simple models are selected through 10-fold cross-validation on the training dataset, with NDCG as objective function. These parameters include the number of trees, the number of leaves in each tree, and the learning rate of LambdaMART. Parameters found were 500 for the number of trees, 10 for the number of leaves and a learning rate of 0.05. We chose for Random Forest algorithm to use 2000 trees with a maximum depth of 5.

Experimental Results

While this strategy is subject to overfitting, models are compared after their NDCG@10 score on the test set. Results are given in Table 5.3 where *λ-MART*, *RF* and *Lin* stand respectively for LambdaMART, Random Forest and linear regression. *Retweet* is the model which predicts as a score the number of retweets of the original tweet (0, when the tweet is an original one, not a retweet).

The suffix *Wrap* indicates that the corresponding regression model is used on dataset for which retweet effect has been cleaned. Specifically, during the training phase, the number of retweets of the original tweet is removed from the features and is subtracted to the user engagement. During the test phase, this number is added to the engagement predicted by the learned model.

TABLE 5.3: NDCG@10 on test dataset

Model	NDCG@10
Retweet	0.806
λ-MART	0.838
RF / WrapRF	0.823 / 0.858
Lin / WrapLin	0.806 / 0.843
Mean(λ-MART, Retweet)	0.876
Mean(λ-MART, WrapRF, WrapLin)	0.874
OptAvg(λ-MART, WrapRF, WrapLin)	0.876
OptAvg(λ-MART, RF, Retweet)	0.878

Finally, *Mean* stands for the meta-model which uses the average predicted score to rank tweets, and *OptAvg* stands for the meta-model using the best linear combination[2]. Notice that these best linear combinations are selected after the NDCG@10 score on the test set.

The first remark on the results is the importance of the retweet score. For example, this score alone is enough to reach an NDCG of 0.806. Similarly, the combination of Retweet with other models (through linear combination or wrapping) increases the NDCG score of these models to the extent of 0.035. Finally, any of the best models uses the retweet score.

The second important remark is that a ranking model is also needed to achieve the best NDCG score. LambdaMART is used by any model with an NDCG greater than 0.87, and the simple solution *Mean(λ-MART, Retweet)* has almost the best NDCG score. However, the wrapping strategy allows regression models to outclass LambdaMART. This suggests that ranking model are more promising as soon as the data are cleaned from the retweet effect.

Notice also that the linear combination of simple models works surprisingly well. LambdaMART is a ranking model, but it is based on a boosted Forest which predicts the inclination for tweet to have a better score than an other one. Then, LambdaMART mixes well with Random Forests or with Retweet.

Finally, Figure 5.8 gives the NDCG after the linear combination of LambdaMART, WrapRF and WrapLin. We observe a large plateau of NDCG around the arithmetic mean. Moreover, the careful selection of the linear combination of simple models only increases the NDCG to an extent of a few thousandth. This tiny increase in NDCG, and the large plateau of equivalent linear combination indicates that the arithmetic mean is a safer approach. A similar analysis of OptAvg(λ-MART, RF, Retweet) leads to the same conclusion.

