Managing recommendation data in large scale

Modou Gueye

2014-ENST-0083

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et réseaux »

présentée et soutenue publiquement par

Modou GUEYE

le 15 Décembre 2014

Gestion de données de recommandation à très large échelle

Directeur de thèse : **Talel ABDESSALEM**
Encadrant de thèse : **Hubert NAACKE**

T H È S E

TELECOM ParisTech
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

Abstract

In this thesis, we address the scalability problem of recommender systems. We propose accurate and scalable algorithms. We first consider the case of matrix factorization techniques in a dynamic context, where new ratings are continuously produced. In such case, it is not possible to have an up to date model, due to the incompressible time needed to compute it. This happens even if a distributed technique is used for matrix factorization. At least, the ratings produced during the model computation will be missing. Our solution reduces the loss of the quality of the recommendations over time, by introducing some stable biases which track users' behavior deviation. These biases are continuously updated with the new ratings, in order to maintain the quality of recommendations at a high level for a longer time.

We also consider the context of online social networks and tag recommendation. We propose an algorithm that takes into account the popularity of the tags and the opinions of the users' neighborhood. But, unlike common nearest neighbors' approaches, our algorithm does not rely on a fixed number of neighbors while computing a recommendation. It uses a heuristic that bounds the network traversal in a way that enables computing the recommendations on the fly, with a limited computation cost, while preserving the quality of the recommendations.

Finally, we propose a novel approach that improves the accuracy of the recommendations for top-k algorithms. Instead of a fixed list size, we adjust the number of items to recommend in a way that optimizes the global accuracy of the recommendations. We other words, we optimize the likelihood that all the recommended items will be chosen by the user, and find the best candidate sub-list (i.e., the most accurate one) to recommend to the user.

Keywords
Recommender systems, Scalability, Dynamicity, Collaborative filtering, Matrix Factorization, Social networks, Tag recommendation.

Contents

Contents **v**

List of Figures **viii**

List of Tables **ix**

General Introduction **1**

I A survey of recommender systems **3**

1 Introduction to Recommender Systems **5**

1.1 Formal Definition of Recommendation Task 5
1.2 Recommender Systems Function 6
1.3 Classification of Recommender Systems 7
1.3.1 Content-based Recommender Systems 7
1.3.1.1 Item Representation 8
1.3.1.2 User Profile 9
1.3.1.3 Advantages and limitations 12
1.3.2 Collaborative Filtering Systems 13
1.3.2.1 Memory-based CF 14
1.3.2.2 Model-based CF 16
1.3.2.3 Advantages and limitations 20
1.3.3 Hybrid Systems 21
1.4 Conclusion 22

2 Challenges and Evaluation of Recommender Systems **23**

2.1 Challenges of Recommender Systems 23
2.1.1 Data Sparsity 23
2.1.2 Scalability 24
2.1.3 Diversity vs. accuracy 25
2.1.4 User interface 25
2.1.5 Vulnerability to attacks 26
2.1.6 Some other Challenges 26

2.2 Evaluating Recommender Systems . . . 27
2.2.1 Evaluation approaches . . . 27
2.2.1.1 Offline evaluation . . . 27
2.2.1.2 User studies . . . 28
2.2.1.3 Online Evaluation . . . 28
2.2.2 Evaluation measures . . . 29
2.2.2.1 Prediction Accuracy . . . 29
2.2.2.2 Coverage . . . 32
2.2.2.3 Diversity . . . 33
2.2.2.4 Novelty . . . 34
2.3 Conclusion . . . 34

II The contributions of the thesis 37

3 Using Cluster-based Biases for Dynamic Recommendations 39
3.1 Preliminaries . . . 40
3.2 The Dynamicity Problem . . . 43
3.3 Related Work . . . 44
3.3.1 Distributed MF techniques . . . 45
3.3.2 Online-updating approaches . . . 46
3.4 Making Dynamic Recommendations . . . 47
3.4.1 Biased MF . . . 48
3.4.2 The interest of cluster-based local biases . . . 48
3.4.3 The CBMF model . . . 50
3.4.4 Integration of incoming ratings . . . 51
3.5 Complexity analysis . . . 53
3.6 Experimental Results . . . 54
3.6.1 Implementation and experimental setup . . . 54
3.6.2 Datasets . . . 55
3.6.3 Initial quality . . . 55
3.6.4 Large training sets improve the quality of the model . . . 56
3.6.5 Quantifying the need for online integration . . . 58
3.6.6 Robustness over time of our online integration model . . . 58
3.6.7 Quality vs. Performance tradeoff for online integration . . . 59
3.6.8 Benefit of refactorization . . . 60
3.7 Conclusion . . . 62

4 Making Social and Popularity-based Tag Recommendations 65
4.1 Preliminaries . . . 66
4.1.1 Similarity propagation . . . 67
4.1.2 Extended neighborhood opinion . . . 68
4.2 Social and Popularity-based Tag Recommendation . . . 69
4.2.1 Score model and tag relevance . . . 69

4.2.2 The FasTag Algorithm 70
4.2.3 Handling the Network Partitioning 73
4.3 Related Work 73
4.4 Experimentation 74
4.4.1 Datasets 74
4.4.2 Evaluation Measures and Methodology 74
4.4.3 Effectiveness of FasTag 75
4.4.4 Comparison with the result of ECML PKDD challenge 09 76
4.4.5 Scalability of FasTag 78
4.5 Conclusion 79

5 Optimizing Tag Recommendation List Size 81
5.1 Preliminaries 82
5.1.1 Factor Models for Tag Recommendation 83
5.1.2 FolkRank - A Topic-Specific Ranking 83
5.1.3 Recommending the Most Popular Tags 84
5.1.4 Social and popularity-based Recommender 84
5.2 Adjusted Recommendation list size 85
5.2.1 Linear combination models 85
5.2.2 The blsC algorithm 86
5.2.2.1 Simple relevance measure 87
5.2.2.2 Refining the relevance measure 87
5.3 Experimentation 89
5.3.1 Datasets 89
5.3.2 Evaluation Measures and Methodology 89
5.3.3 Experimental Results 90
5.3.3.1 Effectiveness of our proposal 90
5.3.3.2 Giving up some recall for more precision 91
5.3.3.3 Distribution of optimal list sizes 92
5.4 Conclusion 92

Research Perspectives 97

Self References 99

Bibliography 101

Appendices 115

A Résumé en Français 117

List of Figures

1.1 A romance-and-action space of movies and users 17
1.2 Graphical presentation of neuron 18
1.3 A multi-layer perceptron for collaborative filtering 19

3.1 Amazon's five star widget 40
3.2 Patterns of independent blocks for a 3-by-3 gridded matrix 45
3.3 Training sets partitioning 57
3.4 Quality improvement for increasing training sets sizes 57
3.5 Offline quality (RMSE) with increasing delays (# million of ratings) 59
3.6 Quality of online integration for increasing delay 60
3.7 Quality vs. Performance tradeoff 61
3.8 Refactorization benefit 62

4.1 Social link propagation leads to better decisions 68
4.2 Results about the scalability of FasTag 79

5.1 Relative relevance vs tag position in a recommendation list 88
5.2 Quality increases vs recommendation list sizes 91
5.3 Recall vs Precision 94
5.4 Distribution of the proposed optimal list sizes 94

List of Tables

1.1 An example of polysemy and synonymy in natural language 9

3.1 Caracteristics of the datasets 55
3.2 Initial quality of the three models in terms of RMSE score 56
3.3 Percentage of quality improvement 56
3.4 Quality vs. Performance tradeoff 60

4.1 Definitions from folksonomy 67
4.2 Characteristics of the datasets 75
4.3 Comparison of FasTag with some baselines 76
4.4 Comparison of the qualities gained by the four strategies 76
4.5 Result of the task 2 of ECML PKDD Discovery Challenge 2009 77

5.1 Characteristics of the datasets 90
5.2 Comparison of the three methods 92
5.3 Comparison of the three methods in twos 92
5.4 Average optimal list length with 10 tags at maximum 93

General Introduction

Overview

Recommender systems (RS) become increasingly popular both in online applications and in the research community, where many algorithms have been proposed. The aim of RS is to predict user preferences on a large selection of items. Recommender systems try to find items that are likely to be of interest for a given user. Because the user is often overwhelmed to face the considerable amount of items provided by electronic retailers and service providers, the predictions are a salient function of all types of e-commerce [121, 14]. For this reason, recommender systems attract a lot of attention due to their great commercial value [30, 66, 82, 35]. Books suggestion on Amazon, or movies on Netflix, are perfect examples of use in online stores.

Although it is obvious that the major goal of a recommender system is to generate meaningful recommendations , there are several challenges that RS have to face in addition to suggesting interesting items. They are, for example, concerned about the presentation of the recommendations in order to maximize their acceptance and the users' willingness to buy some items and reuse the recommender system later. However they often must manage large and growing amount of information. Therefore, the need of scalable algorithms is as important as the quality of recommendation. Indeed users are generally impatient and can not wait so long for some recommendations. As example, the most-known VoD provider, Netflix, offered a grand prize of US $1 million for an algorithm that is 10% more accurate than theirs [14], but they never used it despite they spent so much money, because they found the proposal not scalable [8].

The purpose of this thesis is to investigate and design scalable recommender systems. We aim to improve existing techniques and propose new ones that help on improving the scalability of recommender systems.

Thesis Contributions

We propose and implement two ways to face the scalability issue in different recommendation contexts. To enumerate, we study incremental methods as well as distributed

and heuristics-based techniques. In addition, we introduced a new algorithm which improves the quality of recommendation lists by optimizing their sizes.
Our contributions can be divided into three parts as listed below:

- *Explicit feedback* context refers to cases where users can explicitly give their opinions on items. Usually ratings are used (e.g., from 1 to 5 stars as in Amazon or Netflix). In such a context, recommending amounts to predicting the ratings a user would give to items, then classifying the items according to the predicted ratings.
  Matrix factorization (MF) is considered as the best of them in terms of accuracy [76, 70, 86]. However, one drawback of MF is that it lacks of scalability. The models generated by matrix factorization are static. Once model is generated, it delivers recommendations based on a snapshot of the incoming ratings frozen at the beginning of the generation. Therefore, it has to be computed periodically in order to take into account the new ratings. Although distributed MF techniques were proposed [110, 5, 161], it is not realistic to carry out the model frequently, because of the high cost of its computation.
  In this thesis, we propose a way to dynamically integrate new ratings without recomputing all the prediction model. We implement it, and our experimentations point out its efficiency to maintain the accuracy of the predictions at a good level [54, 51, 47].
- We also design a popularity and social-based tag recommender, we call FasTag. Unlike common nearest neighbors approaches, it has not a fixed number of neighbors to consider when making recommendations. It uses as it deems necessary based on a heuristic method. Moreover, Fastag is not limited to a user's vicinity, it exploits the transitivity of the users' similarity to enlarge the neighborhood circle. Thus, it enhances the quality of the recommendations, while limiting the social network traversal as short as possible.
- Finally, we study a novel way to improve recommendation accuracy. Instead of seeking for the best size-fixed list of items, we investigate how to optimize the list size. We also present a method for extracting the best candidate sublist, i.e., the most accurate one to recommend to the user.

Thesis Organization

We organized this dissertation in two major parts. The first one is a survey on recommender systems. It presents in two chapters their classification, the main challenges, and the methods and measures used to evaluate and compare some recommender systems. This part may be not essential for those who are familiar to the field of recommender systems.
The second part is dedicated to our contributions. It contains three chapters. Each of them tackles either a different problem. The contributions are somehow linked, so we recommend the reader to read them in the given order.

Finally, in the conclusion, we discuss some perspectives.

Part I

A survey of recommender systems

1 Introduction to Recommender Systems

Recommender systems (RS) are increasingly popular in the research community, where many algorithms have been suggested for providing recommendations. Their principal purpose is to predict user preferences on a large selection of items. Recommender systems try to find items [1] that are likely to be of interest for a given user. Because the user is often overwhelmed to face the considerable amount of items provided by electronic retailers, the predictions are a salient function of all types of e-commerce [121, 14]. For this reason, recommender systems attract increasingly a lot of attention due to their great commercial value [30, 66, 82, 35]. Making suggestions of books on Amazon, or movies on Netflix, are real world examples of the operation of industry-strength recommender systems.

In this chapter, we give an overview of RS. We present a classification of RS technology and some popular algorithms. We start by formally defining the main task of any recommender system in Section 1.1, then we discuss about their utility in a global view (Section 1.2). We end by introducing the most-accepted classification by the researchers in Section 1.3 [127, 3, 1, 22, 58], and present a state-of-the-art of the algorithms used by each class of recommender systems.

1.1 Formal Definition of Recommendation Task

A recommender system is an application which usually takes in input three classes of entities. We refer to them as users, items and the contexts in which we make recommendation. A context is additional information to take account that describes the specific situation in which the items will be recommended to a user (e.g., the location of the user when recommending close restaurants; the time for suggesting an itinerary that minimizes certain traffic tailbacks to a driver). In order to give a formal definition of recommendation task, we first introduce some notations. Let us denote by:

– U: the set of all users,

1. In this dissertation, we choose to use the term "item" to design the objects to be recommended to the users. However there are equivalent terms in the literature like "product", "resource" or "object" itself.

– I: the set of all possible items that can be recommended, such as songs, movies, books or restaurants,
– C: the set of contexts for making recommendation,
– f: a utility function which measures the interest of a given item for a user in a specific context, i.e., $f : U \times I \times C \rightarrow R$, where R is is a totally ordered set. For instance this interest may reflect the probability a user will buy a specific book or watch a movie.

The recommendation task can be summarized to find the top-K highest interesting items for a user $u \in U$ in the context $c \in C$ and recommend them. Indeed whatever the approach we used (rating prediction or purchase-likelihood estimation, for example), we always end to suggest a list of items to users, what matters the list size. Thus we can formally define the task of RS as follows

$$Top(u, c) = \underset{i \in I}{\overset{K}{argmax}}\ f(u, i, c) \tag{1.1}$$

Most of the existing approaches of recommender systems focus on recommending the most relevant items to users and do not consider any contextual information, such as time, place and the company of other people (e.g., for watching movies or dining out). In other words, they deal with applications having only two types of entities, users and items, and do not put them into a context when providing recommendations [116].

In this chapter, we mainly focus on the recommendation generalities. Thus we do not consider context-aware recommender systems. For more details about the latter, we refer the reader to [116, 11]. The utility function may then be reduced to $f : U \times I \rightarrow R$ and the highest interesting items to

$$Top(u) = \underset{i \in I}{\overset{K}{argmax}}\ f(u, i) \tag{1.2}$$

Let us notice that each user may be defined with a profile that includes user properties like age, gender, occupation etc. and also for the items using their characteristics. For example, in a movie recommender system, a movie can be defined by its id, genre, release date, director, actors etc.

1.2 Recommender Systems Function

In one hand, recommender systems are expected to suggest useful items to users by supporting them in various decision-making processes, such as what items to buy, what news to read, or which restaurants to try. On the other hand, electronic retailers and online service providers which use RS hope to increase their profits. For example, a travel intermediary wants to sell more hotel rooms by the increase the number of tourists to some destinations, while tourists wish to find a suitable hotel and interesting attractions when visiting a destination. Recommender systems aim to come up to these expectations. Thus they have to play a variety of functions. We list below some of them.

- *Increase the conversion rate* is probably the most important function of any RS, particularly for commercial RS where it amounts to raise the number of items sold. The conversion rate is the percentage of users who accept the recommendations and consume some items. This is the main function for which someone would want to exploit a recommender system.
- *Increase satisfaction and fidelity of users* through their experiences with the system. By relevant recommendations, and perhaps a properly designed human-computer interaction, the users could better evaluate the system and enjoy using it. This in turn will increase the system usage and the likelihood that the recommendations will be accepted. The first consequence of the users' enjoyment to use a system is their fidelity. The latter comes into more effect for a Web site with the recognition of users when they are visited the site and treated as a valuable visitor. This is fully carried out by RS with gradually refined suggestions as they can leverage the information acquired from the users in previous interactions.
- *Sell more diverse items* is another major function of a RS. Indeed while giving to users the possibility to get some hard-to-find items, it allows to bring to front all the items, not just the most popular ones, and then to reduce the long tail [129, 35]. This could be difficult without a RS.

1.3 Classification of Recommender Systems

Recommendation systems are usually classified on the basis of their approaches to estimate the interests of users [116, 127]. The authors usually agree on two broad classes.

- *Content-based filtering* recommend similar items to the ones a user was interested in the past. Their basic process consist in matching up the attributes of an item and the description of a user profile in which preferences and interests are stored.
- As for *collaborative filtering (CF) systems*, they recommend items to users based on the interests the other users expressed for those items. In practice, these interests may be expressed by the ratings users give to items (e.g., 1-5 stars) or the purchases they made. CF focuses on the relationship between users and items. It is the most widely used prediction technique in recommendation.

The major difference between collaborative filtering and content-based approaches is that CF mainly uses the interest of users on items to make predictions and recommendations, while content-based recommender systems rely on the features of users and items for predictions [127].

A third type of recommendation are *hybrid systems* which combine both above approaches in some manner. They allow to overcome their limitations, as we will present shortly.

1.3.1 Content-based Recommender Systems

Content-based recommendation systems (CbRS) share in common a means of representing the items that may be recommended, a way for creating a user's profile that contains information about its tastes, preferences and needs. Users can construct such a profile explicitly, but it can also be learned from the user's interaction with the system [105, 83]. CbRS rely on matching methods in order to determine which items may interest the most the user and recommend them.

Let us notice here that, in some applications, it can be appropriate to recommend an item that the user has already seen (i.e., purchased or rated) while in some others it is not interesting. Thus some items may be discarded from a recommendation list depending on the application. For example, a system should continue to recommend items that wear out (e.g., a razor blade) or are expended (e.g., print cartridge), while there is little value in recommending a movie that a user watched.

Furthermore, because the representation of items differs from a context to another, matching functions change according to that. In this section, we present alternative representations of items and some matching functions depending on each of these representations. We then discuss the strengths and weaknesses of content-based recommendation systems.

1.3.1.1 Item Representation

Items can be represented by a set of features and its description stored in structured data. For example, in a movie recommendation application, each movie can be described by the actors and directors involved in it, its genre and release year. In this case, many machine learning algorithms can learn a user profile from the ones of the items which interested him before, or a menu interface can be created to allow a user to easily create his profile. Section 1.3.1.2 discusses some approaches to learn a user profile from structured data.

In most content-based filtering systems, all item features are not available in organized form. Some information is extracted from free text (e.g., web pages, news articles or product descriptions). Thus unlike structured data, they are not attributes with well-defined values. In this case, term-based profiles may be used to represent the users [119, 142]. Each user is then represented by a vector of terms (i.e., words) selected from the free text describing the items he saw. In other words, each term is viewed as an attribute to which we can assign a weight depending on its popularity for the user. Moreover the root forms of terms are typically used thanks to some stemming processes [107, 101, 100].

term frequency-inverse document frequency (tf-idf) is generally used to compute the weight of each term. It originates from text search and is defined as follows:

$$\textit{tf-idf}(t,d) = \underbrace{\frac{f_{t,d}}{\max\limits_{z} f_{z,d}}}_{tf} \times \underbrace{\log\frac{N}{n_t}}_{idf} \tag{1.3}$$

where N denotes the number of documents [2] in the collection, and n_k denotes the number

2. Note that we used the word "document" in the description because it is traditionally the case due to original motivation of *tf-idf* to retrieve documents. In practice, we refer to items.

of them in in which the term t occurs at least once. The tf part computes the relative popularity of term t compared with the most one inside the document d, $f_{t,d}$ is its frequency. As for the *idf* part, it estimates the specificity of the term for the document.

Cosine normalization is usually used to normalize the $\textit{tf-idf}(t,d)$ values computes in the [0,1] interval. Thus the weight $w(t,d)$ of a term t for a document d is given by

$$w(t,d) = \frac{\textit{tf-idf}(t,d)}{\sqrt{\sum_{z} \textit{tf-idf}(z,d)^2}} \tag{1.4}$$

Let us say that free text data may lead to some complications when learning a user profile, due to the natural language ambiguity [116, 105]. The problem is that traditional term-based profiles are unable to capture the semantics of user interests since they are often driven by a string matching operation. If a term is found in both the users profile and the item, a match is made and the item is considered as relevant. Indeed the matching of string-based terms suffers from problems of polysemy[3] and synonymy[4]. For example, in Table 1.1 we give a part of news article. In this example that we take from [105], “Gray” is a given name rather than a color, and “power” and “electricity” refer to the same underlying concept.

Table 1.1: An example of polysemy and synonymy in natural language

With California's energy reserves remaining all but depleted, lawmakers prepared to work through the weekend fine-tuning a plan Gov. **Gray** Davis says will put the state in the **power** business for "a long time to come." The proposal involves partially taking over California's two largest utilities and signing long-term contracts of up to 10 years to buy **electricity** from wholesalers.