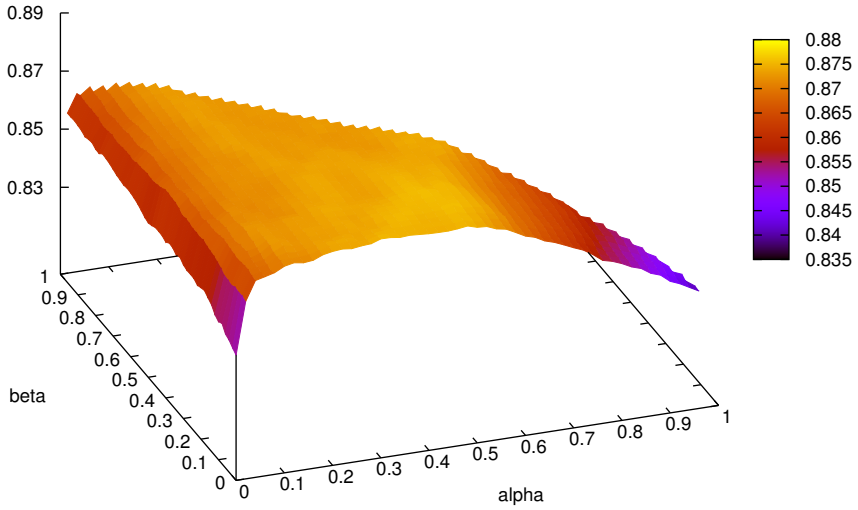


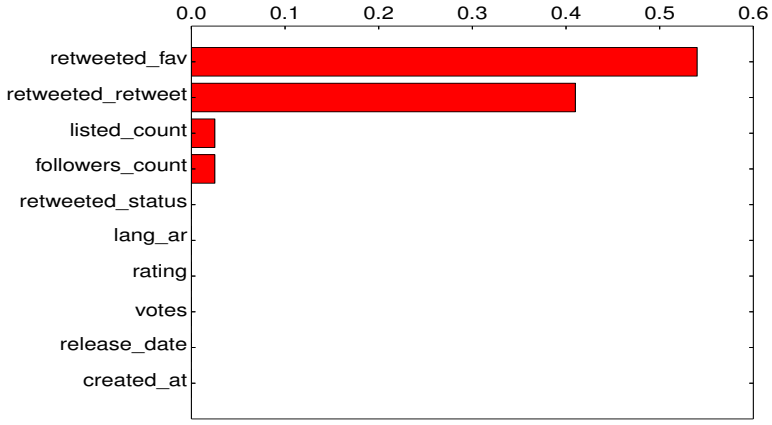
FIGURE 5.8: NDCG@10 on test set, from the linear combination of LambdaMART, WrapRF and WrapLin. Their weight is respectively α, β and $1 - \alpha - \beta$.

Relevant Features

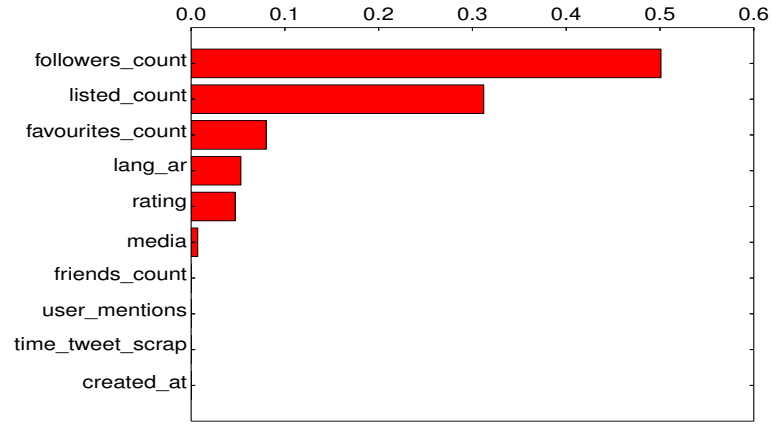
The relevance of features has been measured on different sets of Random Forests to observe which effect has the removal of one or several features. In order to measure the relevance, we can use the relative depth at which the feature appears in a tree. A feature which appears at the top is contributing more to the final prediction as there is a larger fraction of input samples going through this node. The estimate used to measure the importance of a feature is thus the expected fraction of the samples to which this feature will contribute. Figure 5.9a presents the feature relevance on original data: we keep all input features, including the retweet count of the original tweet, and we do not modify the user engagement of the tweet. We notice in this case the features related to the original tweet, such as the number of favorites or retweet of the original tweet are the features contributing the most to the prediction. This contribution again highlights the importance of the original retweet count to build an efficient model.

Figure 5.9b exhibits the relevance of features on data cleaned from the retweet effect: the retweet count of the original tweet is removed from input features and user engagement is modified as mentioned previously. We observe that after removal of the retweet effect, the Random Forest makes a prediction about the user engagement mostly based on user features, such as the number of followers, the number of favorites, or the number of lists in which the user is. A user who has a higher number of followers or is included in more lists is more likely to receive higher engagement on his

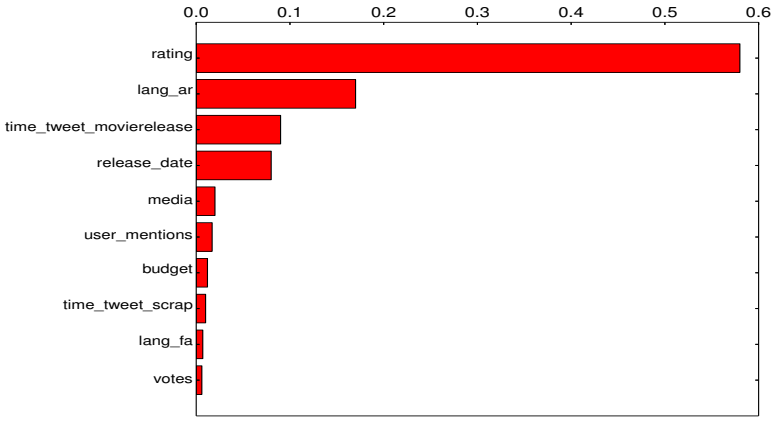
[2]The best linear combinations are
$0.48\times\lambda$-MART + $0.08\times$WrapRF + $0.44\times$WrapLin and
$0.3\times\lambda$-MART + $0.12\times$RF + $0.58\times$Retweet



(A) Relevance of features using all features and unmodified user engagement.



(B) Relevance of features after cleaning the retweet effect.



(C) Relevance of features after cleaning the retweet effect and removing user features.

FIGURE 5.9: Relevance of features.

tweets. However, focusing mostly on user features sounds inappropriate for the challenge, as the tweets to rank are always emitted by the same user. While comparing two tweets, the only difference in user properties comes from the fact that both tweets were not emitted at the same date.

In the last case (cf. Figure 5.9c), we evaluate the importance of all features except user features. Since the NDCG score is computed for each user and user features seem not to change a lot along time, it would be interesting to observe which movie or tweet features are influencing the regression prediction. Remark that the most relevant features sound reasonable: the rating given by the user in the tweet, one language feature (arabic), the time passed between the release of the movie and the tweet post, the date at which the movie was released.

Discussions

We discuss on this section some conclusions that can be drawn from results of the approach in the Challenge and which can be generalized to most real-world RS.

Problems are often simple, be smart A lot of data was available for the Challenge, but a simple attribute of the input data decided of most of the score obtained by the algorithm. In the era of Big Data, a lot of complex algorithms are built trying to make use of most of the data available. However, problems can often be solved using some specific attributes which give most of the information.

No need to use all examples Our approach in this Challenge only used some examples to build its model (the training set for LambdaMART was really small), as others would mostly bring noise or no information at all. Reducing the size of the data on a relevant subset of examples is necessary to build a significant model in a short amount of time, and saving time during the learning of the model is a valuable characteristic for a real-world RS.

Build simple models Recommendation problems are often considered as Learning to Rank problem, but sometimes building a simple model and focusing on the prediction only is sufficient to reach a good performance. Here the Random Forest model alone reaches already a good performance compared to LambdaMART. The same remark can be done about Chapter 3, where our simple approach SeALS achieves better results than PTS, and also about Chapter 4, where Learning to Rank approaches actually perform worse than a simple Matrix Factorization approach like ALS.

Do not forget temporal and sequential aspect Several temporal aspects can influence the model prediction and evaluation, and the importance of

the sequential aspect has been displayed in previous Chapters. In this Challenge, some important temporal or sequential features of the problem were forgotten:

- The time interval from which the tweets were posted for both the train set and the test/evaluation set were strongly different, even if a user has more chance throughout a long period of time to gather new followers who could bring potentially a higher engagement on his future tweets.
- Movies corresponding to tweets with the highest engagement were mostly popular movies which have been released recently at the time the tweet was extracted. Taking a small interval of time for a dataset can also modify the structure of data on this part since popular movies are often released at a specific period of time during the year.
- Finally, the evolution of a user in the Twitter environment, or the release of movies can be seen as sequential problems, but this challenge did not consider any sequential context for this Challenge.

5.4 Concluding Remarks

We study in this chapter some aspects and challenges of real-world Recommendation Systems. The first part described several important characteristics of real RS. We integrated some of these characteristics in an offline sequential evaluation setting, more realistic than the one in Chapter 3. Results show that some simple aspects of real-world RS can make the sequential evaluation far more complex to set offline, and highlight the need for datasets adapted to perform such tasks. Finally, our results on a RS Challenge on real-world data also emphasizes that smart and simple models are the target to aim at when building a real-world RS, but that these models need to be built while considering the temporal aspects and the inherent sequential context in which RS perform their recommendations.

Chapter 6

Conclusion

In this thesis, we focused on the recommendation problem seen as a sequential process, where recommendations are provided to the user one after another and feedback is gathered progressively through time. By changing the perspective into which Recommendation Systems are usually studied, several crucial aspects that the system needs to target appear.

6.1 Thesis Contributions

We focused on two aspects resulting from the sequential setting, by considering two different possible recommendation process:

- In Chapter 3, we focused on a setting where one item is recommended by the RS at each time step to the user, and the feedback is gathered after each recommendation. We made the relation of this setting to the Multi-Armed Bandit setting, where each item is a possible arm to recommend to the user, and showed that the exploration-exploitation dilemma raises in the recommendation problem. We proposed a simple and efficient approach to provide appropriate recommendations by combining a Collaborative Filtering method with a Multi-Armed Bandit approach and evaluated the system in a proper sequential offline setting compared to fixed batch setting. Empirical studies showed that embedding the exploration vs. exploitation dilemma into the algorithm helps the system to maximize the ratings received by the system through time. Finally, since the system also needs to learn through time, we also studied the model update in order to make the system able to provide recommendations as accurate as possible without losing too much time to update itself.
- In Chapter 4, we extended the sequential setting to a recommendation process where several items are suggested to the user at each time step. The RS has to optimize the order of the items in the recommended list, in order to put items with highest chances of matching the user's tastes at the top of the list. In the case of a list of items, the feedback provided by the user can take the form of explicit rating about the chosen item, but also the form of implicit feedback on

items that were chosen or not in the list. The implicit feedback also provides some sort of information about which item is preferred to another through the click. By choosing a case study and modeling the user's interaction with the system through a click model, we then proposed two methods to integrate both explicit and implicit feedback into the model, to learn more quickly about the user's tastes and provide better ordering of the recommended items. Empirical studies on several datasets showed that the inclusion of both types of feedback indeed allows better performance. Experiments with other click models also confirm the necessity of using implicit feedback, but also highlight the need for caution in the way the implicit feedback is included and the weight that is given to it.

- In Chapter 5, we discussed about some challenges in real-world RS. We first described some important aspects observed in real RS, and made the link between these aspects and the sequential offline evaluation. Then, we modified the setting established in Chapter 3 to make it more realistic, by applying a power-law distribution for the arrival of users into the RS, and by forbidding the system to recommend the same item several times to a user. The evaluation in such sequential setting highlighted the difficulty of evaluating RS in a sequential context only based on offline datasets. Finally, some results obtained on a real-world dataset during a RS Challenge emphasized that having a lot of data and searching for a complex model to perform recommendation is sometimes misleading.