1.3.1.2 User Profile

In simple words, a user profile gathers his interests. It may consist of different types of information as we introduced above. This data can include the items that explicitly interest the user (i.e., the items the user purchased, liked or rated) and the ones that implicitly draw his attention. They typically correspond to the items the user searched or viewed (e.g., the mouse moves, for a certain time, over a picture of a book in a web page of electronic retailer). Of course, implicit data have to be taken with some uncertainty. Therefore, one has to mind transparent methods which can collect a large amount of implicit data in order to reduce the uncertainty. In contrast, there is little noise in the collected data when the user explicitly rates items. But, in practice, only a small percentage of items receives explicit feedback from the users [105].

There are mainly two means to create a user profile.

3. the presence of multiple meanings for one word
4. multiple words with the same meaning

– A first means, rather simple, is to provide an interface to a user that allows him to set his own interests. And from these data, the recommender system creates a representation of the user, i.e., his profile. Often check and combo boxes with known values of attributes are presented to users, e.g., genres of preferred movies, the names of favorite sports teams, or favorite news types. The users can also type some keywords in case of free text descriptions of items, e.g., the name of author or hotel chain that interests the user. Once the user has entered this information, a simple database matching process is used to find items that meet the specified criteria and display them to the user. However this approach although simple has several limitations. First, it requires effort from the user and it is difficult to get many users making this effort. This is particularly true as the user's interests are not necessary fixed. Second, the used user interfaces do not always provide a way to determine the order in which to classify and present items.
– An alternative to asking a user to construct his personal profile is to learn it from the history of the user's interactions with the recommendation system, i.e., his past transactions, actions or searches. Indeed this history may serve as training data for a machine learning algorithm which creates the user profile. The creation of the user profile can be seen as a form of classification learning where it consists in learning and dividing the items into a list C of classes. For instance, as two classes, one can consider the binary categories "user-likes" and "user-dislikes". We can consider that a user who purchased an item liked it but if the user purchased and returned an item, we take it as a sign that the user does not like the item. The process of classification becomes here to build a model for the user profile which may determine which items he should like or not.

Below we review some of the most used learning algorithms in the context of content-based recommender systems.

1.3.1.2.1 Probabilistic Methods and Naïve Bayes

Naïve Bayes is a probabilistic approach to inductive learning, and belongs to the general class of Bayesian classifiers. These approaches generate a probabilistic model based on previously observed data. In the literature there are two commonly used variant of naïve Bayes, the multivariate Bernoulli and the multinomial model [80, 94]. Empirically, the former outperforms the latter [94].

The model that they learnt estimates the **a posteriori** probability, $\mathcal{P}(c|i)$, of item $i \in I$ belonging to class $c \in C$. This estimation is based on the **a priori** probability, $\mathcal{P}(c)$, the probability of observing an item in class c, $\mathcal{P}(i|c)$, the probability of observing the item i given c, and $\mathcal{P}(i)$, the probability of observing the item i. Using these probabilities, the Bayes theorem is applied to calculate $\mathcal{P}(c|i)$:

$$\mathcal{P}(c|i) = \frac{\mathcal{P}(c)\mathcal{P}(i|c)}{\mathcal{P}(i)} \tag{1.5}$$

As $P(i)$ is equal for all $c \in C$, it may be removed from Equation 1.5. To classify the item i, the class c_i with the highest probability is chosen:

$$c_i = \underset{c \in C}{argmax}\ \mathcal{P}(c|i) \tag{1.6}$$

Here, the problem is that we do not know the values for $\mathcal{P}(i|c)$ and $\mathcal{P}(c)$, but we can estimate them from the observed data. However the estimation is problematic as it is very unlikely to see a user interested by an item more than once [5]. Thus the observed data is generally not enough for generating good probabilities.

The naïve Bayes classifier overcomes this problem by simplifying the model through the independence assumption: all the attributes (e.g., words or tokens of document in text classification) of the observed item i are conditionally independent of each other given a class. Because of this, one can estimate individual probabilities for the attributes of an item one by one rather than the complete item as a whole. Even if the naïve Bayes assumption of the independence of class-conditional attribute is clearly violated in the context of text classification, naïve Bayes performs very well [116, 105].

The assumption of independence allows to express $\mathcal{P}(i|c)$ as follows:

$$\mathcal{P}(i|c) = \mathcal{P}(c) \prod_{a \in A_i} \mathcal{P}(a|c)^{N(i,a)} \tag{1.7}$$

where A_i is the list of attributes of the item i and $N(i, a)$ is defined as the weight of the attribute $a \in A_i$ for the item. For example, in text classification it is taken as the number of times a word or token appeared in a document. Generally, a smoothing method is used to assess the probabilities $\mathcal{P}(a|c)$ of the attributes computed by simple event counting beforehand. For further details about naïve Bayes methods for recommendations, we refer the reader to [116, 105].

Naïve Bayes has the advantage to be efficient and easy to implement but it is not as good as some other statistical learning methods like nearest-neighbor classifiers that we present in the following section.

1.3.1.2.2 Nearest Neighbor Classifiers

For the classification of an unseen item, nearest neighbor classifiers, also called lazy learners, rely on a similarity function for the comparison of items in order to retrieve a number nearest neighbors to consider for a given item. They begin to determine the k-nearest neighbors of the item thanks to the similarity function, then they derive its class from the ones of these neighbors. A simple way is to take the class which occurs the most among the neighbors. However it can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the final decision than the more distant ones.

The similarity function used to obtain the the nearest neighbors depends on the type of data [1]. For example, a Euclidean distance metric is often used in the case

5. Let us notice again that this depends on the application. For example, a system should continue to recommend items that wear out (e.g., a razor blade) or are expended (e.g., print cartridge), while there is little value in recommending a movie that a user watched.

of structured data, and the cosine similarity measure is the mostly used when using a term-based vector space model. The latter is defined as follows:

$$sim(i,j) = \frac{\sum_{t \in T} w(t,i) \cdot w(t,j)}{\sqrt{\sum_{t \in T} w(t,i)^2} \cdot \sqrt{\sum_{t \in T} w(t,j)^2}} \tag{1.8}$$

where T represents the list of possible terms.

1.3.1.2.3 Decision Trees and Rule Induction

Decision trees, as their name indicates, are trees in which internal node represents a "test" on an attribute of profiles (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (i.e., decision taken after computing all attributes).

The process of trees' learning is done by recursively partitioning training data into subgroups, until the subgroups contain only instances of a single class. The splits are generally decided by maximizing the information gain on each attribute $a/$ The information gain, defined in terms of entropy H, is given by the next formula:

$$IG(a) = H(Parent) - \sum_{v \in vals(a)} H(p_v) \frac{N(p_v)}{N} \tag{1.9}$$

where $vals(a)$ is the list of existing values of the attribute a, N the size of the training set and $N(p_v)$ the one of the partition p_v which contains objects having v as value for attribute a. Let us note that in addition to Entropy, Gini Index and misclassification error are the most common used[12, 9, 42]. Decision trees may be used with model-based approaches for recommendation [6] by using content features to build a decision tree that models all the variables involved in the user preferences [19].

Before discussing the advantages and disadvantages of content-based RS, we emphasize that there are, in the literature, a lot of algorithms that may be used like Linear Classifiers, Relevance Feedback and Rocchio's Algorithm. We refer the reader to [83, 105] for broader surveys.

1.3.1.3 Advantages and limitations

Among the advantage of using a content-based RS, one can cite the user independence. Indeed the system exploits only the active user's preferences to build his own profile. What is not the case of collaborative filtering (see Section 1.3.2). Moreover, explanations on how the recommender system works can be provided by pointing out why a given item is recommended. for example, one can list content features or descriptions

6. we see this type of Collaborative filtering RS in Section 1.3.2.2

that caused its recommendation. Clear explanations help users to trust the system and thus can increase the recommendations' conversion rate [83].

Content-based recommenders do not suffer from the cold-start problem. This is the difficulty of recommending items not yet viewed by any user or to suggest items to a user who has not yet viewed any item. The use of content allows them to face this problem.

However, content-based filtering is limited by the features that are explicitly associated with the items that these systems recommend. Although some information retrieval techniques work well in extracting features from text document, it is hard to apply them to multimedia data (e.g., graphical images, audio and video). In this case, one can have recourse to assigning manually features to items, but that is often not practical due to limitations of resources.
In addition, with content analysis, if two different items are represented by the same set of features, they are indistinguishable. For example, since text-based items are usually represented by their most important keywords, content-based filtering can not differentiate between a well-written article and a badly written one, if they happen to use the same terms [3].

Another drawback of content-based filtering is that the system can only recommend items that suit the user's profile [7]. Therefore, the user is limited to receive items similar to those he already prefers.

Finally, content-based RS are known to be less accurate than collaborative filtering systems. We talk about the latter in the next section.

1.3.2 Collaborative Filtering Systems

collaborative filtering (CF) is the most successful approach for building recommender systems. It uses the known preferences of the other users to predict the unknown preferences of a given user, then it makes its recommendations. Collaborative filtering takes account of the fact that, in real life, people rely on recommendations from other people, and above all from those who share "common things" with them like friends or family. It assists and augments this natural and social process to help people sift through a collection of items (e.g., books, articles, news articles, music, restaurants, and so forth) and find the most interesting and valuable items for them.

Collaborative filtering algorithms can be further categorized into model-based CF and memory-based CF. The former group try to learn and build predictive models which reflect the behavior and interests of the users. From these models, it is possible to estimate what items might be relevant to recommend to a user. Memory-based CF algorithms try to represent directly the shared "common things" by users. They typically rely on correlation or similarity measure to put in obvious the relations between users. Their fundamental assumption is that if users X and Y have similar behaviors (e.g., buying, watching, listening or rating), they will act on other items similarly. Thus, all

7. This is the over-specialization problem

we have to do is to find the most similar users of a given one when we want to make recommendation for him.

1.3.2.1 Memory-based CF

Memory-based CF algorithms refer generally to neighborhood-based approaches [127]. Each user belongs to a group of people with similar interests. Once the so-called neighbors of a user are identified, a prediction of his preferences on new items can be done.

The key idea of neighbor based approaches is that the interest of a user u in an item i is likely close to the one of another user v, if u and v have similar interest in other items. From another point of view, u is likely to have close interests in two items i and j, if other users have given similar interest in these two items.

Neighbor based approaches automate the common principle of word-of-mouth, where one relies on the opinion of trusted sources or other like-minded people to evaluate the value of an item according to his own preferences. Therefore such methods are characterized to be simple, justifiable, efficient and stable [116, 127].

Due to relying on the opinion of like-minded people neighbor based approaches need to compute the nearest neighbors and then assign a similarity value to each item pair or to each user pair of their training data. The first variant is referred as the *item neighbor* and the second as the *user neighbor method.* Assuming the user neighbor method, the interest $f(u,i)$ of the user u in item i can be estimated as

$$f(u,i) = \frac{1}{\sum\limits_{v \in \mathcal{N}_i^k(u)} s_{uv}} \left(\sum_{v \in \mathcal{N}_i^k(u)} s_{uv} \rho_{uv}(r_{vi}) \right) \tag{1.10}$$

where $\mathcal{N}_i^k(u)$ represents the k nearest neighbors of u who already saw, i.e., purchased or rated, the item i and s_{uv} the similarity or proximity between the users u and v. r_{vi} stands for the interest that v stated in i and ρ_{uv} a uni-variate predictor function (e.g., the identity function).

In case the user interests are just given by transactions where we only know what the users purchased, the Jaccard or Dice index, or the Hausdorff metric may be used to compute the similarities [1]. When the user interests are formulated with ratings, correlation-based similarities are generally taken. A popular correlation-based similarity measure is the Cosine Vector similarity. It is defined as follows:

$$s_{uv} = \frac{\sum\limits_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum\limits_{i \in I_u} r_{ui}^2 \sum\limits_{j \in I_v} r_{vj}^2}} \tag{1.11}$$

where r_{ui} is the rating that u assigned to i and I_u the set of items that u have rated. I_{uv} represents the set of common items rated by both u and v.

As someone may remark, a problem with this measure is that it does not consider the differences in the mean and variance of the ratings made by users u and v. The

Pearson Correlation similarity, another popular measure, allows to take account of that by discarding the effects of mean and variance when comparing ratings with the next equation:

$$s_{uv} = \frac{\sum\limits_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum\limits_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2 \sum\limits_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}} \tag{1.12}$$

In this equation, $\bar{r}_u$ (resp. $\bar{r}_v$) is the mean of ratings of u (resp. v). When the Pearson Correlation similarity is used, $\rho_{uv}(r_{vi})$ is set to $r_{vi} - \bar{r}_v + \bar{r}_u$ and Equation 1.10 becomes

$$f(u,i) = \bar{r}_u + \frac{1}{\sum\limits_{v \in \mathcal{N}_i^k(u)} |s_{uv}|} \left(\sum_{v \in \mathcal{N}_i^k(u)} s_{uv}(r_{vi} - \bar{r}_v) \right) \tag{1.13}$$

There are other similarity measures like the Mean Squared Difference which evaluates the similarity between two users u and v as the inverse of the average squared difference between the ratings given by u and v on the same items. A list of similarity measures for neighbor based approaches are discussed in [1].

Let us notice that user correlations are unreliable, since there are typically very few common item between two arbitrary users. Therefore item neighbor methods are preferred in practice [82]. Moreover the latter scales more than user neighbor methods although reductional techniques may be used to improve performance of systems by decreasing data. An exhaustive enough survey about reductional techniques like Condensed k-Nearest Neighbor, Model Based k-Nearest Neighbor or Clustered k-Nearest Neighbor is presented in [1].

Computing the similarity of users or items from the user-item database have limited scalability for large datasets. Furthermore the computed similarities must be updated when users submit new interests. This may explain the passion for “social recommendation” where users’ friends from an online social network can be used instead of computing some similarities.

Online social networks present new opportunities for neighbor based approaches. Indeed in real life, people often ask to their social networks for advice before purchasing a product or consuming a service. Research in the fields of sociology and psychology indicates that people tend to associate and bond with similar others, also known as homophily [96]. They are more willing to share their personal opinions with their friends, and typically trust recommendations from their friends more than those from strangers and vendors. Popular online social networks, such as Facebook [8], Twitter [9], and Youtube [10], provide novel ways to communicate and build virtual communities. Online social networks do not only make easier the sharing of opinions with other persons, but they also serve as platforms for developing new RS algorithms which automate the manual and anecdotal social recommendations in real life [144, 84].

8. https://www.facebook.com
9. https://twitter.com/
10. https://www.youtube.com

1.3.2.2 Model-based CF

Model-based CF algorithms learn a model from previous user activities and use this model to classify items according to their interestingness. They are able to recognize complex patterns based on data and can make intelligent predictions.

The algorithms used to learn models are the same than those used in machine learning and data mining. For instance Bayesian models, clustering models, and diffusion methods, have been investigated [116, 127, 18]. Among all the propositions, dimensionality reduction techniques are the most popular. They model the user-item interactions with factors representing latent features of users and items. Matrix Factorization, one of these technique, is today considered as giving the most powerful predictive model. However more recently neural networks have been introduced for the task of recommendation, and researchers are particularly hopeful in boltzmann machines [108, 118].

We present in the following a short review of some popular model-based CF algorithms.

1.3.2.2.1 Dimensionality reduction techniques

It is common in RS to have not only a data set with features that define a high-dimensional space, but also very sparse information in that space. Therefore, dimensionality reduction comes in naturally. Dimensionality reduction techniques allow to downsize the amount of relevant data while preserving the major information content. They are very used a domains like data mining, machine learning and cluster analysis.

In the following, we summarize two relevant dimensionality reduction techniques in the context of RS which are Matrix Factorization (MF) and probabilistic Latent Semantic Analysis (pLSA).

Matrix Factorization (MF) Matrix Factorization is one of the most popular dimensionality reduction techniques for RS. It gives good scalability while allowing remarkable prediction accuracy. In the literature, matrix factorization is well investigated and many types of factorization have been proposed [76, 70, 86]. In its basic form, matrix factorization characterizes both items and users by vectors of factors (also called latent features) inferred from user feedback patterns. The correspondence degree between user and item determines the position of the latter in the final list of recommendations.

As illustration, one can consider the context of movie recommendation and a two-dimensional space where the x-axis gives the romance's degree of a movie and the y-axis action's one. We can plot into this space users according to the extend they like romance movies or action movies. Figure 1.1 presents an example of this space where *Arthur* has a preference in action movies while *Bob* prefers romance ones. Thus one can recommend $movie_1$ to *Arthur* and $movie_2$ to *Bob*. To classify the items when recommending to a user, the inner product of the user vector of factors and the ones of items is generally used as a scoring function. Let us take p_u as the vector of factors of user u and q_i the

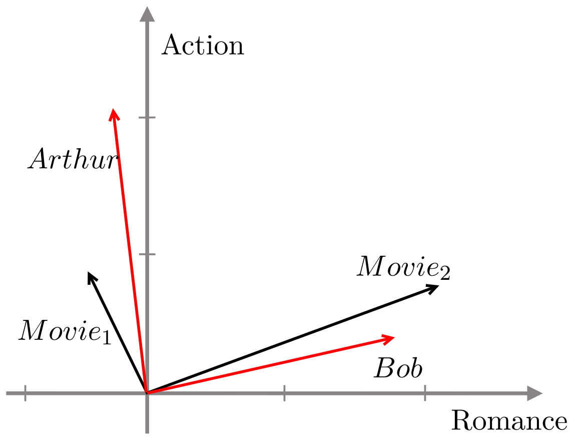


Figure 1.1: A romance-and-action space of movies and users

one of item i, the interest $f(u,i)$ of u in i is defined by

$$f(u,i) = p_u \cdot q_i^T = \sum_k^K p_{uk} q_{ki} \tag{1.14}$$

Hence the main task of factorization engines is obviously to compute reliable vectors of factors. We discuss more widely this point in Chapter 3. Another advantage of matrix Factorization is that it is particularly suitable for large data sets which are costly to store and manipulate. Indeed if K latent features are used for each user or item, with a dataset of M users and N items, only $K(M+N)$ elements have to be stored and not MN.

Probabilistic latent semantic analysis (pLSA) pLSA uses hidden variables to explain the co-occurrence pairs of data. But unlike MF, it is a statistical technique based on a probabilistic model. Well developed inference methods including likelihood maximization and Gibbs sampling can be employed in pLSA [59, 39]. pLSA models the relations between users and items through the implicit overlap of genres, as compared to the two-sided Bayesian clustering where each user and item belong to a single specific category. In pLSA, the co-occurrence probability $\mathcal{P}(u,i)$ of user u and item i is expressed using the conditional probability given a hidden variable k

$$\mathcal{P}(i|u) = \sum_k^K \mathcal{P}(i|k)\mathcal{P}(k|u) \tag{1.15}$$

Some variational approaches allow to obtain $\mathcal{P}(i|k)$ and $\mathcal{P}(k|u)$ by maximizing the per-link log-likelihood of the observed dataset, as proposed in [64] and shown in the following:

$$L(P,Q) = \frac{1}{E} \sum_{(u,i)} \log \mathcal{P}(i|u) = \frac{1}{E} \sum_{(u,i)} \log \left(\sum_k^K \mathcal{P}(i|k)\mathcal{P}(k|u) \right) \tag{1.16}$$

with respect to vectors P and Q which parametrize $\mathcal{P}(i|k)$ and $\mathcal{P}(k|u)$ by $\mathcal{P}(i|k) = q_{ki}$ and $\mathcal{P}(k|u) = p_{uk}$. Here E is the total number of user-item links. We remark that the sum over (u, i) includes only the observed user-item pairs. The expectation maximization (EM) algorithms can be used to find the value of Q that maximize $L(P, Q)$.

1.3.2.2.2 Artificial Neural Networks (ANNs)

In recent years, Artificial Neural Networks are being increasingly recognized in the area of classification and prediction, where regression models and other related statistical techniques have traditionally been employed [117, 116, 108, 138].

ANNs are non-linear mapping structures based on the functioning of the human brain. They can identify and learn correlated patterns between input data and corresponding target values with great capacity in predictive modeling.

ANNs imitate the learning process of the human brain. Therefore, they can process problems involving non-linear and complex data, even if the latter are imprecise and noisy. An artificial Neural Network consists of simple functional and highly interconnected units called neurons. Figure 1.2 gives a graphical presentation of a neuron.