The aspects treated in chapters above, which are the exploration vs. exploitation dilemma, the update of the model, and the integration of multiple type of feedback are emphasized due to the sequential evaluation settings set up in this thesis, and are central issues that need to be targeted in real-world RS.

6.2 Future Work

There are numerous directions in which the work in this thesis can be extended. The first direction is related to the extension of our approaches to integrate external information, the second one regards the integration of Multi-Armed bandits algorithms in the recommendation process to solve the cold start issue, and finally, the last two directions target the improvement of the RS when multiple items are received, either through the improvement of the click model integration, or through explore-exploit strategies.

Mix feedback information with contextual or external information

In this whole thesis, we considered only systems learning from explicit feedback (in Chapter 3) or from a mix of explicit and implicit feedback (in Chapter 4). There have been a lot of research in RS about adding external information about the users or the items, or including contextual information

to improve the performance of the system or reduce the cold start effect. Such information could actually easily be included into the Factorization Model used in Chapter 4 by simply adding the additional information into the input feature vector.

Extend RS algorithms using Multi-Armed Bandits approach for the cold start on users and/or items

We developed in Chapter 3 a simple and efficient approach to tackle the explore-exploit dilemma in RS. However, more complex approaches would allow to judge the uncertainty on users and items and use this information to improve performance. In particular, we focused in the experimental setting on the user side, where the RS tries to recommend items matching the user's tastes as much as possible, but the decrease of the uncertainty from the items' perspective should also be considered (gather information about items quickly to recommend them to adequate users in the future). The combined use of uncertainty on both the users and the items to tackle user and items cold start would allow the system to perform better recommendation.

Infer click model in RS recommending several items

We built in Chapter 4 one approach (DualMF) directly based on a click model but the performance of this approach directly depends on the click model used in the offline evaluation to represent the interaction between the user and the system. A more appropriate solution would be to infer the click model directly as new feedback is received, to incorporate the implicit feedback correctly into the model.

Incorporate explore-exploit strategies into the ranking problem for multiple-recommendations RS

As seen in Chapter 3, a sequential setting also raises concerns about finding good balance between gathering information about the user and providing good recommendations, a.k.a. the exploration-exploitation dilemma. We integrated an approach which considers this issue in Chapter 4, in the case where multiple items are recommended but results showed that exploration vs. exploitation dilemma needs to be carefully included in this context. The inclusion of explore-exploit strategies when multiple items are recommended seems to be the most promising field of research: some implicit goals of ranking RS and explore-exploit are indeed tightly connected. For example, explore-exploit strategies aim at gathering information on items and user for which little knowledge is known, while suggesting a diverse list of recommendations is usually advised for ranking RS. It is likely that using both the implicit or explicit feedback given by user on a diverse set of items in only one recommendation step would increase greatly the knowledge about both the user and the items. However, the integration of explore-exploit strategies in ranking approaches is almost an empty field:

should the exploration be emphasized at the beginning or at the end of the list of items? Should it be spread along the whole list? These questions are left for future research.

Appendix A

About UCB1 and Popular baselines

This Appendix gives a simple example about the UCB1 and Popular baselines, to explain why UCB1 does not necessarily converge to the solution given by Popular in the setting described in Chapter 3.

Let us consider the matrix of ratings $\mathbf{R}$ with 3 users $\{u_1, u_2, u_3\}$ and 3 items $\{i_1, i_2, i_3\}$:

$$\mathbf{R} = \begin{pmatrix} 1 & NA & 2 \\ 5 & NA & 5 \\ NA & 5 & 1 \end{pmatrix}$$

where each line represents a user, each column represents an item, and NA represents an unknown rating.

The average rating for each item is respectively 3, 5 and $2.\overline{66}$ for i_1, i_2 and i_3. Starting from an empty matrix, the Popular strategy yet knows in advance the average rating for all items, and it will recommend accordingly to users (among items for which the rating exists in the dataset). In this case, this means that Popular will always recommend i_1 for u_1, i_1 for u_2 and i_2 for u_3. This action will bring a regret of 1 each time a recommendation is made for u_1, as the best possible recommendation would be i_3.

On the contrary, UCB1 starts with no knowledge about the items' mean ratings. In this example, UCB1 will play items with no prior knowledge, and eventually find that it is a better strategy to play i_3 for both u_1 and u_2 (and play i_2 for u_3). The average rating for Popular gives an equal weight to every rating made by any user (it considers every user with a rating on an item in the dataset played this item one time only), while for UCB1, the mean rating to which it will converge will be different, as the item i_3 will be played many times for user u_1 and u_2, but almost never for u_3.

We display in Figure A.1 the cumulative regret (over 50runs) for 300 recommendations over this example. We also display on the curves the standard deviation of the cumulative regret.

We observe that UCB1 indeed reaches a lower cumulative regret than Popular. The standard deviation of UCB1 is also much larger than the one in Popular: this is because the time necessary for the algorithm to discover that i_3 is a better choice than i_1 for both u_1 and u_2 will depend on the order

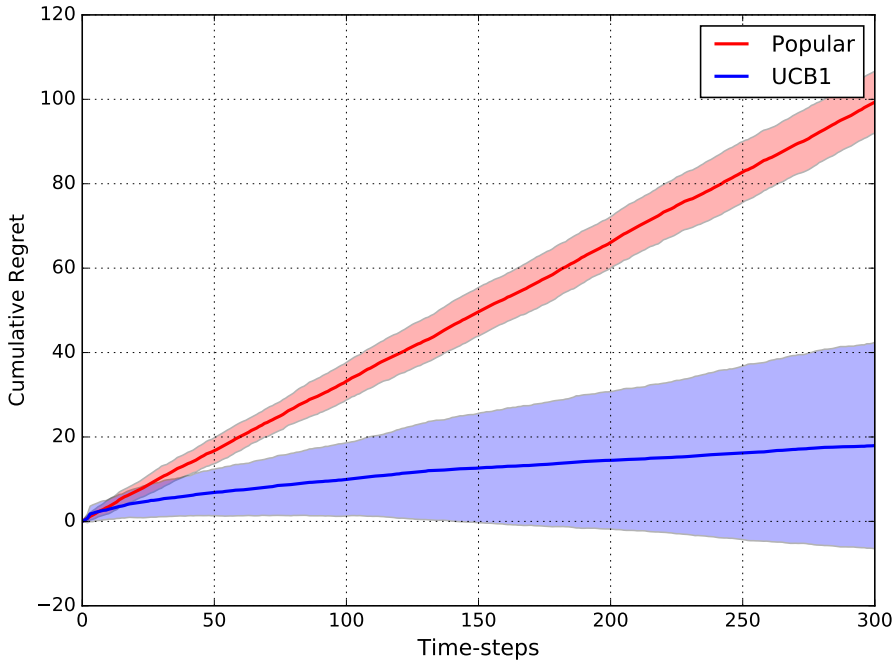


FIGURE A.1: An example to compare UCB1 and Popular approaches.

of the users selected to receive the recommendation, and this selection is done randomly (for example, if u_1 is selected often at the beginning of the evaluation, the algorithm will find out quickly that playing i_3 is a better strategy overall than i_1).

Bibliography

Adomavicius, G., Sankaranarayanan, R., Sen, S., and Tuzhilin, A. (2005). Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems (TOIS)*, 23(1):103–145.

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749.

Adomavicius, G. and Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer.

Agarwal, D., Chen, B.-C., Elango, P., Motgi, N., Park, S.-T., Ramakrishnan, R., Roy, S., and Zachariah, J. (2008). Online models for content optimization. In *Advances in Neural Information Processing Systems, Proc. of the 22nd Annual Conference on Neural Information Processing Systems (NIPS)*, pages 17–24.

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

Ahn, H. J. (2008). A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51.

Audibert, J.-Y., Munos, R., and Szepesvári, C. (2009). Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.*, 410(19):1876–1902.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256.

Balakrishnan, S. and Chopra, S. (2012). Collaborative ranking. In *Proc. of the fifth ACM int. conf. on Web Search and Data Mining (WSDM'12)*, pages 143–152. ACM.

Bennett, J., Lanning, S., and Netflix (2007). The Netflix prize. In *KDD Cup and Workshop*.

Bhagat, S., Weinsberg, U., Ioannidis, S., and Taft, N. (2014). Recommending with an agenda: Active learning of private attributes using matrix factorization. In *Proc. of the 8th ACM Conf. on Recommender Systems (RecSys'14)*, pages 65–72.

Bottou, L. and Bousquet, O. (2007). The tradeoffs of large scale learning. In *Proc. of the 21st Annual Conf. on Neural Information Processing Systems (NIPS'07)*, pages 161–168.

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

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brozovsky, L. and Petricek, V. (2007). Recommender system for online dating service. In *Proceedings of Conference Znalosti 2007*, Ostrava. VSB.

Bubeck, S. and Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *CoRR*, abs/1204.5721.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proc. of the 22nd int. conf. on Machine Learning (ICML'05)*, pages 89–96. ACM.

Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. *Microsoft Research Technical Report MSR-TR-2010-82*.

Burges, C. J., Svore, K. M., Bennett, P. N., Pastusiak, A., and Wu, Q. (2011). Learning to rank using an ensemble of lambda-gradient models. In *Yahoo! Learning to Rank Challenge*, pages 25–35.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370.

Campos, P. G., Diez, F., and Cantador, I. (2014). Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1-2):67–119.

Celma, Ò. and Herrera, P. (2008). A new approach to evaluating novel recommendations. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 179–186. ACM.

Chai, T. and Draxler, R. R. (2014). Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250.

Chapelle, O. and Chang, Y. (2011). Yahoo! learning to rank challenge overview. In *Yahoo! Learning to Rank Challenge*, pages 1–24.

Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Proc. of the 25th Annual Conference on Neural Information Processing Systems (NIPS'11)*, pages 2249–2257. Curran Associates, Inc.