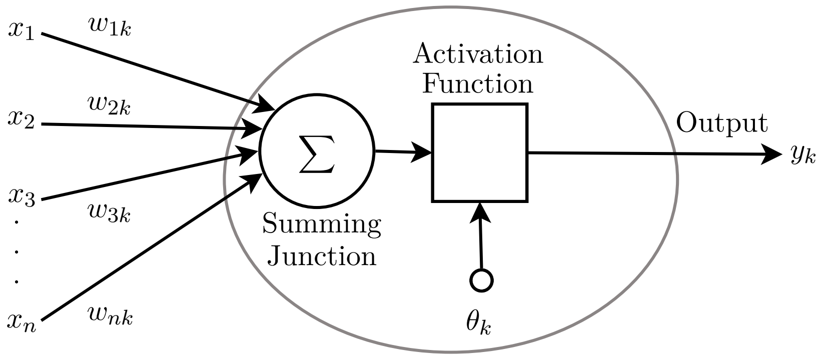


Figure 1.2: Graphical presentation of neuron

The activation of a neuron, i.e. its output, depends on the amount of signals it received, typically summation is used (e.g., $\sum_i x_i w_{ik}$) and some threshold θ_k over which the neuron becomes activated. The output of the activation function can be expressed by

$$y_k = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} x_i w_{ik} \geq \theta_k \\ 0, & \text{else} \end{cases}$$

The neural networks are built from layers of neurons connected so that on layer of neurons receives its input from the previous layer and gives its output on the subsequent layer. In other words, the activation of a neuron means that in its turn it sends a signal to its outbound links (i.e., synapses).

There are various types of ANNs like multilayer perceptron and Kohonen networks. The reader can have a large discussion about the variety of existing ANN types with [78, 81, 149, 79].

The most widely used learning algorithm in an ANN is the Backpropagation[11] algorithm. It is a common method for training artificial neural networks. It is used in conjunction with an optimization method such as gradient descent. The latter calculates the gradient of a loss function by considering all the weights in the network.

Hence let us notice that some matrix factorization models can be considered as the learning of a multi-layer perceptron with the identity activation function in each neuron [131]. Figure 1.3 is an example of such a multi-layer perceptron. It has the users as inputs, the items as outputs and K hidden neurons. The weights represent the factors introduced above for MF. To make recommendation for a user, only its corresponding

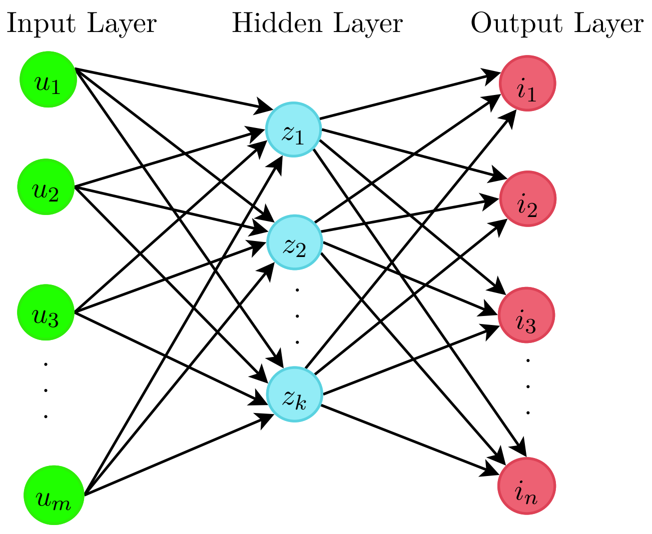


Figure 1.3: A multi-layer perceptron for collaborative filtering

input neuron has to be activated and all the rest of inputs taken off. Then the output neurons (i.e., the items) are classified and recommended according to their degree of activation.

1.3.2.2.3 Diffusion-based methods

PageRank is perhaps the most popular diffusion-based algorithm with Google's advent [21]. Almost all diffusion-based algorithms exploit item-item networks, they can always use specific transformations, like projections, on their input data if the latter are not so suited. In the network, the links between items represent their similarity degrees. For example, Zhang et al. consider in [151] that the items are either similar or dissimilar,

11. an abbreviation for "backward propagation of errors"

and used 1 and 0 to reflect that.
Generally a symmetric adjacency matrix A is used to model the network. Making some recommendations to a user comes to use his past preferences (i.e., the items that interest him before) as starting points in the network and propagate the weight of links towards yet unevaluated items by the user.

In [151], Zhang et al. recommend items to a user by a process motivated by heat diffusion: they represent the items liked and those disliked by this user respectively by hot and cold spots, and recommendation is made according to the stability of the "equilibrium temperature" of the network nodes. The discrete Laplace operator [12] of the network takes the form $L = 1_N - D^{-1}A$ where D is the diagonal degree matrix of the network, with elements $D_{\alpha\beta} = k_\alpha \delta_{\alpha\beta}$. The resulting temperature vector h_u for a user u, is the solution of the heat diffusion equation

$$Lh_u = f_u \tag{1.17}$$

The fixed elements of h_u correspond to items already evaluated by user u; they are set to 1 for the items liked by the user (which act as heat sources) or 0 for those disliked by him (they act as heat sinks). This mathematically corresponds to the Dirichlet boundary condition. The external flux vector f_u is non-zero only for items evaluated by user u and allows for fixed values attributed to sources and sinks. Equation 1.17 can be solved by using the Green's function method and the involved computational cost can be lowered by the use of various algebraical properties of L [151]. Let us note that it is straightforward to find the equilibrium h_u iteratively by setting the initial temperature vector h_u^0 to contain only the fixed heat sources and sinks and then iterate

$$h_u^{n+1} = L' h_u^n \tag{1.18}$$

where L' is the same as the above Laplace operator, but it keeps unchanged the elements in h_u corresponding to u's evaluated items.

Many types of diffusion-based algorithm are in the literature. From multilevel to probabilistic spreading methods, the reader is spoiled for choice [154, 152].

Next, we talk about the most-known advantages of CF and its main drawbacks.

1.3.2.3 Advantages and limitations

Collaborative filtering work generally better than content-based one. As it uses other users' preferences, it can deal with any kind of content and then recommend different items, even the ones that are dissimilar to those seen in the past. This is one of its greatest strength.

Nonetheless, collaborative filtering has its own drawbacks due to the fact that it relies solely on users' preferences to make recommendations. It is affected by the cold-start problem. Indeed, until a new item received a substantial number of users' feedback

12. It is a discrete analog of the heat diffusion operator $-\nabla^2$ which is well-used in physics.

(e.g., purchases or ratings), the system would not be able to recommend it, so it is for a new user. Furthermore, CF techniques do not work well for the so-called “gray sheep”, which refers to users to whom the opinions do not consistently agree or disagree with any group of people and thus do not benefit from other users’ expressed preferences [127].

To overcome all these limitations, hybrid systems which can mix collaborative and content-based filtering have been proposed.

1.3.3 Hybrid Systems

As we said above, a third type of recommendation uses at a time several recommender systems[22, 56, 43, 28, 109]. Commonly called hybrid RS, they help to avoid certain limitations of collaborative and content-based filtering as described above. For instance, content-based recommendation systems can provide recommendations for “cold-start” items (i.e., those for which little or no training data is available), but typically have lower accuracy than collaborative filtering systems. Conversely, CF techniques often provide accurate recommendations, but fail on cold start items. Hybrid systems are then more-suited for real world context in order to yield better recommendations across the board. A study of Good et al. asserts that a hybridized recommender system is better than any single algorithm [45].

In [23], Burke listed seven hybridization methods to combine collaborative and content-based filtering that we report below:

1. *Weighted:* The scores of several recommendation techniques are combined together to produce a single recommendation.
2. *Switching:* The system switches between recommendation techniques depending on the current situation. It chooses one of them and applies it.
3. *Mixed:* Recommendations from several different recommenders are presented at the same time.
4. *Cascade:* One recommender refines the recommendations given by another.
5. *Meta-level:* The model learned by one recommender is used as input to another.
6. *Feature combination:* Features from different recommendation data sources are thrown together into a single recommendation algorithm.
7. *Feature augmentation:* The output from one technique is used as an input feature to another.

Hybridization is not limited to combine different recommendations classes. Some hybrid systems combine different implementations of the same class of technique like switching between two different content-based recommenders.

There are few studies about comparing hybridization methods. We refer the reader to [23, 22, 106] for more details.

1.4 Conclusion

We presented in this chapter a survey on recommender systems. We discussed their most-known classification and techniques. In the next chapter, we will talk over their main challenges, methods and measures that researchers usually use to evaluate them.

2 | Challenges and Evaluation of Recommender Systems

It is obvious that the major goal of a recommender system is to generate meaningful recommendations to users for items that might interest them. However there are several challenges than RS have to face in addition to suggesting interesting items. They are, for example, concerned about the presentation of the recommendations in order to maximize the acceptance of recommendations and the users' willingness to buy some items and reuse the recommender system. They have often to manage growing amount of information, therefore scaling algorithms are also well-sought.

We expose below a not-exhaustive list of challenging research topics in recommender systems. Indeed it never cease to come new emerging topics in the literature. We refer the reader to [116] for a discussion on these coming topics. We tackle also the evaluation of recommender systems. We discuss the approaches and some popular measures which are used.

The sequel of this chapter is organized as follows. Section 2.1 lists some challenges of RS. In Section 2.2, we review evaluation approaches and measures used to compare different recommendation algorithms, and how they fulfill a task of interest. We conclude in Section 2.3.

2.1 Challenges of Recommender Systems

As we said above, the first concern of recommender systems is to give recommendations which satisfied the most the users. Users' satisfaction may depend on a lot of aspects like the accuracy of the recommendations, their presentation or the RS response time. Researchers in the field of recommender systems face several challenges on which result the use and performance of their algorithms. We expose here a list of challenging research topics for researchers.

2.1.1 Data Sparsity

Data sparsity is one of the major problems encountered by recommender system. Online retailers often face exceedingly large pool of items. For instance, the world's largest online retailer, Amazon.com, has more than 244 million customers[1] and sells over 200 million products in the USA[2].

From this very large range of items, overlap between two users is often very small or non-existing. Users do not rate most of the items and the available user feedback (e.g., ratings) are usually sparse. This is the main reason that data sparsity has great influence on the quality of recommendation. New items cannot be recommended until some users rate them, and new users are unlikely given good recommendations because of the lack of ratings or purchase history. Furthermore, it may be difficult to identify users with similar tastes as such if they have not both rated/brought any of the same items. This could reduce the effectiveness of a recommendation system which relies on comparing users in pairs and therefore generating predictions[127, 3]. Dimensionality reduction techniques are more suited to alleviate the data sparsity problem. Our proposal in Chapter 3 relies on such techniques.

2.1.2 Scalability

Scalability indicates the ability of a system to handle growing amount of information in a graceful manner. In the last year, Amazon added 30 million customers[1]. This is not an exception for e-commerce sites. The number of users and items of major sites is tremendously growing. To attract and keep purchasing their customers, these sites need to react immediately to online client requirements and make recommendations for all users according to their purchases and ratings history, which demands a high scalable recommendation solution. For comparison, Amazon sold 426 items per second in run-up to last Christmas[3], therefore it had to make so many recommendations as they usually do after each sale.

In practice, a solution that works fine when tested offline on relatively small data sets may become inefficient or even totally inapplicable on very large datasets [116]. It is therefore essential to consider the computational cost issues and search for recommender algorithms that are little demanding, easy to parallelize or both. A third option is to rely on incremental techniques which allow to not globally recompute the recommendations but slightly adjust them in accordance with the newly arrived data [85, 145, 143, 103, 120].
Dimensionality reduction techniques such as Matrix Factorization deal well with the scalability problem at the moment of recommendation, but they have to undergo expensive factorization steps. Incremental approaches were proposed in [143, 112, 120]. There are also some solution for memory-based CF as the ones presented in [85, 145, 103].
Let us remind here that the precise subject of our thesis is to face the scalability problem of recommender systems. We tackle it in Chapters 3 and 4.

1. `http://bit.ly/VKAdh5`
2. `http://bit.ly/VO2kMi`
3. `http://bit.ly/1dRIXF3`

2.1.3 Diversity vs. accuracy

In order to satisfy and positively surprise the users, a recommender system needs to recommend items the users will like and most probably would not have found on their own. This requires the recommender system to suggest a broader range of items including niche items as well. Indeed recommending only popular and highly rated items has very little value for the users, since popular items are easy to find.

Thus, a key feature of a recommender system is its ability to (i) satisfy and (ii) positively surprise the users with less obvious items that are unlikely to be reached by the users. The majority of the algorithms proposed in the literature focus on the first point by improving recommendation accuracy. To achieve the second point, the use of diversity among the recommendation is the main way. Indeed it is more likely that the users will find a suitable item if there is a certain degree of diversity among the suggested items. The problem here is how to combine the diversity goal with the accuracy of the recommendation. From the study of the items' usage contexts [99] to the introduction of some item ranking techniques [2] or hybrid algorithms [153], several propositions exist in the literature. In Chapter 5, we propose a novel parameter-free algorithm that can deal with this dilemma.

2.1.4 User interface

Recommendation goes beyond telling to users what items they might like. The user interface (UI) of a recommender system can have a critical and decisive effect on factors such as the overall system usability, system acceptance, item rating behavior, selection behavior, trust, willingness to buy, willingness to reuse the recommender system, and willingness to promote the system to others [34]. Two main points of UI are the explanations of the recommendations and their presentation. Indeed users appreciate when it is clear why a particular item is recommended for them. Furthermore, since the list of potentially interesting items may be very long, it needs to be presented in a simple way. Indeed it should be easy to navigate through it, and browse different recommendations which are often obtained by distinct approaches [84]. The presentation of the recommendations and the interactions with the users are limited in the case of particular devices like smartphones [115].

Explanations summarize the reasons why a specific item is proposed. They can have many advantages, from inspiring user trust to helping users make good decisions [147, 135, 93]. [147] made an online experiment on a real-world platform indicating that explanations are an essential component of recommendation systems, that significantly increases users' perception of the utility of a recommender system, the intention to use it repeatedly as well as the commitment to recommend it to others. [126] conducted a user study on five music Recommender Systems. It shows that users like and feel more confident about recommendations that they perceive as transparent. Explanations can be made though a variety of means like a list of tags, informative text (e.g., "80% of the people who bought this also bought that") or social explanations (e.g., "*Alice*, *Bob* and 2 friends like this") [140, 129, 124].

2.1.5 Vulnerability to attacks

In cases where anyone can provide recommendations, it is desirable for recommender systems to introduce precautions that discourage malicious actions. Collaborative filtering (CF) algorithms are capable of generating personalized recommendations. However, they are vulnerable to shilling attacks, where a group of spam users collaborate to manipulate the recommendations by inserting malicious user profiles into the system to push or nuke the reputations of targeted items. Robustness is the aptitude of recommender systems to face these attacks. Several attack detection algorithms have been developed, in recent years, to detect spam users and remove them from the system [26, 162]. They may rely on a panel of solutions like belief propagation [162], supervised learning method [26] or abnormal profiles detection [155]. In [148], the author argues that trust-based recommender systems are facing novel recommendation attack which is different from the profile injection attacks in traditional recommender system. It proposes a data provenance method to trace malicious users.

Another vulnerability is due to power users. The latter are those who can exert considerable influence over the recommendation outcomes presented to other users in Collaborative Filtering (CF). RS operators encourage the existence of "power user" [4] communities and leverage them to help "fellow users"[4] make informed purchase decisions. Thus, RS research in this area has focused on power user selection and utilization to address challenges such as data sparsity for new items or users. But, it remains a potential for corruption by power users who can provide biased opinions. Because of the influence that power users wield, biased opinions they provide can have significant impacts on RS accuracy and robustness. [122, 141] investigate the impact of biased opinions on RS accuracy. The results show that the in-degree centrality is a good criteria to identify the power users. Their influence was measured by comparing the accuracy and robustness of RS before and after attacks of power users [141].

2.1.6 Some other Challenges

Besides the above well-investigated challenges, there are some additional ones that are discussed in the literature [62, 127, 116, 84]. Among the latter, one can list:

– **Time Value.** Most recommendation algorithms neglect the time stamps of the evaluations. In some item spaces, such as books or movies, the relationships between users and items changes slowly over time. But in other item spaces, such as daily news, items' relevance change rapidly. Also some users might value above all the most recent information, while other users might prefer the deeper insights of careful reporting that takes days or weeks to complete.

– **User Action Interpretation.** Explicit ratings are a valuable signal of user interest. However, in most systems much more information is available in implicit signals of user interest, such as what items he clicked on, how long he read them, which items were added to a wish list or shopping cart, etc. One deep challenge in these data is the interpretation of negative choices in addition to positive ones.

4. It is the term used in the literature [122, 141]

For example, what we can conclude if a user does not choose to click on or rate a news item [92].

– **Privacy.** In the attempt of making increasingly better recommendations, recommender systems collect as much user data as possible. This will clearly have a negative impact on the privacy of the users. Therefore the users might start feeling that the system knows too much about their true preferences. The problem here is how to develop recommender systems that use protected data.

Another problem, and not the last, is the evaluation of the ability of a RS to face a challenging problem. Our next section tackle this point.

2.2 Evaluating Recommender Systems

Evaluation is a key factor to reflect the quality of a recommendation algorithm. Indeed many algorithms have been suggested for the recommendation task. Therefore, as they typically perform differently in various domains and tasks, it is very important to be able to decide what algorithm matches the best a domain and a task of interest.

We review in this section the approaches for evaluating recommender systems and different quality measures used by researchers in this field.

2.2.1 Evaluation approaches

One can evaluate RS by using three approaches: offline analysis, user studies or reality-closed experiments. Furthermore, a combination of these approaches is also possible [123, 29].

In the following subsections, we begin with offline approaches. They are typically easy to conduct, as they require no interaction with real users. We then describe user studies, where a small group of persons uses the system in a controlled environment. The studies allow to collect both quantitative and qualitative information about the systems, but we may have to consider various biases in the experimental design. The recourse of a pool of real users, preferably unaware of the experiment, is perhaps the the most trustworthy approach. However, it allows to collect only certain types of data.

2.2.1.1 Offline evaluation

Offline approaches rely on pre-collected dataset that contain a history of interactions (e.g., ratings, purchases or votes) between a set of users and items. Demographic information about the users and some properties of the items are often available also. The dataset is split in a test and a training set. Thus to evaluate a recommendation algorithm, we ask it to predict some hidden interactions in the test set and measure its ability to satisfy certain target objectives like the accuracy of its predictions.

K-fold cross validation is a common approach. It consists in partitioning the dataset into K subsets. Each subset is retained as test set by turns, and the rest is used as training set. The leave-one-out cross-validation is a special case of the K-fold cross validation where K equals the number of users in the data set.

Offline approaches have the advantage of being quick, economical and easy to conduct on a large amount of data, several data sets, and with multiple algorithms. They require no interaction with real users. Moreover, when the dataset includes timestamps about the actions, it is even possible to repeat all the interactions. Among the downsides of offline approaches, we can first cite the fact that they can only answer a very narrow set of questions, typically the ones about the prediction power of an algorithm. Shany and Gunawardana led a large discussion about this point in [123].

2.2.1.2 User studies

A user study is conducted by recruiting a set of test subjects, and asking them to perform several tasks requiring an interaction with the recommendation system. While they perform the tasks, a reporting of their behavior is done which gives quantitative and qualitative measures. Quantitative measures can refer to what portion of task a subject completed and the time he took to perform it, while the qualitative ones are indirect observations like the satisfaction for the user interface or the perception of the recommendations by a subject.

Compared to offline evaluation, user studies can answer a wider set of questions while allowing to test the behavior of users during their interacting with the recommendation system. It is the only approach for collecting qualitative data, which are often crucial for interpreting the quantitative ones. User studies however have some disadvantages. First they are very expensive to conduct in terms of financial compensation. Choosing a large set of subjects and asking them to perform a large enough set of tasks is also costly from the viewpoint of user time. Moreover, each chosen scenario has to be repeated several time in order to expect reliable conclusions, what may limit the range of distinct tasks that can be tested.

Let us notice that, even when the subjects may represent properly the true population of users, the results can still be biased because they are aware that they are participating in an experiment. For example, it is well known that paid subjects tend to satisfy the person or company conducting the experiment. If the subjects are aware of the hypothesis that is tested, they may unconsciously provide evidence that supports it. To accommodate that, it is typically better not to disclose the goal of the experiment prior to collecting data [123].

2.2.1.3 Online Evaluation

In an online evaluation, real users interact with a running recommender system. Many real word systems wish to positively influence the behavior of their users with the recommendations they give. This is why online testing systems are employed to compare different algorithms [73, 123]. Typically, such systems redirect, usually randomly, a small percentage of the traffic to different alternative recommendation engine, and record the users interactions with the systems.

Online evaluation is time consuming and difficult, and it may be risky in some situations. Shany and Gunawardana mention the fact that a test system that provides irrelevant recommendations may discourage the test users from using the real system ever again [123]. Thus, the experiment can have a negative effect on the system, which may be unacceptable in commercial application. They argued that for all these reasons, it is best to run an online evaluation last, after an extensive offline study provides evidence that the candidate approaches are reasonable, and perhaps after a user study that measures the user's attitude towards the system. In their opinion such a gradual process reduces the risk in causing significant user dissatisfaction.

2.2.2 Evaluation measures

According to what challenging topic or task of interest we want to evaluate a RS, many quality measures have been proposed. We give here a brief state-of-the-art of the measures relative to the following challenging topics:

– the accuracy of predictions
– the coverage over all the items
– the diversity of the recommendations
– their novelty for the users

Let us notice that we only present the measures commonly-used by the research community. Furthermore, the decision on the proper evaluation measure to use is often critical, as each of them may favor a particular algorithm.

2.2.2.1 Prediction Accuracy

Prediction accuracy is by far the most discussed property in the recommendation system literature. Basically, the majority of recommender systems try to predict user opinions over items (e.g. ratings of movies) or the probability of usage (e.g. purchase). The main purpose of recommender systems is to predict users' future likes and interests. A basic assumption is that a recommender system that provides more accurate predictions will be preferred by the user (although other properties like scalability have to be considered too). Providing better predictions is the most-faced challenge as the million dollar programming prize showed [14].

Prediction accuracy is typically independent of the user interface, and can thus be measured in an offline experiment. There are three broad classes of prediction accuracy measures: (i) measuring the accuracy of ratings predictions, (ii) measuring the accuracy of usage predictions, and (iii) measuring the accuracy of rankings of items [62, 57, 123]. We discuss here these classes of prediction accuracy measures.

2.2.2.1.1 Predictive accuracy measures

Many online retailers like Amazon.com or Netflix allow their users to rate the items they offer (e.g. 1-star through 5-stars). These ratings may help to discover which items would be well-suited for a user. Thus, given a user and the ratings that he gave to some items, RS have to predict the rating he would give to a new item.

In such cases, some measures of predictive accuracy of a RS were proposed. The *Mean Absolute Error* (MAE) is perhaps the most popular measure. It is used to measure the closeness of predicted ratings to the true ratings. The lower it is, better is the predictions of a recommender system. If r_{ui} is the true rating on item i by user u, $f(u,i)$ the predicted rating and $\mathcal{T}$ the set of all hidden user-item ratings (i.e., a set of triples (u, i, r_{ui})), the MAE is defined as

$$MAE = \frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} |r_{ui} - f(u,i)| \tag{2.1}$$

Another popular measure is the *Root Mean Squared Error* (RMSE). It becomes specially popular with the Netflix prize [127, 74, 15]. The RMSE is defined by the following formula:

$$RMSE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (r_{ui} - f(u,i))^2} \tag{2.2}$$

One special effect of RMSE, with the use of the square function, is that it penalizes large errors.

There are also some normalized versions of these measures. We can cite the Normalized MAE (NMAE) and the one of RMSE (NRMSE). They use the range of the ratings (i.e. $r_{max} - r_{min}$) as the normalization value. Let us notice that since they are simply scaled versions of RMSE and MAE, the resulting ranking of algorithms remains the same. As these measures focus only on the predicted ratings and do not matter of the positions of items in the recommendation list, they are not optimal for some common tasks such as finding a small number of items that are likely to be appreciated by a given user. Moreover, ratings do not always exist in all recommendation applications.

2.2.2.1.2 Usage measures

As we said above, in many applications the recommendation system does not predict the user's preferences of items, such as movie ratings in Netflix, but tries to recommend to users items that they may use. Let us take the case of Amazon. When a user add an item to its basket, their system suggests a list of items that may interest the user for purchase, and then increase their profit and customer loyalty. In this case, the interest is whether their system properly predicts that the user will add these items to his basket.

Offline evaluation of usage prediction consists in hiding a test set $\mathcal{T}$ of items used by some users [5], then asking the recommender system to suggest sets of items that these users will use. When the predictions are made, their usage by the users are measured. Let $\mathcal{T}_u$ be the set of items that a user u has used (a part of the test set) and S_u the set of items that the system recommended for him. Three measures are widely used for usage evaluation:

5. We keep this definition of $\mathcal{T}$ for all the rest of this chapter

– *precision*, which indicates the proportion of relevant recommended items from the total number of recommended items

$$precision = \frac{1}{|\mathcal{T}|} \sum_{u \in \mathcal{T}} \frac{|\mathcal{T}_u \cap S_u|}{|S_u|} \tag{2.3}$$

– *recall*, which indicates the proportion of relevant recommended items from the number of relevant items

$$recall = \frac{1}{|\mathcal{T}|} \sum_{u \in \mathcal{T}} \frac{|\mathcal{T}_u \cap S_u|}{|\mathcal{T}_u|} \tag{2.4}$$

– and the F_α-measures, a family of measures which are a combination of precision and recall

$$F_\alpha = (1 + \alpha^2) \times \frac{precision \times recall}{(\alpha^2 \times precision) + recall} \tag{2.5}$$

F_1-measure is the most used among the F_α-measures.

2.2.2.1.3 Ranking measures

When the list of recommendations is large, users usually tend to give greater importance to the first items because of their limited patience. Thus, the mistakes incurred with the first items may be considered more serious than those with the last items on the list. Indeed, many applications impose to their users a certain natural browsing order with the presentation of their lists of recommendations, typically as vertical or horizontal list.

In such application cases, it is very important to ensure a good ranking of the items of the list in order to allow users to find as soon as possible the interesting items. Furthermore, users may become impatient after leafing through a certain number of uninteresting items, and thus leave off the navigation.
Some ranking measures take account of this situation. Among the most often used ranking measures, there are the following standard information retrieval measures: (a) *Half-life utility* [20], which assumes an exponential decrease in the interest of users as they move away from the first recommendations; and (b) *Discounted cumulative gain* [67], wherein decay is logarithmic.

Half-life utility (HL) was firstly introduced by Breese et al [20]. It attempts to evaluate the utility of a recommendation list to a user. Breese et al define the expected utility of a list as the probability of viewing a recommended item times its utility. They rely on the assumption that the likelihood that a user examines a recommended item decreases exponentially with the item's ranking and then, for lists of N items, they formulate HL as follows:

$$HL = \frac{1}{\mathcal{T}} \sum_{u \in \mathcal{T}} \sum_{p=1}^{N} \frac{\max(r_{ui_p} - d, 0)}{2^{(p-1)/(h-1)}} \tag{2.6}$$

Here, r_{ui_p} represents the true vote of the user u for the item i_p (i.e., at the p^{th} position of the list) and d the default or neutral vote. Their difference stands for the utility of the item for the user. In the case of ratings prediction, r_{ui_p} denotes the rating given by the user to the item (e.g. 4 stars), and d is the default vote (e.g. 3 stars). In the case of usage prediction, r_{ui_p} is typically 1 if the user selects (e.g., purchase) the item and 0 otherwise, while d is 0.

h is called the "half-life". It is the rank of the item on the list for which there is a 50% chance that the user will eventually examine it. For example, if h is set to 5, the first five items have 50% chance to be considered y the user.

The basic idea of Discounted cumulative gain (DCG) is that highly relevant items appearing lower in a ranked list should be penalized as the graded relevance value is reduced logarithmically and proportional to the position of the item. The DCG of a ranked list of N items is defined as:

$$DCG = \frac{1}{\mathcal{T}} \sum_{u \in \mathcal{T}} \left(rel_{u,1} + \sum_{p=2}^{N} \frac{rel_{u,p}}{\log_2(p)} \right) \tag{2.7}$$

where $rel_{u,p}$ is the graded relevance of the item at position p. It may be set to 1 for a relevant item and zero otherwise.

Besides accuracy there are a number of other dimensions that can be measured. We present below some of them that are important to consider.

2.2.2.2 Coverage

Most commonly, the term coverage refers to the proportion of items that the recommendation system can recommend, the item space coverage. However it can also designate the proportion of users for which the system can recommend items. Indeed in some applications, the recommender may not provide recommendations for some users due to, e.g. low confidence in the accuracy of predictions for that user. In such cases one may prefer RS that can provide recommendations for a wider range of users [38, 62]. Of course, we should evaluate such recommenders on the tradeoff between coverage and accuracy.

In this section we speak about item space coverage. A simplest measure is to compute the percentage of all items that can ever be recommended. Therefore denoting the total number of distinct items in the top K places of all recommendation lists as N_K, the K-dependent coverage is defined as

$$COV_K = \frac{N_K}{N} \tag{2.8}$$

Low coverage indicates that the algorithm can access and recommend only a small number of distinct items, usually the most popular ones. This quite often results in little diverse recommendations and leads to the long tail effect[6]. For instance, in music, the

6. `http://en.wikipedia.org/wiki/Long_tail`

Long Tail is composed of a small number of popular items, the well-known hits, and the rest are located in the heavy tail, those that do not sell as well. On the other hand, algorithms with high coverage are more likely to provide diverse recommendations. From this point of view, coverage can be also considered as a diversity metric [84].

In some cases it may be desirable to weight the items (e.g., their popularity or utility). Although this method tends to discard some items which are very rarely used anyhow, it has the advantage of keeping high profile items whose absences may be less tolerable. The sales diversity of Fleder and Hosanaga [35] belongs to such coverage measures. It evaluates how unequally different items are chosen by users when a particular recommender system is used. Let $\rho(i)$ be the proportion of user choices that the item i accounts. The *Gini Index* is defined as follows:

$$G = \frac{1}{K-1} \sum_{p=1}^{K} \rho(i_p)\,(2j - K - 1) \tag{2.9}$$

where i_p denotes the item at position p in the list. The items are ordered according to an increasing $\rho(i)$. The index is 0 when all items are chosen equally often, and 1 when a single item is always chosen. Another measure of distributional inequality is the *Shannon Entropy*:

$$H = -\sum_{p=1}^{K} \rho(i_p) \log(\rho(i_p)) \tag{2.10}$$

The choice of a particular measure to evaluate some recommender systems depends on the goals that the system is supposed to fulfill. One may specify different goals which further complicates the evaluation process. For a better overview, we refer the reader to [62, 29, 38, 123].

2.2.2.3 Diversity

From the fact that there is a little value in recommending a notorious item (although it is expected to be relevant), the consideration of novelty and diversity in recommendation becomes increasingly present [95]. Let us note that, currently, novelty and diversity measures do not have standard definition an measures. Therefore, different approaches are proposed in the literature. We present here some of the most-quoted of them [137, 153, 84, 18].

The diversity quality measure evaluates how much are different, with respect to each other, the items of a recommendation list. Two levels of diversity can be taken: (i) the inter-user diversity which estimates the diversity between recommendation lists presented to users and (ii) the intra-user diversity which evaluates the diversity within a recommendation list.

The inter-user diversity measures allow to compare the ability of various algorithms to return different lists to different users. Given two users u and v and K items to recommend, the difference between their recommendation lists, here L_u and L_v, can be measured by the *Hamming distance* as

$$H_{uv} = 1 - \frac{|L_u \cap L_v|}{K} \tag{2.11}$$

The hamming distance is equals to zero if the two lists are identical, and 1 if they are completely different. An average over all the users gives the mean Hamming distance of a recommendation algorithm.

As for intra-user diversity measures, they estimate the extent to which an algorithm can provide diverse items to each user separately. Notably, they can be used to enhance improve recommendation lists by avoiding recommendation of excessively similar items [160]. As a measure, one can rely to the average similarity of items in a recommendation list given by

$$IUD_u = 1 - \frac{\sum\limits_{i \in L_u} \sum\limits_{j \in L_u, i \neq j} sim(i,j)}{K(K-1)} \tag{2.12}$$

These measures can be also averaged over all users to get a mean diversification value of a recommender system. The higher is the obtained value, the more diverse items the system can recommend together.

2.2.2.4 Novelty

The novelty amounts to the factor to agreeably surprise the users. In practice, authors take it as the degree of difference between the recommended items and the ones that the users have already know. The simplest way to quantify the ability of an algorithm to generate novel and unexpected items is to measure the average popularity (e.g., the ratio of the number of users who see the items before) of the recommended items

$$Nov = 1 - \frac{1}{K\,|\mathcal{T}|} \sum_{u \in \mathcal{T}} \sum_{i \in L_u} pop(i) \tag{2.13}$$

In this equation, $pop(i)$ is the popularity of the item i. Lower popularity leads to higher novelty in recommendation. Another way is to assume the chance that a randomly-selected user collects an item i is close to $pop(i)/\,|U|$, with U is the set of users. The item's self-information of novelty is $\log_2(|U|\,/pop(i))$. From this, we can take the novelty of a recommended list as the average of those of its items and then we have

$$Nov = \frac{1}{K\,|\mathcal{T}|} \sum_{u \in \mathcal{T}} \sum_{i \in L_u} \log_2\left(\frac{|U|}{pop(i)}\right) \tag{2.14}$$

2.3 Conclusion

In this chapter, we talked over numerous challenges of recommender systems and evaluation methods and measures. Recommendation accuracy is known as the most studied RS problem. In the literature, researchers propose more and more complex techniques in order to improve recommendations. However the ratio between the computation time and the yielded improvement is often not acceptable. For instance, Netflix offered a grand prize of US $1 million for an algorithm that is 10% more accurate than the one they use to predict customers' movie preferences (aka, Cinematch [14]), but despite they spent so much money, they never used the final proposed solution due to its lack of scalability [8].

Part II

The contributions of the thesis

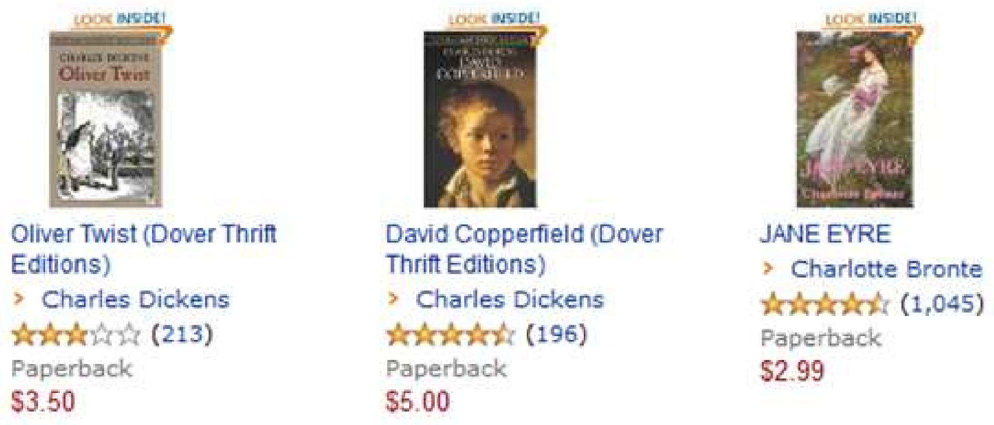
3 | Using Cluster-based Biases for Dynamic Recommendations

The main purpose of recommender systems is to predict user preferences on a large selection of items. They try to find items that are likely to be of interest for the user. Because the user is often overwhelmed by the considerable amount of items provided by electronic retailers, the predictions are a salient function of all types of e-commerce [121, 14].

In Chapter 1, we introduced collaborative filtering which is a widely-used category of recommender systems. It consists in analyzing relationships between users and interdependencies among items to identify new user-item associations [127, 76, 104], and based on these associations, recommendations are inferred.

We described briefly Matrix Factorization (MF), one of the most popular dimensionality reduction techniques in recent years. In the literature, it is well investigated and many types of factorization have been proposed [76, 70, 86]. In its basic form, matrix factorization characterizes both items and users by vectors of factors (also called "latent factors") inferred from user feedback patterns. Thus both users and items are mapped into a latent factor space. Figure 1.1 in Subsection 1.3.2.2.1 presents an illustration of such a space of factors. The affinity degree between a user's factors and an item's ones determines the position of the latter in the final list of recommendations for the user.

Although matrix factorization is very popular because it gives good scalability at the time of recommending while allowing remarkable prediction accuracy, some shortcomings remain. One of these is the fact that the model generated by MF is static. Once it has been generated, the model delivers recommendations based on a snapshot of the incoming ratings frozen at the beginning of the generation. To take into account the missing ratings (that arrived after the last model generation), the model has to be computed periodically. We propose here a solution that reduces the loss of quality of the recommendations over time. We include into our matrix factorization model some stable biases which track users' behavior deviation [104, 132]. The biases are continuously updated with the new ratings, in order to maintain a satisfactory quality of recommendations for a longer time. Our solution is based on the observation that the rating tendency of a user is not uniform, and can change from one set of items to another. A set of biases is then associated with each user, one bias for each set of similar items. Thus, the integration of



Oliver Twist (Dover Thrift Editions)
› Charles Dickens
★★★☆☆ (213)
Paperback
$3.50

David Copperfield (Dover Thrift Editions)
› Charles Dickens
★★★★☆ (196)
Paperback
$5.00

JANE EYRE
› Charlotte Bronte
★★★★☆ (1,045)
Paperback
$2.99

Figure 3.1: Amazon's five star widget

a new rating is provided by recomputing a local user bias (a bias of a user for a specific cluster of items), which may be done with a very low computation cost.

Our approach improves the scalability of recommender systems by reducing the frequency of model recomputations. The experiments we conducted on the Netflix dataset and the largest MovieLens dataset confirmed that our technique is well adapted for dynamic environments where ratings happen continuously. The cost of the integration of new ratings is very low, and the quality of our recommendations does not decrease very fast between two successive matrix factorizations. Also our idea of refining the user biases is orthogonal to the factorization models. It can be used in fully-fledged models with weights, temporal dynamics and so on [75, 76, 132, 13]. Morever our factorization model is easily parallelizable as we will describe shortly.

In the remainder of this chapter, we give some preliminaries in Section 3.1. Then we state the problem more so in Section 3.2 before summarizing related work in Section 3.3. In Section 3.4, we detail our cluster-based matrix factorization solution. We present an experimental analysis of our proposal in Section 3.6, and conclude this chapter in Section 3.7.

3.1 Preliminaries

Recommender systems learn the affinities from the users' expressed interests in items. These interests are often given through ratings which measure how much a user likes a rated item. Most of the time, this interest is represented by numerical values from a fixed range. A set of interfaces, e.g. widgets, are used to allow the users to rate the items. The ones used to enter ratings at a 1-to-5 star scale are still very popular on the web. Figure 3.1 shows the one of the well-known online retailer Amazon.com. Hence we can formalize the prediction problem as follows. Let us consider a set U of users and a set I of items. User ratings can be seen as tuples (u, i, r_{ui}, t_{ui}), where u denotes a user,

i denotes an item, r_{ui} the rating of user u for the item i, and t_{ui} is a timestamp. We assume that a user rates an item at most once.

The challenge is to predict the future ratings such that the difference between an estimated rating $f(u, i)$ and its true value r_{ui} is the lowest possible.

The ratings can be arranged into a sparse matrix R where its columns represent the users and its rows the items. The value of each not empty cell c_{ui} of R, corresponding to user u and item i, is a pair of values (r_{ui}, t_{ui}) with r_{ui} the rating given by u to the item i at time t_{ui}[1]. An empty, i.e. missing, cell c_{ui} in R indicates that user u has not yet rated item i. Hence, the task of recommender system is to predict these missing rating values. The table below represents such a matrix.

	u_1	u_2	$\dots$	u_n
i_1	3		$\dots$	1
i_2		2	$\dots$	5
i_3	1		$\dots$	
i_4			$\dots$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
i_m		4	$\dots$	2

In its basic form (Basic MF), matrix factorization techniques try to capture the factors that produce the different rating values. They approximate the matrix R of existing ratings as a product of two matrices P and Q which contain vectors of factors for the profiling of the users and the items respectively

$$R \approx P \cdot Q \tag{3.1}$$

Let us note that these matrices of factors are much more smaller than R. Thus, we gain in dimension while getting predictive ratings simply by the following formula

$$f(u, i) = p_u \cdot q_i^T \tag{3.2}$$

with p_u and q_i the vectors of factors corresponding to user u and item i respectively in P and Q.

In practice, it is very difficult to obtain exactly R with the product of P and Q. Usually, some residuals remain. These latter constitute the reconstruction error, i.e. its inaccuracy, which can be represented by a matrix E of errors having the same size than R. Then Equation 3.1 can be changed to

$$R = P \cdot Q + E \tag{3.3}$$

We can see that the more the matrix E is close to a zero matrix, the more accurate will be the prediction. The process of training looks for the better values of P and Q such

1. In practice, more sophisticated data structures are used in order to alleviate the memory consumption.

that the matrix E is the closest possible to a zero matrix. Thus, it tries to adjust all the values e_{ui} of the matrix E to zero using a stochastic gradient descent (SGD) algorithm. The SGD algorithm computes a local minimum where the total sum of error values is one of the lowest according to initial ratings. In other words, it tries to minimize as small as possible the sum of quadratic errors $\sum\limits_{ui} e_{ui}^2$ between the predictive ratings $f(u,i)$ and the real ones r_{ui}. Errors are squared in order to avoid the effects of negative values in the sum, and increase the weights of abnormal values. The fact of minimizing $\sum\limits_{ui} e_{ui}^2$ amounts to minimize each e_{ui}^2.