Chapelle, O., Metlzer, D., Zhang, Y., and Grinspan, P. (2009). Expected reciprocal rank for graded relevance. In *Proc. of the 18th ACM conf. on Information and knowledge management (CIKM'09)*, pages 621–630. ACM.

Chen, K., Chen, T., Zheng, G., Jin, O., Yao, E., and Yu, Y. (2012). Collaborative personalized tweet recommendation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 661–670. ACM.

Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., and Sartin, M. (1999). Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR workshop on recommender systems*, volume 60. Citeseer.

Cremonesi, P., Koren, Y., and Turrin, R. (2010). Performance of recommender algorithms on top-N recommendation tasks. In *Proc. of RecSys'10*, pages 39–46.

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

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

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

Dooms, S., De Pessemier, T., and Martens, L. (2013). Movietweetings: a movie rating dataset collected from twitter. In *Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys*, volume 2013.

Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218.

Ekstrand, M. D., Riedl, J. T., and Konstan, J. A. (2011). Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

Fouss, F., Pirotte, A., Renders, J.-M., and Saerens, M. (2007). Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering*, 19(3):355–369.

Garcin, F., Faltings, B., Donatsch, O., Alazzawi, A., Bruttin, C., and Huber, A. (2014). Offline and online evaluation of news recommender systems at swissinfo.ch. In *Proc. of RecSys'14*, pages 169–176. ACM.

Garivier, A. and Cappé, O. (2011). The KL-UCB algorithm for bounded stochastic bandits and beyond. In *Proc. of the 24^{th} Annual Conference on Learning Theory (COLT'11)*, pages 359–376.

Ge, M., Delgado-Battenfeld, C., and Jannach, D. (2010). Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 257–260. ACM.

Gemulla, R., Nijkamp, E., Haas, P. J., and Sismanis, Y. (2011). Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM.

George, T. and Merugu, S. (2005). A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 4–pp. IEEE.

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192.

Guillou, F., Gaudel, R., Mary, J., and Preux, P. (2014). User engagement as evaluation: a ranking or a regression problem? In *Proceedings of the 2014 Recommender Systems Challenge*, page 7. ACM.

Guillou, F., Gaudel, R., and Preux, P. (2015). Collaborative filtering as a multi-armed bandit. In *NIPS'15 Workshop: Machine Learning for eCommerce*.

Guillou, F., Gaudel, R., and Preux, P. (2016a). Large-scale bandit recommender systems. In *Proc. of the Second International Workshop on Machine Learning, Optimization and Big Data (MOD)*, LNCS. Springer.

Guillou, F., Gaudel, R., and Preux, P. (2016b). Scalable explore-exploit collaborative filtering. In *Proc. of the 20th Pacific Asia Conference on Information Systems (PACIS)*.

Guillou, F., Gaudel, R., and Preux, P. (2016c). Sequential collaborative ranking using (no-)click implicit feedbacks. In *Proc. of the 23rd International Conference on Neural Information Processing (ICONIP)*, LNCS. Springer.

Guo, F., Li, L., and Faloutsos, C. (2009a). Tailoring click models to user goals. In *Proceedings of the 2009 workshop on Web Search Click Data*, pages 88–92. ACM.

Guo, F., Liu, C., and Wang, Y. M. (2009b). Efficient multiple-click models in web search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 124–131. ACM.

Harper, F. M. and Konstan, J. A. (2015). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19.

Herlocker, J., Konstan, J. A., and Riedl, J. (2002). An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information retrieval*, 5(4):287–310.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53.

Hofmann, K. et al. (2013a). *Fast and reliable online learning to rank for information retrieval*. PhD thesis, University of Amsterdam.

Hofmann, K., Whiteson, S., and de Rijke, M. (2013b). Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval*, 16(1):63–90.

Hofmann, T. (2004). Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115.

Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee.

Jahrer, M., Töscher, A., and Legenstein, R. (2010). Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 693–702. ACM.

Jain, P., Netrapalli, P., and Sanghavi, S. (2013). Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 665–674. ACM.

Karatzoglou, A., Amatriain, X., Baltrunas, L., and Oliver, N. (2010). Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM.

Kaufmann, E., Korda, N., and Munos, R. (2012). Thompson sampling: An asymptotically optimal finite-time analysis. In Bshouty, N., Stoltz, G., Vayatis, N., and Zeugmann, T., editors, *Algorithmic Learning Theory*, volume 7568 of *Lecture Notes in Computer Science*, pages 199–213. Springer Berlin Heidelberg.

Kawale, J., Bui, H., Kveton, B., Thanh, L. T., and Chawla, S. (2015). Efficient thompson sampling for online matrix-factorization recommendation. In *NIPS'15*.

Kendall, M. G. (1948). *Rank correlation methods.* Griffin.

Kim, H. and Park, H. (2008). Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM journal on matrix analysis and applications*, 30(2):713–730.

Kohli, P., Salek, M., and Stoddard, G. (2013). A fast bandit algorithm for recommendation to users with heterogenous tastes. In *Proc. of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013)*. AAAI.