We have $e_{ui} \overset{def}{=} r_{ui} - f(u,i)$. By using the vectors of factors p_u and q_i, we get $e_{ui} \overset{def}{=} r_{ui} - p_u \cdot q_i^T$. If we denote by K the number of considered factors, we can avoid overfitting the observed data by regularizing the squared error of known ratings. Thus we have the next regularized sum of squared errors

$$\sum_{ui} e_{ui}^2 = \sum_{ui} (r_{ui} - p_u \cdot q_i^T)^2 + \beta \cdot (\|p_u\|^2 + \|q_i\|^2) \tag{3.4}$$

β is a regularization parameter which serves to prevent large values of p_{uk} and q_{ki}. More precisely, we have

$$\sum_{ui} e_{ui}^2 = \sum_{ui} (r_{ui} - \sum_{k}^{K} p_{uk} q_{ki})^2 + \beta \cdot (\|p_u\|^2 + \|q_i\|^2) \tag{3.5}$$

Then to minimize the quadratic errors, in order to get better predictions, we compute the differential (i.e., the gradients) of the squared error e_{ui}^2 to determine the part of change due to each factor (p_{uk} and q_{ki}):

$$\frac{\partial e_{ui}^2}{\partial p_{uk}} = -2 \cdot e_{ui} \cdot q_{ki} \ , \qquad \frac{\partial e_{ui}^2}{\partial q_{ki}} = -2 \cdot e_{ui} \cdot p_{uk} \tag{3.6}$$

We update p_{uk} and q_{ki} in the opposite direction of the gradients in order to decrease the errors and thus obtain a better approximation of the real ratings.

$$p_{uk} \leftarrow p_{uk} + \alpha \cdot (2 \cdot e_{ui} \cdot q_{ki} - \beta \cdot p_{uk}) \tag{3.7}$$

$$q_{ki} \leftarrow q_{ki} + \alpha \cdot (2 \cdot e_{ui} \cdot p_{uk} - \beta \cdot q_{ki}) \tag{3.8}$$

α is a learning rate. The SGD algorithm iterates on Equations 3.4, 3.7 and 3.8 until the regularized sum of the quadratic errors in Equation 3.4 does not decrease any more. This process corresponds to the training step.
After this training, the predictions $f(u,i)$ are computed through the products $p_u \cdot q_i^T$ of both vectors of factors. A sorting step allows to find the most relevant items to recommend to each user, i.e. the items with the greatest product values.

3.2 The Dynamicity Problem

As mentioned earlier, the model generated by matrix factorization is static. Once it has been generated, the model delivers recommendations based on a snapshot of the incoming ratings frozen at the beginning of the generation. Therefore, it has to be computed periodically in order to take into account the missing ratings which arrived after its generation. However, it is not realistic to run the model frequently, because of the high cost of its computation. The latter is $O(|\mathcal{T}| \cdot K \cdot n)$ where $\mathcal{T}$ is the set of user-item ratings and n the number of iterations for minimizing the quadratic errors. Real-word applications like Amazon or Netflix handle several billions ratings from millions of users and hundred of thousands items, thus the factorization model can not be recomputed every time. In the conclusion of our previous chapter, we gave the example of Netflix which offered a grand prize of US $1 million for an algorithm that is 10% more accurate than the one they use to predict customers' movie preferences, called Cinematch [14]. However despite they spent so much money, they never used the final proposed solution due to its lack of scalability [8]. Therefore, the quality of the recommendations will decrease gradually until a new model is computed.

In real-world contexts where new ratings happen continuously, users profile evolve dynamically. Consider, for instance, a costumer of an online music-store looking for good pop songs. He asks the application for some recommendations and the system proposes to him a short list of songs with high probability of interest (based on the latest available model). The costumer selects and rates the songs he already knows or he just listened to, and asks for new recommendations. Since the preferences of the customers evolve accordingly to the songs they have listened to, it is important to be able to integrate the new ratings for the subsequent recommendations. Otherwise, the accuracy of these recommendations will be low.
Online shops attempt to keep their customers loyalty and thus search to better satisfy them by providing relevant recommendations. This accounts for all attention brought to the evolution of user preferences. Indeed it has been claimed that even an improvement as small as 1% of the accuracy leads to a significant difference in the ranking of the top-K most recommended items for a user [74, 31]. Thus the decreasing of the accuracy of predictions is no longer acceptable. To face it, recommender systems must either regularly recompute their models, which represents an expensive task in terms of computation time, but can be alleviated by distributing computation, or by using online-updating methods which allow to take account of new ratings with a low cost of computation. Hence, the dynamicity problem can be defined as follows: *how to integrate new ratings in the prediction model as fast as possible?* The goal is to consistently maintain the accuracy of the predictions at a good level.

We present in the following some important requirements that any solution for the dynamicity problem has to satisfy.

1. **Recommendation quality** Assuming some fixed sets of users and items. We consider users continuously asking for items, and rating them. For instance, a

user asks for a short list of items with high probability of interest (i.e. high predicted rating), then selects and rates some of them, and so on. In such online recommendation scenario, the user expects the recommended items to be of high interest. We measure the quality of service in terms of the Root Mean Square Error (RMSE) between the predicted and the real ratings. We express the user requirement for quality, as a constraint on the RMSE, which value must be greater than a given threshold ϵ.

$$RMSE < \epsilon \tag{3.9}$$

2. **Response time** Another requirement for online recommendation is the response time tolerated by the end users. When a user asks for a recommendation, he expects to receive it almost immediately. Such requirement for online user demand is usually described by an upper bound along with a ratio of appliance [136]: 90% of the demands must be served in less than 5 seconds. This response time constraint forces us (1) to generate the model in advance, in order to anticipate the future demands, and (2) to limit the computational cost needed for the integration of the ratings that arrived after the model generation.

   We can summarize the performance requirements into the following challenge: design a recommendation system which provides sufficient quality, when generating the "top-quality" model takes a long time, when the predictions quality is decreasing over time, featuring fast recommendation delivery on user demand, and reducing the overall computation cost.

   Our solution to tackle this challenge is based on the following process:

   a) Combine clustering, MF and bias adjustment, to take into account the specificity of each user and start with a high quality model.

   b) Continuously update the biases (with a low computation cost), in order to maintain as long as possible the quality of the predictions at satisfactory level.

In the following we will present related work, then we detail our solution for the first two points.

3.3 Related Work

In the literature, two approaches can be used to face in a certain manner the dynamicity problem. The first approach adresses distributed[2] matrix factorizations, and studies how to alleviate the factorization's computation cost. Many techniques have been proposed [110, 41, 111, 156, 146]. They allow to significantly reduce the cost of factorizing. However the generated models remain static and take into account only the ratings available at the beginning of their generation. Furthermore, the ability of

2. Let us note that we use the term "distribution" and its derivatives but it implies both parallel and distributed computing. The reader can get more information about these notions on `http://en.wikipedia.org/wiki/Distributed_computing`

distributing the factorization process do not always allow to carry out up-to-date models frequently.

The second approach which may be considered as complementary of the first one, is to online-update users' factors (sometimes items' factors also, like in [112]) in order to dynamically follow the trends. Moreover this approach allows, in case of limited computational resources, to postpone as far as possible the need of recomputing the model.

We present below some popular state-of-the-art techniques of these two approaches.

3.3.1 Distributed MF techniques

By distributing the matrix factorization, we try to improve the computation time needed to learn a model. The main difficulty here is how to make parallelized-update on factors p_{uk} and q_{ki} (as in Equations 3.7 and 3.8) while ensuring potential access conflicts.

Niu et al. propose in [110] a lock-free algorithm called HogWild. Based on the intuition that the probability of updating the same factor in P or Q is small when the matrix R is very sparse, they assume that the overwriting issue can be ignored. From this viewpoint, they have just to randomly select a subset of ratings r_{ui} instances and apply updates simultaneously in all the threads (without synchronization between them). HogWild is very efficient compared to delayed-updating approach like in [5, 161]. However, it is designed only for shared-memory systems.

The distributed SGD (DSGD for short) investigated in [41, 111] partitions the matrix R of ratings into blocks as illustrated in Figure 3.2. Independent blocks constitute a

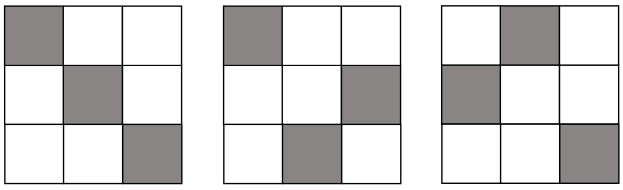


Figure 3.2: Patterns of independent blocks for a 3-by-3 gridded matrix

stratum (in gray color). In Figure 3.2, we represent such strata from a 3-by-3 gridded matrix. Due to the fact that the blocks of the same stratum do not share neither any row (i.e., item) nor a column (i.e., user), they can be updated in parallel at the same time. Thus, DSGD can be regarded as an exact SGD implementation with a specific ordering of updates.

One drawback of DSGD is that it suffers from the locking problem. For a parallel algorithm aiming to maximize computation performances, always keeping all execution nodes (e.g., threads) busy is important. The locking problem occurs if an execution node idles because of waiting for others. As DSGD has a stratum-by-stratum execution model, the execution nodes are synchronized. Thus, when an execution node ends earlier its computation on a block of the current stratum, it has to wait for the other nodes before starting together on another stratum. This naturally follows from the varying of their running time specially if R is unbalanced. Moreover, execution nodes must exchange the factors' values they computed when they pass from a stratum to another.

More recently, Zhuang et al. present in [156] some ways to overcome these shortcomings. First they target shared-memory systems, hence the threads do not have to exchange some data. They access concurrently to data. In addition to the use of some methods to balance more R like random shuffling [3], their main idea may be summarized as using less threads than there are blocks in a stratum. Therefore, when a thread earlier finishes processing a block, a scheduler can assign it a new block that meets some defined criteria. Among the latter we can cite the fact that this block is free and its number of past updates is the smallest among all free blocks.

With such an approach Zhuang et al. developped a fast parallel SGD method for matrix factorization, but as Niu et al., they are restricted to shared-memory systems. Only the initial DSGD approach avoids this limitation.

Let us notice that our proposition that we will present in Section 3.4, can be easily adapted to all these techniques of distributed computing.

3.3.2 Online-updating approaches

The problem of the integration of the incoming ratings is not well investigated in the literature. We can cite Sarwar et al. in [120]. They deal with "new user/item" problem, which aims at integrating newly registered users and items (and their ratings). Even though this problem includes the integration of new rating, its special nature requires specific solutions. In our approach, we only deal with the new ratings of known users and items.

Rendle et al. focus on users (and items) which have small rating profiles [112]. They present an approximation method that updates the matrices of an existing model (previously generated by MF). The proposed *UserUpdate* and *ItemUpdate* algorithms retrain the factor vector for the concerned user, or item, and keep all the other entries in the matrix unchanged. The time complexity of this method is $O(|V(u,.)|\,.k.t)$, where k is the given number of factors and t the number of iterations. The whole factor vector of the user is retrained (i.e. his rating profile for all the items). In Section 3.5 we will discuss and compare our solution with theirs. We show also that our solution is faster.

Agarwal et al. propose in [6] a fast online bilinear factor model called FOBFM. It uses an offline analysis of item/user features to initialize the online models. Moreover, it computes linear projections that reduces the dimensionality and, in turn, allows to learn fast both user and item factors in an online fashion. Their offline analysis uses a large amount of historical data (e.g., keywords, categories, browsing behavior) and their model needs to online learn both user and item factors in order to integrate the new ratings. So, their technique is much more costly than ours. Furthermore, our approach

3. Randomly permuting users' columns and items' rows before processing in order to have a better balanced matrix. The interest is to have blocks with closer amount of ratings.

can work even in applications where no item/user features are available which is not proven in the experimentations of the FOBFM model.

Cao et al. [25] point the problem of data dynamicity in latent factors detection approaches. They propose an *online* nonnegative matrix factorization (ONMF) algorithm that detects latent factors and tracks their evolution when the data evolves. Let us remind that a nonnegative matrix factorization is a factorization where all the factors in both matrices P and Q are positive. They base their solution on the *Full-Rang Decomposition Theorem*, which states that: for two full rank decompositions $P_1.Q_1$ and $P_2.Q_2$ of a matrix R, there exists one invertible matrix X satisfying $P_1 = X.P_2$ and $Q_1 = X^{-1}.Q_2$. They use this relation to integrate the new ratings. Although the process seems to be relatively fast, its computation time is greater than ours. This is due to the fact that their technique updates the whole profiles of all the users where our solution limits the computations to the bias of the concerned user.

3.4 Making Dynamic Recommendations

As we said above, we focus on dynamic contexts where new ratings are continuously produced. In such case, it is not possible to have an up to date model, due to the incompressible time needed to compute the recommendation model even if we use distributed matrix factorization. At least, the ratings produced during the model computation will be missing. After each generation of a new model, the situation can degrade quickly enough since the number of non processed ratings may increase very fast. Then, a growing loss of quality can be observed in the recommendations, as long as the static model is used.

To tackle this problem, our model relies on biases which are among the most overlooked components of recommender models [70]. Biases allow to capture a significant part of the observed rating behavior. We first cluster the items according to their similarities. For that, one can either rely on clustering algorithms as we did in [54, 51] or, if they are available, on items' categories given by certain of their attributes. We combine global user biases with local user biases defined on each set of similar items. The local user biases allow to refine user's tendency on small sets of items, whether the global biases capture the general behaviors of the users. In case where the local user bias has not enough information (i.e., ratings) to be sound, the global user bias plays a role of balance. It ensures, in the worst case, that user's tendency will follow his general behavior.

In the following, we explain how to integrate users' biases in a matrix factorization's model. Then we highlight the importance of using local biases, then we detail our proposed solution which combines global biases and cluster-based local biases. And lastly, we present the algorithm that integrates the new ratings by adjusting the local biases in the recommendation model.

3.4.1 Biased MF

Several improvements of the basic matrix factorization technique that we presented above are proposed in the literature. One of them assumes that much of the observed variations in the rating values is due to some effects associated with either the users or the items, independently of any interactions [133, 76, 104]. Indeed, there are always some users who tend to give higher (or lower) ratings than others, and some items may be higher (or lower) rated than others, because they are widely perceived as better (or worse) than the others. Basic MF can not capture these tendencies, thus some biases are introduced to highlight these rating variations. We call such techniques *Biased MF*. The biases reflect users or items tendencies. A first-order approximation of the biases involved in rating r_{ui} is as follows:

$$b_{ui} = \mu + b_u + b_i \tag{3.10}$$

b_{ui} is the global effect of the considered biases, it takes into account users tendencies and items perceptions. μ denotes the overall average rating (for all the items, by all the users). b_u and b_i indicate the observed deviations from the average of user u and item i, respectively. Hence, Equation 3.2 becomes

$$f(u,i) = p_u \cdot q_i^T + b_{ui} \tag{3.11}$$

Since biases tend to capture much of the observed variations and can bring significant improvements, we consider that their accurate modeling is crucial [104, 76]. As for the factors p_{uk} and q_{ki} (Equations 3.7 and 3.8), the biases have to be refined through a training step using the following equations:

$$b_i \leftarrow b_i + \alpha \cdot (2 \cdot e_{ui} - \lambda \cdot b_i) \tag{3.12}$$

$$b_u \leftarrow b_u + \alpha \cdot (2 \cdot e_{ui} - \lambda \cdot b_u) \tag{3.13}$$

where λ is a regularization parameter. It plays the same role as β in equations 3.7 and 3.8. λ allows us to assign different contributions to the user's biases and factors.

Let us notice that this definition of users' biases assumes their global behaviors are uniform, which is not the case. In our observation, and this is not surprising, the behavior of a user can change from one set of items to another. Hence it would be interesting to associate a set of local biases to each user with one bias per set of similar items. Our next section formalizes more the relation between the similarity of a set of items and the variance of users biases.

3.4.2 The interest of cluster-based local biases

We argued above that the accuracy of local user biases depends on the degree of similarity between the items in each set (i.e. cluster). We show here that the more similar are the items in each cluster, the more the variance of the local user biases is small. A smaller variance means stable users' behaviors, in other words, a lower prediction error and then a more accurate recommendation. We first assert and demonstrate the next lemma:

Lemma 3.4.1. *Given a set of items, more they are similar, lesser varying are the users' ratings about them.*

Proof. Let U be a set of users, I a set of items, r_{ui} a rating of a user $u \in U$ for an item $i \in I$, and μ the overall average of rating. Consider $I_u \subset I$, the set of items rated by a user u, then the bias b_u of the user u is defined as follows:

$$b_u = \frac{1}{card(I_u)} \sum_{i \in I_u} (r_{ui} - \mu) \tag{3.14}$$

For a given item $i \in I_u$, the local deviation of the user u relative to the overall average of rating μ is:

$$b_{ui} = r_{ui} - \mu \tag{3.15}$$

Then, Equation 3.14 can be simplified as:

$$b_u = \frac{1}{card(I_u)} \sum_{i \in I_u} b_{ui} \tag{3.16}$$

To measure the user bias variation, we compute for each user u his bias variance Var_u as follows:

$$Var_u = \frac{1}{card(I_u)} \sum_{i \in I_u} (b_{ui} - b_u)^2 \tag{3.17}$$

Then, equations 3.15, 3.16 and 3.17 lead to the following formula:

$$Var_u = \frac{1}{card(I_u)^3} \sum_{i \in I_u} \left(\sum_{j \in I_u} (r_{ui} - r_{uj}) \right)^2 \tag{3.18}$$

To compute the variance, the user must have at least two ratings. Then, the variance can be bound as shown in the following equation:

$$\begin{aligned} Var_u &\leq \frac{1}{2^3} \sum_{i \in I_u} \left(\sum_{j \in I_u} (r_{ui} - r_{uj}) \right)^2 \\ &\leq \frac{1}{8} \left(\sum_{i \in I_u} \sum_{j \in I_u} |r_{ui} - r_{uj}| \right)^2 \end{aligned} \tag{3.19}$$

Then, considering all the users we obtain:

$$0 \leq \sum_{u \in U} Var_u \leq \frac{1}{8} \left(\sum_{u \in U} \sum_{i \in I_u} \sum_{j \in I_u} |r_{ui} - r_{uj}| \right)^2 \tag{3.20}$$

Measuring the dissimilarity of items.
Consider two items $(i, j) \in I^2$, and let $U_{ij} \subset U$ be the set of users having rated both of them. The dissimilarity of the items i and j can be measured according to the difference of the ratings r_{ui} and r_{uj} given to them by each user u. Hence, we define the dissimilarity of two items $(i, j) \in I^2$ as follows:

$$dissim_{ij} = \sum_{u \in U_{ij}} |r_{ui} - r_{uj}| \tag{3.21}$$

$dissim_{ij}$ tends to zero when all the users in U_{ij} have close ratings for both items. The sum of the dissimilarities of all the couples of items is:

$$\sum_{(i,j) \in I^2} dissim_{ij} = \frac{1}{2} \sum_{i \in I} \sum_{j \in I} \sum_{u \in U_{ij}} |r_{ui} - r_{uj}| \tag{3.22}$$

Since $dissim_{ij} = dissim_{ji}$, we divide by 2 the sum in the right part of the previous equation.
Equations 3.20 and 3.22 lead to the following ascertainment on the dissimilarity of the items and the user bias variances:

$$0 \leq \sum_{u \in U} Var_u \leq \left(\sum_{(i,j) \in I^2} dissim_{ij} \right)^2 \tag{3.23}$$

For a given set of items, the less dissimilar (i.e., more similar) they are (i.e. $\sum_{(i,j) \in I^2} dissim_{ij} \to 0$), the less varying are the user biases (i.e. $\sum_{u \in U} Var_u \to 0$). In other words, the users tend to have uniform behaviors on such a set of similar items. □

Corollary 3.4.2. *Defining a local bias per user and on each set of similar items leads to a small variance in their behaviors and, consequently, a better accuracy in prediction.*

3.4.3 The CBMF model

Our cluster-based matrix factorization model (CBMF) is based on the observation that many users usually tend to underestimate (or overestimate) the items they rate. A user may have a tendency to rate above (or beyond) the average. We aim to quantify such tendency. A simple way to take it into account is to assign a single bias per user (as shown in section 3.4.1). However, the user's tendency is generally not uniform: it can change from one item to another. For some sets of items, a user can tend to rate close to the average. While for some other items (e.g., those she really likes/dislikes), the user fails to rate objectively, either using extreme ratings, or keeping moderated ratings.

To take into account this discrepancy, we define several biases per user, instead of a single one. We assign one bias b_u^C for each user u and each set C of similar items. We expect as demonstrated above that handling finer-grained biases will lead to more accurate recommendation. Once the clusters are built, we assign a vector of biases to each user. One bias for each group of items. Then, we apply our matrix factorization (CBMF) on the ratings to generate the recommendation model.

Thus, we come down to observe local ratings variation instead of a single global ratings variation as used in previous approaches [104, 76, 132]. We derive the local bias b_u^C of a user u for a cluster C from the ratings of the items in this cluster. For each item $j \in C$ that he rated, we define his deviation b_u^j as the difference between his rating for j and the sum of his global bias b_u and the average users' rating μ. The local bias of the user b_u^C, at the level of the cluster, is obtained by taking his average deviation as shown in Equation 3.24.

$$b_u^C = \frac{1}{|C|} \sum_{j \in C} r_{uj} - (\mu + b_u) \tag{3.24}$$

Thus, our prediction formula is the following:

$$f(u,i) = p_u \cdot q_i^T + \mu + \Big(b_u + \frac{1}{|\varsigma_i|} \sum_{C \in \varsigma_i} b_u^C\Big) + b_i \tag{3.25}$$

where ς_i denotes the sets of all clusters to which the item i belongs. Indeed when considering the properties of the items for the clustering, it often happens that an item appears in many clusters. For instance, if we consider movies, one can belong to several genres like Action, Fantasy and Drama. Thus the movie has to be put into these three classes/clusters.

Our regularized global sum of squared errors becomes:

$$\begin{aligned} \sum_{ui} e_{ui}^2 = \sum_{ui} \left(r_{ui} - (p_u \cdot q_i^T) + \mu + \Big(b_u + \frac{1}{|\varsigma_i|} \sum_{C \in \varsigma_i} b_u^C\Big) + b_i \right)^2 \\ + \beta \cdot (\|p_u\|^2 + \|q_i\|^2 + \left(\frac{1}{|\varsigma_i|} \sum_{C \in \varsigma_i} b_u^C \right)^2 + {b_u}^2 + {b_i}^2) \end{aligned} \tag{3.26}$$

Like the global biases b_u and b_i, the user's local biases b_u^C have to be refined using the formula:

$$b_u^C \leftarrow b_u^C + \alpha \cdot (2 \cdot e_{ui} - \gamma \cdot b_u^C) \tag{3.27}$$

Algorithm 1 details the steps of our CBMF process. In Line 1, we compute the initial bias value of each item, the global bias of each user and his local biases. Line 2 initializes the matrices of factors P and Q. This is done with random low values. Lines 3 to 13 correspond to the main part of the learning process. At each iteration (lines 4 to 11), the error of prediction e_{ui} is computed for each rating. Then, the matrices of factors, the biases (global and local ones) are adjusted accordingly (lines 6 to 10), using equations 3.7, 3.8, 3.12, 3.13, and 3.27. Line 12 measures the global error as indicated in Equation 3.26. The training process ends when the regularized global squared error does not decrease any more or when the maximum number of iterations is reached.

3.4.4 Integration of incoming ratings

After the generation of the recommendation model, the incoming ratings continue to be added to the ratings matrix R. Their integration in the model is done simply

Algorithm 1: Cluster-based MF

```
Data: R: matrix of ratings, $K$: number of factors to consider, $\{C\}$: the list of
      clusters, $\alpha$, $\beta$, $\lambda$ and $\gamma$
Result: $P$, $Q$, $\mu$, $\{b_i\}$, $\{b_u\}$ and $\left\{b_u^C\right\}$
1  For each item $i$ and each user $u$, calculate the biases $b_i$, $b_u$ and $\left\{b_u^C\right\}$;
2  Randomly initialize the matrices $P$ and $Q$;
3  repeat
4      foreach $r_{ui} \in \mathbf{R}$ do
5          Compute $e_{ui}$;
6          for $k \leftarrow 1$ to $K$ do
7              Update $p_{uk} \in P$, $q_{ki} \in Q$;
8          end
9          Update $b_i$ and $b_u$;
10         Update also $b_u^C$, $\forall C \in \varsigma_i$;
11     end
12     Calculate the global error $\sum_{ui} e_{ui}^2$;
13 until terminal condition is met;
14 return $P$, $Q$, $\mu$, $\{b_i\}$, $\{b_u\}$ and $\left\{b_u^C\right\}$
```

by adjusting the local user biases, hence the importance of local biases. Indeed, the top-K item recommendation is constituted generally of items from different clusters (in our experimentations in [54], for three clusters, we observed that 58.47% of the users of Netflix have at least two clusters represented in their top-5, and 55.12% for MovieLens). When we adjust the local user biases with the new ratings, the recommendations can be affected in the composition of the recommended list of items or in the ranking (top-K) of these items.

Let us denote by V the set of known ratings in matrix R, including the newly added ones.

$$V = \{r_{ui} \in R/u \in U, i \in I\} \tag{3.28}$$

where U and I are the sets of referenced users and items, respectively. Then, we denote by $V(u,.)$ the set of all known ratings of a given user $u \in U$.

$$V(u,.) = \{r_{vi} \in V/v = u\} \tag{3.29}$$

The subset of ratings of user u in a cluster $C \in \varsigma_i$ to which a specific item i belongs is denoted by $V(u,C)$.

$$V(u,C) = \{r_{uj} \in V(u,.)/j \in C\} \tag{3.30}$$

The bias adjustment done when a new rating r_{ui} is obtained, requires only the ratings $V(u,C)$ of each cluster C to which i belongs. A gradient descent is performed to update the local bias of user u in the cluster C, using Equation 3.27. Algorithm 2 details the steps of the ratings integration process. As in Algorithm 1, the training process ends when the regularized global squared error does not decrease any more or when the maximum number of iterations is reached. Obviously, when the item belongs to several

Algorithm 2: Incoming ratings integration

```
  Data: $P$, $Q$, $V(u,C)$, $b_i$, $b_u$, $b_u^C$, $\alpha$, $\beta$ and $\gamma$
1 repeat
2     foreach $r_{uj} \in V(u,C)$ do
3         Compute $e_{uj}$;
4         Update $b_u^C$;
5     end
6     Calculate the locally-limited global error $\sum_{j \in C} e_{uj}^2$;
7 until terminal condition is reached;
```

clusters at the same time, this action has to be done on each of them, but it is easily parallelizable.

3.5 Complexity analysis

The cost of our cluster-based matrix factorization solution (see Algorithm 1) can be separated in two parts: the cost of matrix factorization and the cost of the clustering step. The time complexity of the training of the whole model (matrix factorization) is $O(|V| \cdot k \cdot t)$, where V denotes the set of known ratings, k is the number of factors and t the maximum number of iterations. The time complexity of the clustering step depends on the chosen clustering algorithm. When additional information on the items is available (metadata on the items), it may be used for clustering [70, 159]. Such methods can greatly reduce the clustering execution time. If no metadata is available, there are still many possible clustering techniques, only based on item ratings, each one having its own cost: projected K-means, PDDP and so on [72, 71, 128].

The strength of our technique lies in the low computation cost needed for the integration of the ratings received after the generation of the model. So that the integration can be done on the fly and the loss of quality of the recommendations slowed. The time complexity of the integration of a new rating r_{ui} is $O(|V(u,C)| \cdot t)$. Note that in the worst case this cost is equal to $O(|V(u,.)| \cdot t)$, when all the ratings of the considered user are related to the same group of items. Let us stress that $V(u,.)$ is usually small. For instance, for Netflix the average size of $V(u,.)$ is 200 [15]. The more the user ratings are distributed in different groups, the more the cost of updating the user bias is small. Still for Netflix, we have 98.4, 48.7, and 70.3 ratings in average per user with our three clusters of items.

3.6 Experimental Results

In Section 3.2 we proposed to enhance the widely used MF model, coupling it with two techniques that tend to improve the quality of predictions: the preliminary clustering of the ratings before factorization, and the final adjustment of the predicted ratings using biases. This section presents the experiments we settled, in order to validate our approach. We remind that our approach consists of generating a high quality recommendation model based on incoming ratings. Then, we use that model for recommending items, as long as possible (provided that quality remains sufficient) up to next generated model is ready, and so on. Thus, the quality of our approach depends on two factors (i) the initial quality of the generated model, and (ii) the loss of quality over time. Accordingly, we validate each factor independently, proceeding in two separated steps. Step 1 focuses on the initial quality of the model that has just been generated. Step 2 focuses on the loss of quality, of our approach, over time.

– **Step 1: Validation of the initial quality** We plan to show that our model yields good initial predictions compared to other commonly used models. We setup a fully informed environment, meaning that the model is aware of all the ratings that precede the prediction. This environment is optimal since it provides the maximal input to the model generation. Although this environment is rarely met in practice (it implies that no new ratings have occurred during the model generation), it ensures the most favorable conditions for every model. Thus it allows us comparing several models when they expose their best strength. Our objective is to quantify the quality of our model that combines factorization with clustering and bias adjustment. To this end, we compare the accuracy of our model with two commonly used models: (i) the MF alone, and (ii) the biased MF (see Section 3.4.1). Note that, we do not compare our solution with the case of MF preceded by clustering without bias adjustment, since clustering does not improve the accuracy directly in its own. Actually, clustering allows finer biases (one bias per cluster), which in turns yields better accuracy.
– **Step 2: Validation of the loss of quality over time** In the second validation step, we check that the accuracy of prediction decreases over time after each factorization. This aims to justify the relevance of our investigation to provide predictions whose accuracy lasts longer. Then, we will measure the benefits of our approach (continuous bias update, based on new ratings) for keeping up the accuracy of prediction longer than others. In other words, our solution should expose a smaller quality decrease (i.e. a flatter slope) than other solutions. In consequence, it will imply less frequent model re-regeneration, saving a lot of computation work.

3.6.1 Implementation and experimental setup

We implemented our proposition in C++ and ran our experiments on a 64-bits linux computer (Intel/Xeon x 8 threads, 2.66 Ghz, 16 GB RAM). We used a LIL matrix structure to store the dataset of ratings. To cluster the items, we ran a basic factorisation with some iterations and a K-Means algorithm on the items factors.

Table 3.1: Caracteristics of the datasets

Dataset	# of ratings	# of users	# of movies
MovieLens@100K	100K	943	1,682
MovieLens@1M	1M	6,040	3,900
MovieLens@10M	10M	71,567	10,681
Netflix	100M	480,189	17,770

We did preliminary tests to calibrate the parameters of the model and the number of clusters: $\lambda = 0.001$, $\beta = 0.02$, $\gamma = 0.05$, $N_c = 3$. The λ, β, and γ values are close to the ones suggested in [104]. We limit training to 120 iterations at most and use 40 factors for both matrices P and Q.

3.6.2 Datasets

We conduct the experiments on the Netflix dataset [15] and the ones of MovieLens [4]. These datasets are very often used by the recommendation system community [127]. These datasets report ratings that users assign to some movies. Table 3.1 shows their characteristics. The ratings are represented by integers ranging from 1 to 5 for all the datasets except the largest one of Movielens where we have real numbers. Each dataset is ordered by ascending date. With the two smallest datasets, we used the movie's genres to clusters them. Movies having several genres are in several clusters. With the two biggest datasets, we rely only on the available ratings to cluster them. Thus we consider different clustering approaches in our experimentations. Let us remark that any item's property is not available in the Netflix dataset.

3.6.3 Initial quality

The objective of this experiment is to compare the initial qualities of the three models. We split the two biggest datasets into two parts : a training set representing 98% of the set of ratings and a test set which keeps the rest (the 2% most recent ratings to predict). As comparison, the test set from the Netflix dataset contains 1.88M ratings. This number of ratings is greater than the one of the Netflix Prize which has 1.4M ratings [15]. We did the same for the others but with a training set of 80%. As the latter are small, this percentage allows us to have enough data in the training set.

Table 3.2 reports the different RMSE errors obtained for the three models named Basic MF, Biased MF, and CBMF. We remark that CBMF outperforms the other models. It reaches 1.12% of improvement over the biased MF with the Netflix dataset. Let us remind that even an improvement as small as 1% of the accuracy leads to a significant difference in the ranking of the top-K most recommended items for a user [74, 31]. As

4. `http://www.grouplens.org/node/73`

Table 3.2: Initial quality of the three models in terms of RMSE score

Dataset	Basic MF	Biased MF	CBMF
MovieLens@100K	0.9370	0.9361	**0.9289**
MovieLens@1M	0.9129	0.9102	**0.9024**
MovieLens@10M	0.7743	0.7608	**0.7578**
Netflix	0.9599	0.9312	**0.9208**

Table 3.3: Percentage of quality improvement

Dataset	Basic MF	Biased MF	CBMF
Movielens	2.56	5.15	**6.09**
Netflix	4.46	4.39	**5.67**

we target large scale datasets, from here we consider only the Netflix dataset and the biggest one of Movielens in our experimentations.

3.6.4 Large training sets improve the quality of the model

The objective of this experiment is to measure the quality of the model according to the size of the training set. We check the intuitive rule stating that the more ratings we take as input, the best quality we get.

To realize this experiment, we first sort the ratings of each user according to their timestamps. Then, we split the training set (98% of the initial dataset) into 10 chunks (c_1 to c_{10}) of equal size: 10% each. Thus, the number of ratings of a user is almost the same in each chunk. From that, we generate 10 training sets (T_1 to T_{10}) of increasing size by assembling the chunks such that we always use the most recent ratings to generate the model. More precisely, $T_1 = \{c_{10}\}$, $T_2 = \{c_9\} \bigcup \{c_{10}\}$, $T_3 = \bigcup_{i \in [8-10]} \{c_i\}$, ... $T_{10} = \bigcup_{i \in [1-10]} \{c_i\}$. (cf. Figure 3.3).

Figure 3.4 reports the RMSE evolution of the three models, for the two datasets: MovieLens (3.4a), and Netflix (3.4b). We see that the three models are affected by the size of the training set. The more ratings they have, the better quality they tend to propose. Table 3.3 shows the quality improvements of these three models from T_1 to T_{10}. The CBMF model shows 5.7% and 6% of quality improvements respectively for Netflix and MovieLens, thanks to the finer-grained cluster-based bias adjustment. This confirms the ability of local biases to better capture user tendencies over large training sets.

We observe on Figure 3.4 that on the range 10%-60% (training sets T_1 to T_6), the Biased MF model outperforms the CBMF model. Indeed, with the first training sets we do not have a lot of data to compute enough discriminative clusters. Also the fact that the users do not have yet rated a lot of items harms the local biases adjustment.

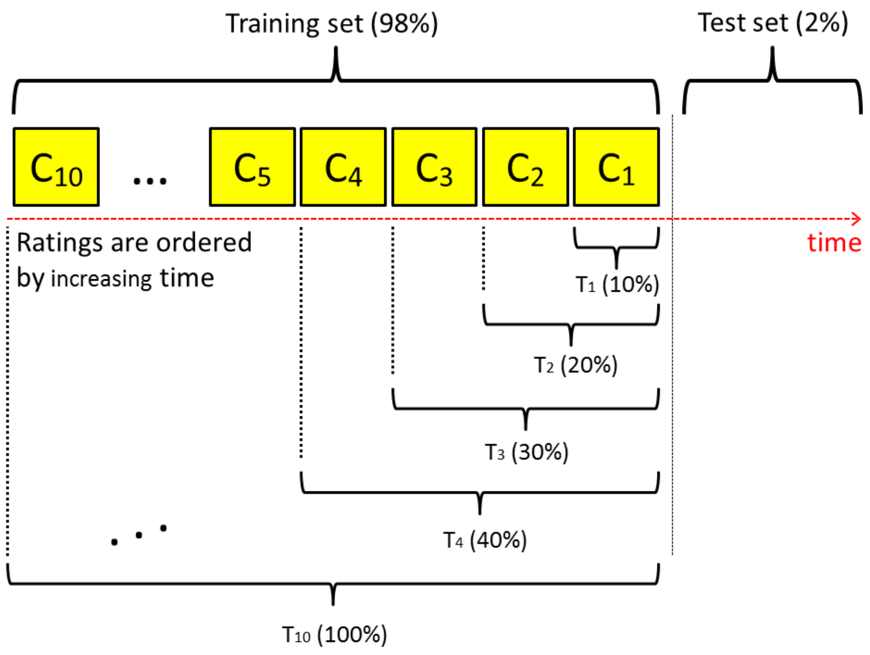
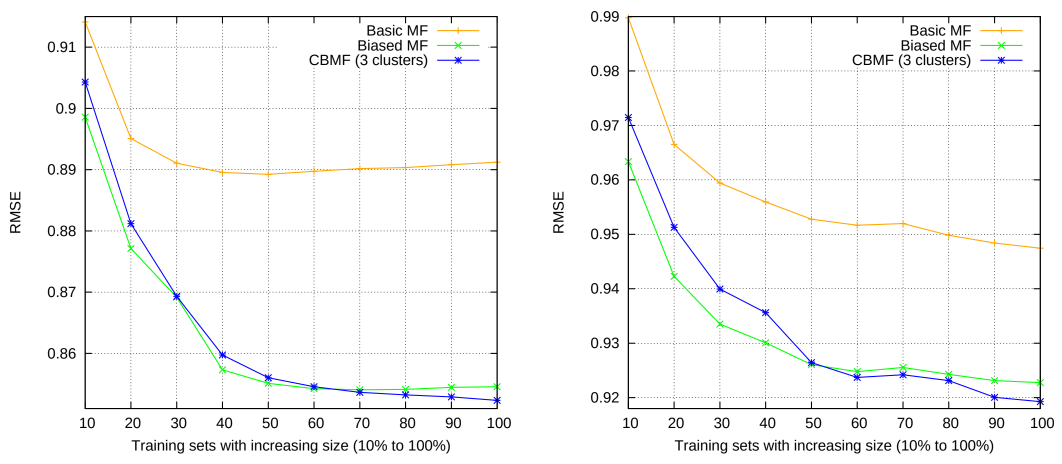


Figure 3.3: Training sets partitioning



(a) MovieLens (b) Netflix

Figure 3.4: Quality improvement for increasing training sets sizes

We also see different RMSE error ranges between the datasets. This difference between the RMSE errors is due to the data characteristics. For instance, the 10M MovieLens dataset has decimal ratings while the Netflix dataset uses only integer values. Adomavicius and Zhang mention this phenomenon in [4]. They point out consistent and significant effects of several data characteristics on recommendation accuracy. Finally, we note the importance of the biases. The basic MF suffers from that, it never catches up the other models whatever the dataset.

3.6.5 Quantifying the need for online integration

Basically, we need online integration when offline solutions fail to provide sufficient quality. The objective here is to measure the impact of missing ratings on the quality that offline models can deliver. We wonder to what extent the most up-to-date ratings influence the recommendation. Given a training set containing a fixed amount of ratings, we investigate the quality variation when the ratings become less and less recent. Moreover, we target the 'input intensive' scenarios where a lot of new ratings are produced in a short period of time, thus million ratings are potentially missing. For instance Netflix company receives 4 million ratings per day [8]. To reflect this, we must consider several millions of missing ratings in our experimentations. Therefore, we experiment only with the Netflix dataset which is the largest one, the MovieLens dataset does not have enough ratings to setup an enough number of missing ratings. Indeed 10M Movielens dataset does not match the experimental requirements, because we risk to reduce drastically the training set size, which becomes too small to produce meaningful results (i.e., few items are rated in both the test set and the training set).

We define the test set and the training set as follows. We keep in the test set 10% of the ratings, the most recent ones. The training set contains the 90% remaining ratings. To better observe the impact of the delay on the RMSE, we balance the delay of each user. More precisely, we order the ratings by arrival position, such that the i^{th} ratings of any user precede the $i+1^{th}$ ratings of any of them, and so on. We measure the evolution of the prediction quality along the ordered test set by computing the RMSEs over a sliding window of 200K ratings as size. Thus two consecutive windows share the half of their ratings (for smoother results).

Figure 3.5 shows the evolutions of the prediction quality for the three models : Basic MF, Biased MF, and CBMF. Figure 3.5 shows that the error is increasing with the number of missing ratings. We observe a 5% RMSE increase for long delays (from 5M to 7M missing ratings). Such quality loss might not be acceptable for recommendation systems. This confirms the need for online integration.

3.6.6 Robustness over time of our online integration model

The goal is to show that our model is robust over time, i.e., it still yields good quality predictions even when many ratings have been produced since the last factorization. Using the same training and test sets, we now take into account the missing ratings to adjust on the fly the local users' biases (cf. Algorithm 2). More precisely, we sequentially scan the test set and consider the ratings one by one. For each rating, we calculate the prediction error, then we immediately integrate the rating in order to improve the next predictions. The average time to integrate one rating is 0.4 millisecond. It is fast and adds few overhead on the online recommendation task.