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM.

Koren, Y. (2010). Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Koren, Y. and Sill, J. (2011). Ordrec: An ordinal model for predicting personalized item rating distributions. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 117–124, New York, NY, USA. ACM.

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

Kurucz, M., Benczúr, A. A., and Csalogány, K. (2007). Methods for large scale svd with missing values. In *Proceedings of KDD cup and workshop*, volume 12, pages 31–38. Citeseer.

Langford, J., Strehl, A., and Wortman, J. (2008). Exploration scavenging. In McCallum, A. and Roweis, S., editors, *Proc. of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 528–535. Omnipress.

Lee, D. D. and Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562.

Lee, J., Bengio, S., Kim, S., Lebanon, G., and Singer, Y. (2014). Local collaborative ranking. In *Proc. of the 23rd int. conf. on World Wide Web (WWW'14)*, pages 85–96.

Levandoski, J. J., Sarwat, M., Eldawy, A., and Mokbel, M. F. (2012). Lars: A location-aware recommender system. In *2012 IEEE 28th International Conference on Data Engineering*, pages 450–461. IEEE.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proc. of the 19th international conference on World Wide Web (WWW)*, pages 661–670, New York, NY, USA. ACM.

Li, L., Chu, W., Langford, J., and Wang, X. (2011). Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proc. of Web Search and Data Mining (WSDM'11)*, pages 297–306. ACM.

Linden, G., Smith, B., and York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80.

Liu, H., Hu, Z., Mian, A., Tian, H., and Zhu, X. (2014). A new user similarity model to improve the accuracy of collaborative filtering. *Knowledge-Based Systems*, 56:156–166.

Liu, N. N., Xiang, E. W., Zhao, M., and Yang, Q. (2010). Unifying explicit and implicit feedback for collaborative filtering. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1445–1448. ACM.

Liu, T.-Y. (2009). Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331.

Liu, X. and Aberer, K. (2014). Towards a dynamic top-n recommendation framework. In *Proc. of the 8th conf. on Recommender Systems (RecSys'14)*, pages 217–224.

Loiacono, D., Lommatzsch, A., and Turrin, R. (2014). An analysis of the 2014 recsys challenge. In *Proceedings of the 2014 Recommender Systems Challenge*, page 1. ACM.

Lops, P., De Gemmis, M., and Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer.

Luo, X., Xia, Y., and Zhu, Q. (2012). Incremental Collaborative Filtering recommender based on Regularized Matrix Factorization. *Knowledge-Based Systems*, 27:271 – 280.

Ma, H., Zhou, D., Liu, C., Lyu, M. R., and King, I. (2011). Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM.

Mahmood, T. and Ricci, F. (2009). Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, pages 73–82. ACM.

Mary, J., Gaudel, R., and Preux, P. (2015). Bandits and Recommender Systems. In *Proc. of International Workshop on Machine Learning, Optimization and big Data (MOD'15)*, LNCS, Taormina, Sicily, Italy. Springer.

McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1097–1101. ACM.

Miller, B. N., Albert, I., Lam, S. K., Konstan, J. A., and Riedl, J. (2003). Movielens unplugged: experiences with an occasionally connected recommender system. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 263–266. ACM.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mobasher, B., Burke, R., Bhaumik, R., and Williams, C. (2007). Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology (TOIT)*, 7(4):23.

Murakami, T., Mori, K., and Orihara, R. (2007). Metrics for evaluating the serendipity of recommendation lists. In *Annual Conference of the Japanese Society for Artificial Intelligence*, pages 40–46. Springer.

Nakamura, A. (2014). A ucb-like strategy of collaborative filtering. In *Proc. of ACML'14*.

Oard, D. W., Kim, J., et al. (1998). Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83.

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8.

Pauca, V. P., Piper, J., and Plemmons, R. J. (2006). Nonnegative matrix factorization for spectral data analysis. *Linear algebra and its applications*, 416(1):29–47.

Pazzani, M. J. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5-6):393–408.

Pazzani, M. J. and Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer.

Radlinski, F., Kleinberg, R., and Joachims, T. (2008). Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, pages 784–791. ACM.

Recht, B., Re, C., Wright, S., and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701.

Rendle, S. (2010). Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE.

Rendle, S. (2012). Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press.

Rendle, S., Gantner, Z., Freudenthaler, C., and Schmidt-Thieme, L. (2011). Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644. ACM.

Rendle, S. and Schmidt-Thieme, L. (2008). Online-updating Regularized Kernel Matrix Factorization Models for Large-scale Recommender Systems. In *Proc. of ACM Conference on Recommender Systems (RecSys'08)*, RecSys '08, pages 251–258, New York, NY, USA. ACM.

Rennie, J. D. and Srebro, N. (2005). Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM.

Resnick, P. and Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3):56–58.

Rubens, N., Kaplan, D., and Sugiyama, M. (2011). Active learning in recommender systems. In *Recommender systems handbook*, pages 735–767. Springer.

Said, A. and Bellogin, A. (2014). Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 129–136. ACM.