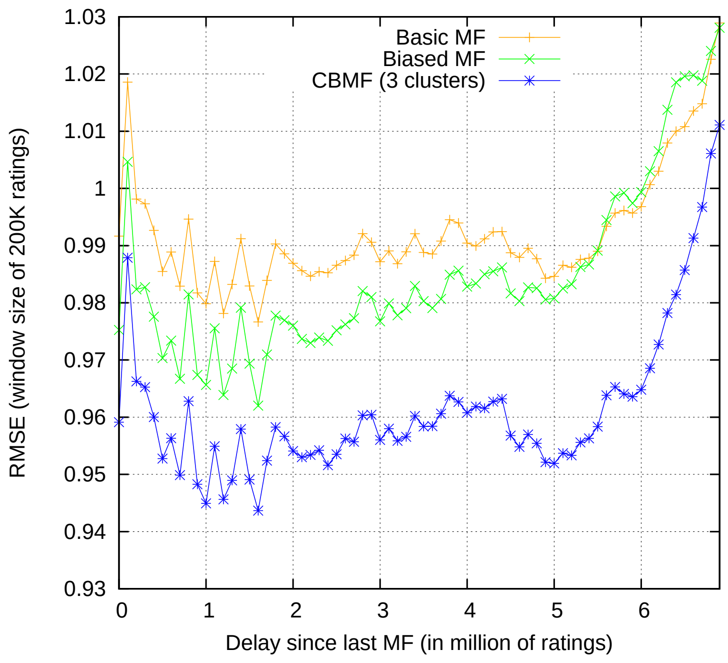


Figure 3.5: Offline quality (RMSE) with increasing delays (# million of ratings)

In Figure 3.6, we report the new evolution of CBMF prediction quality when we integrate the incoming ratings taken from the test set. We first analyze the CBMF errors in Figure 3.6, and compare it with the static (offline) case, to figure out the importance of online integration. The benefit of online integration is up to 13.97% for the largest delay (close to 7M missing ratings), which is a significant improvement for recommendation purpose. This makes our solution quite robust.

3.6.7 Quality vs. Performance tradeoff for online integration

We conducted further experimentations to validate our choice about what part of the model is worth being updated during the online integration phase. We investigated three possible methods to integrate a new rating: (i) update the user factors only, (ii) update the user local biases only, and (iii) update both the user factors and local biases. Naturally, processing more updates comes at a cost. We wondered if the computation time spent in more integration would be eventually amortized by the benefit of postponing the next model re-computation. Figure 3.7 shows the quality improvements brought by these three methods of integration. We reported, on Table 3.4 the update time and the respective mean quality gain (in terms of RMSE) for each of the three above mentioned integration methods. We deduced that integrating both the local biases and the factors bring a relative benefit of 7% compared to integrating the local biases only. On the other hand, it adds up to 151% relative overhead on the computation cost. Given

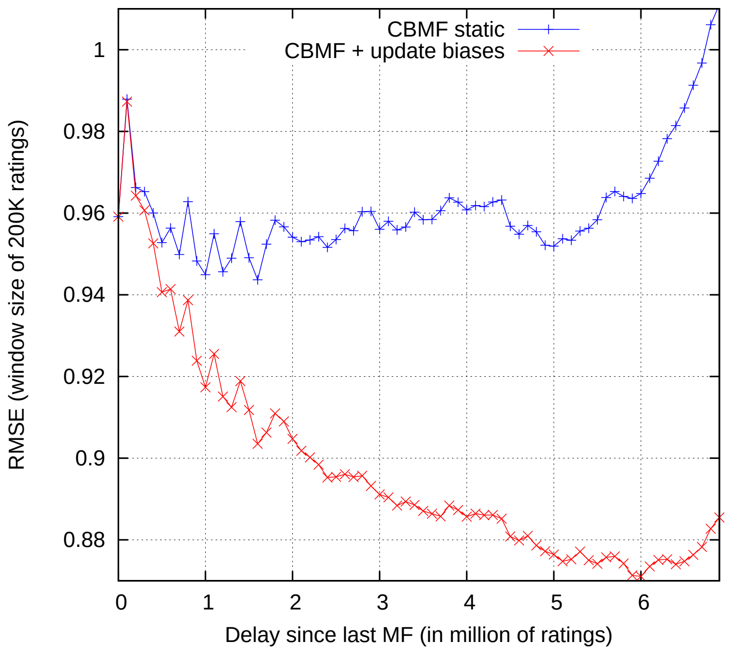


Figure 3.6: Quality of online integration for increasing delay

Table 3.4: Quality vs. Performance tradeoff

Update	Improvement (%)	Average update time
user factors	0.84	3.11 ms
local biases	7.18	1.24 ms
both	7.69	3.75 ms

a tolerated RMSE value, and the absolute values of the matrix factorization cost and the integration cost, we were able to decide which method yields the minimum overall cost. Table 3.4 shows that the local biases-only update method provided the optimal performance (best balance between quality improvement and update cost).

3.6.8 Benefit of refactorization

The objective of this experiment is to quantify the benefit of recomputing the CBMF model. Intuitively, one wish to recompute the model when its quality moves away beyond the expected quality level. On the other hand, in order to save computation resources, we wish to recompute the model id needed only. With this in mind, we set up an experiment which consists of five successive factorizations. We begin with the same test set and training set as in the previous experiment: the 10% most recent ratings are in

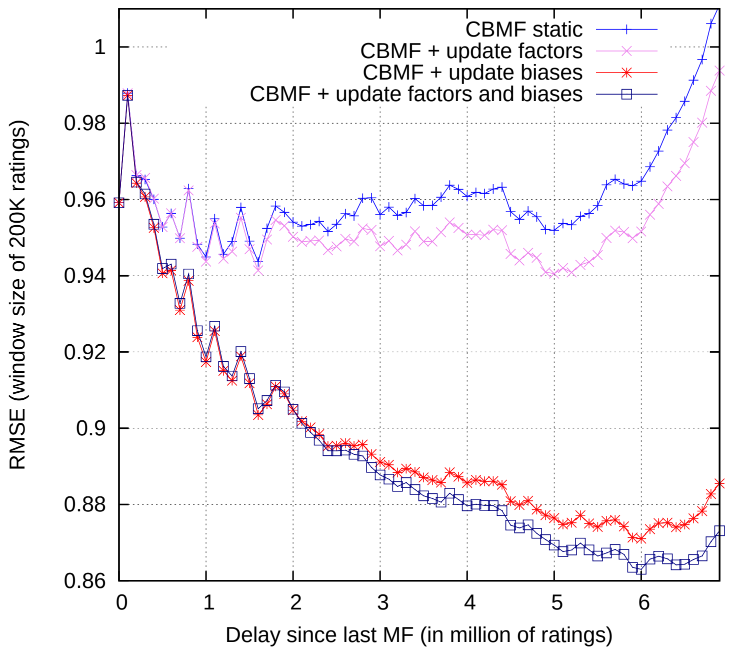


Figure 3.7: Quality vs. Performance tradeoff

the test set, the remaining 90% are in the training set. We generate five models resulting from five successive factorizations, scattered in time as described in the following. Let M_0 denote the initial model resulting from the training set factorization. Then, we sequentially scan the test set, integrating the incoming ratings into M_0, on the fly, until we reach 20% of the test set. At this point, we trigger the re-factorization and generate a new model, denoted M_1, which replaces M_0 to become the current model. Then, we repeat the sequence "scan next 20%, refactorize and replace model" until we reach the end of the test set. We end up generating M_2, M_3, and M_4 which integrate respectively 40 %, 60%, and 80% of the test set in addition to the initial training set. We report the resulting RMSE, on Figure 3.8, while iterating over the test set and using the most current model, namely M_0 to M_4, for prediction. We compute each RMSE value based on all the ratings that occur between the current factorization and the next one. We first globally observe that re-factorization outperforms *CBMF online* at any point in time. Indeed, whatever is the amount of information, a globally optimized model (i.e., factorization) is more accurate than a locally adjusted model (i.e., bias update). Second, we measure that re-factorization slightly improves *CBMF online* up to 1% for M_4. This is mainly because CBMF online performs quite well all along the run. Hopefully, this offers enough time to recompute the model. In our case, M_1 took 8 hours to compute which is the equivalent time to receive 1.33 million ratings (according to the Netflix rate [8]). We observe that CBMF online yields low RMSE during a longer time than the time required for re-factorization. This make our solution practical. Furthermore, a

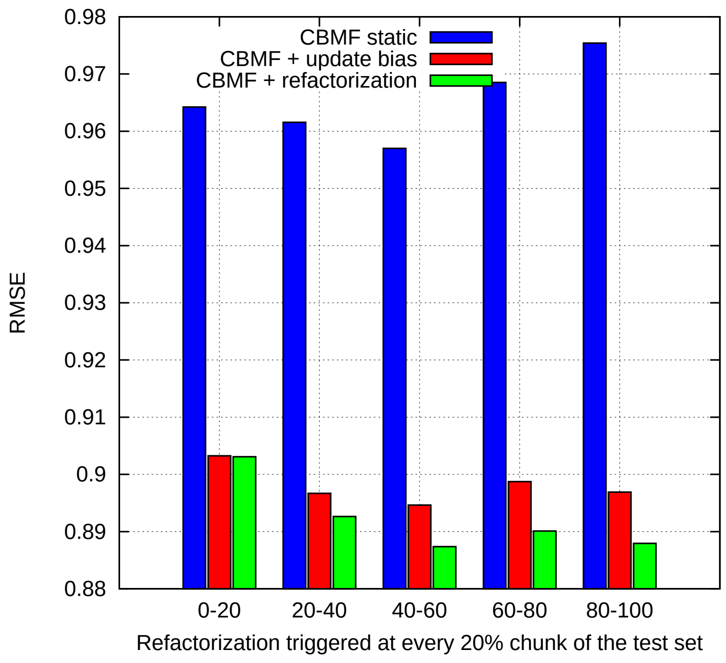


Figure 3.8: Refactorization benefit

longer run, could serve to measure the maximum “validity time” of the CBMF online. In turn, this would allow to deduce the optimal date to trigger the refactorization, while keeping the RMSE bounded.

3.7 Conclusion

In this chapter, we tackled the collaborative filtering problem of accurately recommending items to users, when incoming ratings are continuously produced and when the only available information is several millions of user/item ratings. Through years of experimentation campaigns, the recommendation systems community has demonstrated that the model-based solutions achieve the best quality, however such solutions suffer from a major drawback: they remain static. They take as input a snapshot of the ratings at the time the model computation starts. They simply ignore the more recent ratings, skipping possibly meaningful information for better recommendation.

Our goal was then to find a way to enable the integration of the incoming ratings for a well-know model-based recommendation solution requiring heavy computation with billions of ratings [8]. To this end, we refined the matrix-factorization model that features very good offline quality, by introducing personalized biases that capture the user subjectivity for different groups of items. Items are grouped basing on their ratings.

We proposed a detailed algorithm to update the fine grained (i.e. per item cluster) user biases, which is fast enough to integrate the incoming ratings as soon as they are produced. We implemented the algorithms and performed extensive experiments on two real large datasets containing respectively 10M and 100M ratings, in order to validate both quality and performance of our cluster-based matrix factorization (CBMF) approach. We compared our solution with two state-of-the-art matrix factorization solutions that support 0 and 1 bias respectively. Qualitative results place our solution better to its competitors in the offline case. Our solution demonstrates an improvement of accuracy up to 13.97% (relatively to the offline case) for highly dynamic scenario where millions of incoming ratings are injected into the model. Moreover, performance results expose fast integration of the incoming ratings; which makes our solution viable for online recommendation systems that need to scale up to a higher throughput of incoming ratings.

4 | Making Social and Popularity-based Tag Recommendations

The current Web is full of social media sites which focus on sharing various item types, e.g. pictures (Flickr), or URLs (del.icio.us). These sites massively use metadata (i.e., tags) to enrich their item description and thereby provide better user experiences through applications like item search that relies on tag-based similarity between items and/or users.

For tagging, media sites can rely on expert groups but they fail to face a large number of items. There are also automatic annotation methods based on a predefinied set of classified tags but they lack of evolutivity. Moreover these approaches do not capture the user perception of the items and tags [7].

Social tagging is the practice of allowing users to freely annotate shared content. The users can organize and search content based on annotations, more commonly called tags. While allowing users to apply tags for personal aims [1], social tagging applications give them the possibility to rely on the classification applied by others to browse and get interesting items. However, due to the flexibility of social tagging systems the classification process is not rigorous, hence the need of recommending tags to the users in order to improve the homogeneity of the tags and ease the access to classified items.

Tag recommendation aims to recommend the most suited tags a user can assign to a given item. The popularity of social media sites has made it an active and growing topic of research [69, 97, 61].

In this chapter, we target online social applications dealing with large networks, composed of tens of thousands of users and more. The objective is to provide these applications with a very fast and scalable solution to compute recommendation. In this context, we investigate methods that match both the quality and scalability requirements, and tackle the challenge of delivering tag recommendations which yield good quality in a short time.

1. For example, a user may use some tags to summarize the content of an item like in *Bibsonomy*, when another uses them to locate a picture like in *Flick* or *Instagram.* [27], Dattolo et al. present a useful survey about the role of tags for recommendation

One key aspect of tag recommendation systems is the scoring function that is used to measure the relevance of the recommended tags. In [50] we proposed a scoring function that combines a global tag relevance (i.e., for all the users) with a local relevance limited to the neighborhood of the requesting user. We present here a wisely chosen tag scoring function which strives to improve the accuracy of recommendations. It combines the reputation of the tags with the opinion of the requester's neighborhood. We assume that the users are organized in a weighted graph representing, for instance, the similarity or trust between them. Thus we introduce a 3-components scoring function that captures the tag relevance locally (for the requesting user), socially (for the user neighborhood) and globally (for all the users). The recommended tags are computed on-the-fly using an optimal navigation technique inspired from previous works on aggregation and search algorithms [33, 89, 88]. We exploit the transitivity of the users' similarity to enlarge the neighborhood circle, and thus enhance the quality of the recommendations while limiting the traversal of the graph to a small number of neighbors.
We present also a method to control the number of tags to recommend such that it provides the requesting user with a better relevance. We performed extensive quality and performance experiments, and compared our technique with the state-of-the-art solutions. The obtained results show a significant quality improvement.

For a better understanding of the sequel, we first begin with some preliminaries before discussing related works in Section 4.3. Sections 4.2 details our proposal. In Section 4.4, we present the experiments we have done, and conclude this work in Section 4.5.

4.1 Preliminaries

Social resource sharing systems are central elements of the Web 2.0. They use the same kind of lightweight knowledge representation, called folksonomy [16].
A folksonomy S is a system of classification that allows its users to create and manage tags to annotate and categorize content. It is related to social bookmarking and can be defined formally as a set of users U, a set of tags T, a set of items I, and a ternary relation between them $S \subseteq U \times I \times T$. Each triple $(u, i, t) \in S$ indicates the tagging of an item i, by a user u, using a tag t. We denote by a *post*, $T(u, i)$, the list of tags assigned by a user u to an item i. We assume that a user can tag an item with a given tag at most once.

In order to simplify the rest of this chapter, let us consider the following definitions: For a given user u and a given item i, we denote by $score(t|u, i)$ the estimated relevance of tag t. The K highest scores are obtained as follows:

$$Top(u, i, K) = \underset{t \in T}{\overset{K}{argmax}}\ score(t|u, i) \tag{4.1}$$

Beside this, we consider that the users are organized in a weighted graph $G = (U, E, \theta)$, where U represents the set of vertices, E the set of edges and θ is a function that associates to each edge $e = (u, v) \in E$ a value $\theta(u, v) \in [0, 1]$ corresponding to a proximity (or similarity) measure between the users u and v.

Table 4.1: Definitions from folksonomy

$T(u)$	$\equiv$	$\{t \in T \mid \exists i \in I : (u, i, t) \in S\}$
$T(i)$	$\equiv$	$\{t \in T \mid \exists u \in U : (u, i, t) \in S\}$
$T(u, i)$	$\equiv$	$\{t \in T \mid (u, i, t) \in S\}$
$I(u)$	$\equiv$	$\{i \in I \mid \exists t \in T : (u, i, t) \in S\}$
$I(u, t)$	$\equiv$	$\{i \in I \mid (u, i, t) \in S\}$
$U(i)$	$\equiv$	$\{u \in U \mid \exists t \in T : (u, i, t) \in S\}$
$U(i, t)$	$\equiv$	$\{u \in U \mid (u, i, t) \in S\}$

4.1.1 Similarity propagation

As stated above, the weighted graph G represents the observed similarity between the users, and can be inferred from the tagging behavior of the users. However, the lack of direct link between two users, does not necessarily mean that they can not be similar to some extent. For instance, two users u_1 and u_2 may have different areas of interest and yet share a common neighbor v who may be, for some aspects, similar to u_1 and u_2. Many works addressed the problem of social link propagation, mainly in the case of trust networks [130, 32]. In this work, we consider transitivity as a key element for similarity propagation, and we compute it by multiplying the similarity values observed at each step.

Figure 4.1 shows an example where the propagation of similarity may lead to better recommendations. Here, we suppose that *Arthur* wants to tag item i_1 and the system is expected to propose to him some relevant tags. By $\rho(t_n, Arthur)$ we refer to the popularity of tag t_n, as seen by *Arthur* (i.e., the number of times *Arthur* uses this tag). We assume that in the direct neighbors of *Arthur* only *Celine* has tagged i_1. Thus, he/she is the only user, among the similar neighbors of *Arthur*, who can give us an idea about what tags to recommend (i.e., t_1 and t_2). However, using the similarity propagation, one can easily see that the opinions of *Helen* and *Frank*, who already tagged i_1, may be more relevant than the one of *Celine*. Indeed, the inferred similarity values are 0.54 (0.9×0.6) between *Arthur* and *Frank*, and 0.504 ($0.9 \times 0.8 \times 0.7$) between *Arthur* and *Helen*.

Therefore, the similarity function θ defined above, can be extended to deal with propagation, following a natural interpretation that, as trust, similarity is transitive at least to some extent [130, 158]. This allows to yield an overall tag scoring scheme that depends on the entire network instead of only the user's vicinity. We infer the similarity value between two users u_1 and u_n, connected through a path $p = (u_1, ..., u_n)$, by multiplying the similarity values observed at each step [158, 130]. In the following, we denote this extended similarity by θ^+.

$$\theta^+(p) = \prod_{i=1}^{n-1} \theta(u_i, u_{i+1}) \tag{4.2}$$

The inferred similarity decreases gradually at each new step of a path. Thus, given a social network G and a path $p = (u_1, ..., u_n) \in G$, we always have $\theta^+(u_1, ..., u_{n-1}) \geq$

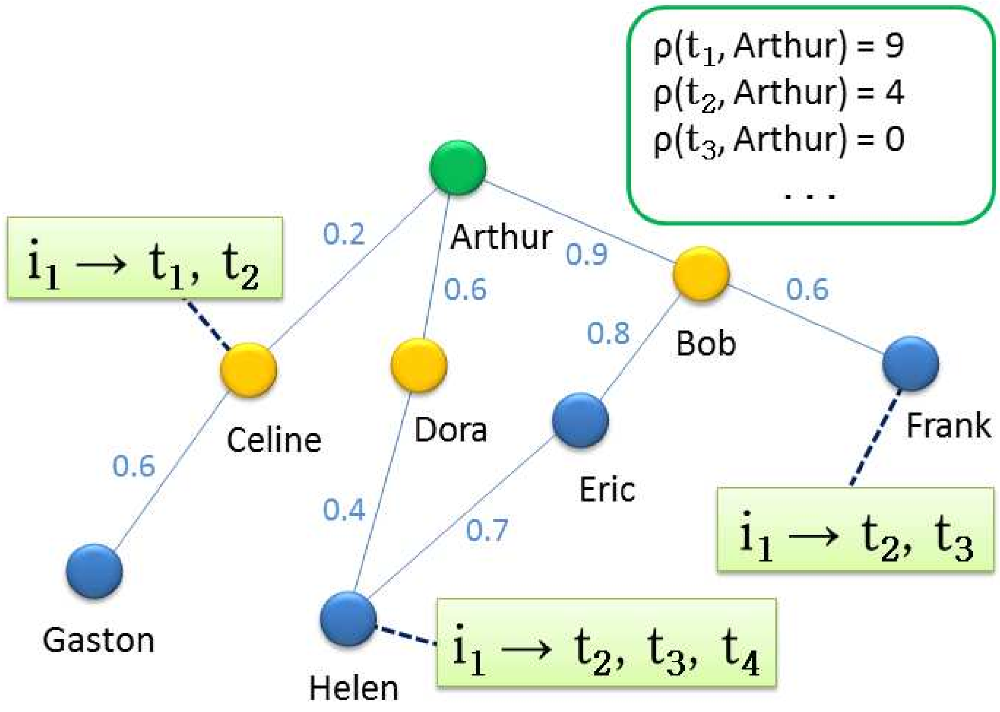


Figure 4.1: Social link propagation leads to better decisions

$\theta^+(u_1, ..., u_n)$. When two users u and v are connected through different paths in the network, the extended similarity between u and v corresponds to the highest similarity value computed from all the paths. More precisely, we define the extended similarity, $\theta^+(u, v)$, as follows:

$$\theta^+(u, v) = \max_{p \in G} \left\{ \theta^+(p) | u \xrightarrow{p} v \right\} \tag{4.3}$$

For instance, consider the example of Figure 4.1. *Helen* and *Arthur* are connected by two paths, and their extended similarity is evaluated to 0.5 ($0.9 \times 0.8 \times 0.7$).

4.1.2 Extended neighborhood opinion

We denote by $\eta(t|u, i)$ the opinion of the neighbors of user u about the relevance of tag t for the annotation of item i, and define it as follows:

$$\eta(t|u, i) = \frac{1}{|U(i)|} \times \sum_{v \in U(i,t)} \theta^+(u, v) \tag{4.4}$$

We sum the similarity values between u and his/her neighbors who already annotated the item i by t. Then, we normalize this sum by dividing it with the number of users who tagged this item ($\eta(t|u, i) \in [0, 1]$). For instance, considering the example of Figure 4.1, $\eta(t_3|Arthur, i_1) = 0.52$ and $\eta(t_2|Arthur, i_1) = 0.41$. Our normalization aims to emphasize two aspects of the tags: their popularity (i.e., frequency) and the similarity with the users who used them.

In order to compute the opinion of the neighborhood of a user, we need to explore the weighted graph starting from the considered user. A naive approach may lead to a prohibitive navigation cost, especially for real world setting. Thus, we use an optimal navigation technique based on previous works on optimal aggregation and search algorithms [37, 33, 89, 50], as we will shortly explain.

4.2 Social and Popularity-based Tag Recommendation

We present in this section our tag recommender system, called FasTag, which enables recommending the most relevant (top-k) tags to be associated with an item. FasTag combines the popularity of the tags with the opinions of the neighbors, while recommending a tag to a user. The right balance between these elements allows a good quality of recommendations, as we show in the experiments. The number K of tags to propose to the user is adjusted by the system in order to maintain a good quality of recommendation.

We present below the score model used by FasTag, in Section 4.2.1. We detail our algorithm in Section 4.2.2.

4.2.1 Score model and tag relevance

Intuitively, when a tag t is popular (i.e., frequently used), it may be relevant to recommend it to the users. That is why tag recommendation models try to take into account the popularity of the tags. However, different users may have different preferences while tagging the same item. Thus, obtaining reliable opinions about the tags from the closest neighbors may be of great help. Let us consider again the example shown in Figure 4.1. The tag t_1 is more popular than t_2 (t_1 is used nine times while t_2 is used only four times). If we consider the popularity of the tags, we will first recommend the tag t_1 to *Arthur*, despite the fact that this tag may be not used to tag the considered item (i_1). On the other hand, if we consider the opinions of *Arthur*'s neighborhood as formulated in Equation 4.4, the tag t_2 appears more relevant to recommend ($\eta(t_2|Arthur, i_1) = 0.41 > \eta(t_3|Arthur, i_1) > \eta(t_1|Arthur, i_1)$). However, *Arthur* has never used the tag t_3. A good balance between popularity and neighborhood opinions seems to be more appropriate.

For a given user u and a given item i, FasTag computes a score for each tag t as follows:

$$score(t|u,i) = \Big(\alpha \times \rho(t,i) + (1-\alpha) \times \rho(t,u)\Big) \times \Big(1 + \eta(t|u,i)\Big) \tag{4.5}$$

Here, $\rho(t,i)$ (resp. $\rho(t,u)$) denotes the popularity of the tag t for the item i (resp. for the user u) and $\eta(t|u,i)$ is a measure of the opinion of the user's neighbors, as already defined in our preliminaries. Thus, our scoring model combines the popularity of the tag with the opinions of the users' neighborhood about the use of this tag to annotate the considered item. The more a neighbor is close (similar) to the user, the more his/her opinion is important in our scoring model. The sum $1 + \eta(t|u,i)$ allows to take into account isolated users. More precisely, we compute the values of $\rho(t, i)$ and $\rho(t, u)$ as follows:

$$\rho(t, i) = \frac{|U(i, t)|}{|U(i)|}, \qquad \rho(t, u) = \frac{|I(u, t)|}{|I(u)|}$$

The parameter α in Equation 4.5 allows to tune the relative importance of each component of the sum (item-related tag popularity versus user-related tag popularity).

4.2.2 The FasTag Algorithm

Considering social link between users is an additional contribution in top-k retrieval process but it may have computing costs in a real-world setting [89, 10]. To face this issue we can apply four strategies:

- $FasTag_1$: We do not consider similarity propagation on the network and then we restrict the user's neighborhood to his/her direct neighbors only.
- $FasTag$: Of course, the above strategy lacks efficiency when the indirectly connected users are relevant to the recommendation. Obviously, the closest neighbors should contribute more than the others. The extended similarity we introduced in our preliminaries reflects this vision. Moreover, while enabling similarity propagation we can use some algorithms of approximation in order to estimate as soon as possible that the current top-k tags will not change anymore [33]. Therefore, we are able to stop the recommendation process earlier without investigating the whole network (see below for details about this approximation).
- $FasTag_{++}$: It extends the previous strategy. It consists in permanently adding new edges in the network, i.e, making some shortcuts between indirectly connected users. We add shortcuts on the fly during the recommendation process [50, 89]. Each traversal may add new edges which are useful to the next recommendations as it saves the cost to recompute the extended similarities.
- $FasTag_{\infty}$: We also consider the baseline strategy which investigates the whole social network for retrieving the exact score of each tag. This strategy still adds new edges as $FasTag_{++}$ but does not optimize the traversal. We use it, for comparison purpose, to measure the impact of the early stop strategy.

We now detail how FasTag saves computation time (early stop strategy) while taking into account the opinion of neighbors whatever they are directly or indirectly connected. The idea is to stop the computation as soon as the current top-k tags to recommend will no more change, even if we have not yet got their exact scores.

At any step of the traversal, we estimate the maximal score a tag may reach in order to know if it may enter into the top-k list. We assume that we have an inverted list IL_i of the popularities of the tags used to tag the item i, $(\rho(t, i))$, sorted in descending order. Thus, starting from the top most tag, we explore the list one tag at a time. The consumed tag becomes candidate for the top-k result. We consider as unknown the popularity of a tag which is not yet used. We denote by $top^{tag}(i)$ the tag currently at the top of IL_i and $top^{\rho}(i)$ as its popularity.

Let us remind that we maintain a max-priority queue H whose top element $top(H)$ will be at any time the most relevant yet unvisited neighbor on the network. At each step of the network traversal, we extract (visit) the top of the queue, and we add its unvisited neighbors to the queue. Moreover if the aggregated similarity of the path between the neighbor and the user is greater than the already known similarity between them, we update the similarity value according to Equation 4.3.

In the sequel, for the sake of clarity, we introduce some definitions.

Definition 1. *We define the minimal possible score of a tag t by a user u for an item i, $MinScore(t|u,i)$, as its pessimistic overall score by only setting the item's share to*

$$\begin{aligned} MinScore(t|u,i) &= \Big(\alpha \times \max\big(\rho(t,i), partial\_\rho(t)\big) \\ &+ (1-\alpha) \times \rho(t,u)\Big) \times \Big(1 + \eta(t|u,i)\Big) \end{aligned} \tag{4.6}$$

where $partial\_\rho(t)$ represents the current percent of visited users who tagged i with t. It is a lower-bound value of $\rho(t,i)$, when it is not yet known.

Hence, let D denote the list of all the candidate tags, sorted in descending order of their minimal possible scores. The first K tags of D are the current top-k elements to recommend.

Let also $unseen\_users(t,i)$ denote the maximum number of yet unvisited users who may have tagged the item i with t. During the neighborhood exploration, each time we visit a user v who tagged item i with t, we *(i)* update $\eta(t|u,i)$ (initially set to 0) by adding to it $\theta^+(u,v)$, then *(ii)* we decrement $unseen\_users(t,i)$.
We obtain the final neighborhood opinion, $\eta(t|u,i)$, when $unseen\_users(t,i)$ reaches 0. For more details about this process see [50, 89].

Definition 2. *Symmetrically to the minimal possible score, we define the maximal possible score of a tag t, $MaxScore(t|u,i)$, that has already been seen in D (an optimistic overall score) by*

$$\begin{aligned} MaxScore(t|u,i) &= \Big(\alpha \times \rho(t,i) + (1-\alpha) \times \rho(t,u)\Big) \times \Big(1 + \\ &\eta(t|u,i) + \tfrac{\theta^+(u,top(H)) \times unseen\_users(t,i)}{|U(i)|}\Big) \end{aligned} \tag{4.7}$$

The maximal possible score of a tag is an estimation of the greatest score it may reach if all the yet unvisited neighbors have annotated item i with it.
From these two definitions, we can introduce the former condition of our computation termination. It occurs when the maximal optimistic scores of tags already in D, but not in the top-k (i.e., the K first tags of D), are less than the minimal score of the last element in the current top-k (i.e., $D[k]$).

Definition 3. *We estimate also an upper-bound score, $MaxScoreUnseen$, on the yet unseen tags using as overall score for all the unseen tags as follows:*

$$\begin{aligned} MaxScoreUnseen &= \Big(\alpha \times top^{\rho}(i) + (1-\alpha) \times top^{\rho}(u)\Big) \\ &\times \Big(1 + \frac{\theta^+(u,top(H)) \times U^i}{|U(i)|}\Big) \end{aligned} \tag{4.8}$$

As for $top^{\rho}(i)$, $top^{\rho}(u)$ is the highest tag popularity for user u. We maintain a sorted list of these popularities in order to get $top^{\rho}(u)$ at any time. The first time a tag is met during the network traversal, we drop it from this list. Therefore it does not count for unseen tags.
U^i represents the current number of unvisited users who tagged the item i. At the start, we initialize it to $|U(i)|$ and then we decrement it each time we meet an unvisited user who tagged the item.
This upper-bound score allows us to determine if a yet unseen tag might be in the top-k. Thus it introduces the second condition of termination of FasTag that is to say no yet unseen tag can change the current top-k, $MaxScoreUnseen < MinScore(D[k]|u, i)$.

As already mentioned we keep and maintain the top candidate tags in the list D sorted in descending order by their minimal possible scores. Thus at any moment we can retrieve the top-k tags to recommend. This list D is updated each time we meet a tag which is not already in. We add this new tag as candidate. When the two conditions of termination presented above occur the execution terminates.

Algorithm 3 presents the functioning of FasTag as we introduced it in the above subsections.

Algorithm 3: FasTag algorithm

```
Input: $(u, i) \in U \times I$
Output: $D[1], \dots, D[k]$
1  /* Initialization */
2  foreach $t \in T$ do
3      Compute $\rho(t, u)$ and $\rho(t, i)$
4      $\eta(t|u, i) \leftarrow 0$
5  end
6  $D \leftarrow \emptyset$ /* The sorted list of candidate tags */
7  $H \leftarrow$ max-priority queue of the neighbors of $u$ sorted by their assigned similarity
   values
8  Compute $IL_i$ and $IL_u$
9  /* H is sorted by $\theta^{+}(u, v)$) */
10 while $H \neq \emptyset$ do
11     $v \leftarrow EXTRACT\_MAX(H)$ /* Extract $top(H)$ */
12     /* Then extend the opinion process to the neighbors of $v$ */
13     $NEIGHBORHOOD\_OPINIONS\_PROCESS(u, v, i)$
14     if Conditions of termination occur then
15         break /* Early stopping */
16     end
17 end
18 Return $D[1], \dots D[k]$
```

4.2.3 Handling the Network Partitioning

To further optimize the network traversal, we try to find the unreachable users who tagged an item. This case may happen when we have a partitioned network. We improve our stop condition at Line 10 of Algorithm 3 to reduce the computation time. We proceed in two steps:

1. detecting the network partitions: when our computation leaves the *WHILE* instruction at Line 10 of Algorithm 3 and that is due to $H = \emptyset$, we evaluate the number of visited neighbors. If the number is lower than the size of the network and the user u not yet assigned to an already discovered network partition, we report a new partition into which we put u and all the visited neighbors during the current recommendation task.
2. using the network partitions: if the user u belongs to a network partition, we count the number of users in this partition who already tagged the item i. This number may be lower than the one on the whole network. For each iteration of our *WHILE* instruction, we decrement the value of this number. When it reaches zero, we end the iterations and return the first tags of D.

The step 1 (detection) occurs only once per partition, during the first recommendation related to a partition. The step 2 (saving computation) improves greatly the next recommendations.

4.3 Related Work

Finally, in [50] we proposed STRec, a social-based tag recommender which optimizes the social network traversal. In this first work, we did not consider the user-related tag popularity (i.e., $\rho(t, u)$) in the scoring function. Thus the recommendations we made are not so personalized despite we used the users' neighborhood. FasTag remedies this and gives better results. Moreover the network partitions were not handled by STRec in order to further optimize its network traversal (see Section 4.2.3). FasTag includes in its optimization the handling of network partitioning. There are two important points which distinguish it from STRec. FasTag improves both the recommendation quality of STRec and its computation time in case of network partitioning.

There are a lot of investigations on graph-based tag recommenders which rely on links between users (e.g., social relation, tagging behavior),items and/or tags [68, 77, 40, 98]. One of the first developed is FolkRank [68]. It relies on a user-item-tag tripartite hypergraph. FolkRank gives good recommendations but has scalability problems. In [150], Zhang et al. combine FolkRank with a collaborative filtering technique which considers, like us, the similarities between users. As a FolkRank-based solution, their approach lacks scalability. Kubatz et al. introduced in 2011 a tag recommender algorithm called LocalRank which works as FolkRank but can make real-time recommendation [77]. LocalRank focuses on the local neighborhood of a given user and resource. It considers only a small part of the tripartite hypergraph and can thus generate tag recommendations in real-time. However their recommendation quality is nearly the same as the one of FolkRank and then is lower compared to FasTag (see Table 4.3).

In [98] and [40], the authors combine some simple recommenders (e.g., popular tags used to annotate an item, the ones by a user and those of his neighborhood). In the experimentation in [40], Gemmell et al. showed that their proposition gives better results. However they extracted some post-cores from the datasets they used in order to focus on the dense core of the folksonomy graph. Therefore, they completely change the character of the tag recommendation problem.

Unlike these cited works, FasTag takes into account the proximity between the users whether they are directly or indirectly connected. Thus it allows a broader consideration of a user's neighborhood. Furthermore it faces scalability problems by controlling the network traversal.

4.4 Experimentation

4.4.1 Datasets

We chose five datasets from four online systems (delicious[2], Movielens[3], Last.fm[4], and BibSonomy[5]) and a larger one from delicious.We took the first fives datasets from *HetRec 2011* [24] and Bibsonomy. For the latter, we have a post-core at level 5 and a one at level 2 (more precisely, the dataset of the Task 2 of ECML PKDD Discovery Challenge 2009). Let us remind that a post-core at level p is a subset of a folksonomy with the property, that *each user, tag and item has/occurs in at least p times*. For more information regarding post-core have a look at [68]. We call these two datasets respectively $bibson5$ and $dc09$[6]).

We use the larger dataset specially for large scale experiments, in order to demonstrate the efficiency of our approximation approach. It is the same one in [46]. Table 4.2 presents the characteristics of the datasets.

4.4.2 Evaluation Measures and Methodology

To evaluate FasTag, we used a variant of the leave-one-out hold-out estimation called LeavePostOut [69]. In all datasets except $dc09$, we randomly picked, for each user u, one item i, which he had tagged before. Thus we created a test set and a training one. The task of our recommender was then to predict the tags the user assigned to i. Moreover for each training set we inferred a social network between users (i.e., their similarities) by computing the Dice coefficient between their posts. Therefore in our experimentations, the user proximity refers to their similarity and not trust relationship. We could not find a tagging dataset with a trust network. However the notions of similarity and trust

2. `http://www.delicious.com`
3. `http://www.grouplens.org`
4. `http://www.lastfm.com`
5. `http://www.bibsonomy.org`
6. `http://www.kde.cs.uni-kassel.de/ws/dc09/`

Table 4.2: Characteristics of the datasets

Dataset	$\|U\|$	$\|I\|$	$\|T\|$	$\|T(u,i)\|$
bibson5	116	361	412	2,526
dc09	1,185	22,389	13,276	64,406
delicious	1,867	69,226	53,388	104,799
last.fm	1,892	17,632	11,946	71,065
movielens	2,113	10,197	13,222	27,713
Large scale datasets (LS)				
delicious(LS)	533K	3,636K	2,442K	46,475K

are not so far. The more similar two users are, the greater their likely established trust would be considered [134, 102]. We can cite studies like [134, 17, 157, 44] which suggest that there is a strong correlation between both trust and users' profile similarity. They pointed out a significant correlation between the expressed trust between the users and their similarity based on the recommendations they made in the system.

As performance measures we use F1-measure which is standard in such scenarios [91]. Thus for each dataset, we compute the average F1-measure of the top-*5* recommendations for all the couples (u, i) in its test set. We repeat this process ten times for each dataset (except $dc09$), each time with another item per user, to further minimize the variance. Thus in the sequel, the listed F1-measure values are always the averages over all ten runs.

We set the parameter α to 0.5 for our experimentation. We kept this value as it is a compromise one for both the user and the item. Of course this may be not optimal for all our datasets but it allows us to avoid seeking its best value for all the training sets we have. We ran our experiments on a linux computer (Intel/Xeon x 24 threads, 2.93 GHz, 64 GB).

4.4.3 Effectiveness of FasTag

The purpose of this section is to evaluate the recommendation quality of FasTag in comparison with existing tag recommendation techniques. We compare it with three baseline tag recommenders. As first baseline we took the most popular mix algorithm, $score(t|u, i) = \alpha \times \rho(t, i) + (1-\alpha) \times \rho(t, u)$, which is a component of the scoring function of FasTag (see Equation 4.5). We chose this baseline in order to highlight the contribution of its social component. We refer to it by $\rho(t|u, i)$. As second baseline we took the well-known tag recommender $FolkRank$, a tripartite graph-based tag recommender designed in the spirit of PageRank [68]. Finally we took as last baseline STRec. The latter uses similar optimization as FasTag but differs from it in term of the scoring function (see [50] for more details). On Table 4.3, we can see that FasTag outperforms these baselines except on the dataset of *delicious* where it fails to give the best recommendations. We can first explain it by the fact that on this dataset 92% of the items are tagged by at most two users and 65% by only one user.What is very small and might lead to very low similarity values inferred from this dataset. This in turn decreases the performance of the neighborhood-related component of the FasTag model in Equation 4.5. Second, we have not looked for the best values for the parameter α on all the datasets. We set the parameter α to 0.5 for all the datasets as a compromise between the two popularities of tags to merge (i.e., $\rho(t,u)$ and $\rho(t,i)$), which is of course not necessary optimal for each dataset. In second step, we compare our four strategies for FasTag that we presented

Table 4.3: Comparison of FasTag with some baselines

Dataset	F1-value			
	$FasTag$	$\rho(t\|u,i)$	$FolkRank$	$STRec$
bibson5	**0.468**	0.463	0.411	0.389
movielens	**0.179**	0.170	0.167	0.146
delicious	0.170	0.187	**0.192**	0.103
last.fm	**0.313**	0.311	0.298	0.274
dc09	**0.315**	0.308	0.285	0.305

in Subsection 4.2.2. Let us notice that we compare them only in term of F1's measure. Table 4.4 presents the results. For the ones of scalability we used the largest dataset, see Subsection 4.4.5. We note that similarity propagation leads to some improvements.

Table 4.4: Comparison of the qualities gained by the four strategies

Dataset	F1-value			
	$FasTag_1$	$FasTag$	$FasTag_{++}$	$FasTag_{\infty}$
bibson5	0.430	0.468	0.468	0.467
movielens	0.175	0.179	0.180	0.181
del.ici.ous	0.181	0.170	0.170	0.170
last.fm	0.289	0.312	0.313	0.313
dc09	0.111	0.313	0.315	0.315

Indeed the lack of similarity propagation (i.e., the strategy $FasTag_1$) gives the worst recommendation quality except on the dataset of *delicious*. We have above explained this fact by the characteristics of this dataset.

4.4.4 Comparison with the result of ECML PKDD challenge 09

The Task 2 of ECML PKDD Discovery Challenge 2009 was especially intended for methods relying on a graph structure of the training data only. To the best of our knowledge, this is the most recent challenge focusing on tag recommendation. It is well adopted by the tag recommendation community as a reference. The other challenges that occur from 2010 to 2013 are useless for tag recommendation[7]. The user, item, and tags of each post in the test data are all contained in the training data. The latter is a post-core at level 2. According to the rules of the challenge, the F1-score is measured over the Top-5 recommended tags, even though one is not forced to always recommend five tags as the first two winners did [114, 90]. We set the parameter α to 0.85 after a calibration over the training set of the dataset *dc*09. Table 4.5 below shows our obtained scores compared to the final results. FasTag reaches