Said, A., Dooms, S., Loni, B., and Tikk, D. (2014). Recommender systems challenge 2014. In *Proceedings of the eighth ACM conference on Recommender systems*, RecSys '14, New York, NY, USA. ACM.

Salah, A., Rogovschi, N., and Nadif, M. (2015). An efficient incremental collaborative filtering system. In *International Conference on Neural Information Processing*, pages 375–383. Springer.

Salakhutdinov, R. and Mnih, A. (2008a). Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887.

Salakhutdinov, R. and Mnih, A. (2008b). Probabilistic matrix factorization. In *Proc. of NIPS'08*.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM.

Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007). Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer.

Schafer, J. B., Konstan, J. A., and Riedl, J. (2001). E-commerce recommendation applications. In *Applications of Data Mining to Electronic Commerce*, pages 115–153. Springer.

Shani, G. and Gunawardana, A. (2011). Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer.

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

Shapira, B., Ricci, F., Kantor, P. B., and Rokach, L. (2011). *Recommender Systems Handbook.* Springer.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., and Hanjalic, A. (2013). xclimf: optimizing expected reciprocal rank for data with multiple levels of relevance. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 431–434. ACM.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N., and Hanjalic, A. (2012). CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proc. of RecSys'12*, pages 139–146.

Shi, Y., Larson, M., and Hanjalic, A. (2010). List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 269–272. ACM.

Shi, Y., Larson, M., and Hanjalic, A. (2014). Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)*, 47(1):3.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Srebro, N., Rennie, J., and Jaakkola, T. S. (2004). Maximum-margin matrix factorization. In *Advances in neural information processing systems*, pages 1329–1336.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. Mit Press.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103.

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2007). Major components of the gravity recommendation system. *ACM SIGKDD Explorations Newsletter*, 9(2):80–83.

Takács, G. and Tikk, D. (2012). Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 83–90. ACM.

Tang, L., Jiang, Y., Li, L., and Li, T. (2014). Ensemble contextual bandits for personalized recommendation. In *Proc. of ACM Conf. on Recommender Systems (RecSys'14)*, RecSys'14, Foster City, Silicon Valley, USA. ACM.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

Trewin, S. (2000). Knowledge-based recommender systems. *Encyclopedia of library and information science*, 69(Supplement 32):180.

Ungar, L. H. and Foster, D. P. (1998). Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, volume 1, pages 114–129.

Vargas, S. and Castells, P. (2011). Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 109–116. ACM.

Volkovs, M. and Zemel, R. S. (2012). Collaborative ranking with 17 parameters. In *Advances in Neural Information Processing Systems*, pages 2294–2302.

Wang, Y., Wang, L., Li, Y., He, D., Chen, W., and Liu, T.-Y. (2013). A theoretical analysis of ndcg ranking measures. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT 2013)*.

Weimer, M., Karatzoglou, A., Le, Q. V., and Smola, A. J. (2008). Cofi rank - maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems 20*, pages 1593–1600.

Weston, J., Yee, H., and Weiss, R. J. (2013). Learning to rank recommendations with the k-order statistic loss. In *Proceedings of the 7th ACM conference on Recommender systems (RecSys 2013)*, pages 245–248. ACM.

Willmott, C. J. and Matsuura, K. (2005). Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82.

Wu, M. (2007). Collaborative filtering via ensembles of matrix factorizations. In *Proceedings of KDD Cup and Workshop*, volume 2007.

Wu, Q., Burges, C. J., Svore, K. M., and Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270.

Xing, Z., Wang, X., and Wang, Y. (2014). Enhancing Collaborative Filtering Music Recommendation by Balancing Exploration and Exploitation. In *Proc. of the 15th International Society for Music Information Retrieval Conference (ISMIR'14)*, pages 445–450, Taipei, Taiwan.

Xiong, L., Chen, X., Huang, T.-K., Schneider, J., and Carbonell, J. G. (2010). Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the SIAM International Conference on Data Mining (SIAM)*, pages 211–222. SIAM.

Yang, W.-S., Cheng, H.-C., and Dia, J.-B. (2008). A location-aware recommender system for mobile shopping environments. *Expert Systems with Applications*, 34(1):437–445.

Yue, Y. and Joachims, T. (2009). Interactively optimizing information retrieval systems as a dueling bandits problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1201–1208. ACM.

Zhang, M. and Hurley, N. (2008). Avoiding monotony: improving the diversity of recommendation lists. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 123–130. ACM.

Zhang, S., Wang, W., Ford, J., and Makedon, F. (2006). Learning from incomplete ratings using non-negative matrix factorization. In *SDM*, volume 6, pages 548–552. SIAM.

Zhao, X., Zhang, W., and Wang, J. (2013). Interactive collaborative filtering. In *CKIM'13*, pages 1411–1420.

Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R., and Zhang, Y.-C. (2010). Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515.

Zhou, Y., Wilkinson, D., Schreiber, R., and Pan, R. (2008). Large-scale parallel collaborative filtering for the netflix prize. In *Proc. of the 4th international conference on Algorithmic Aspects in Information and Management (AAIM)*, pages 337–348, Berlin, Heidelberg. Springer-Verlag.

Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G. (2005). Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM.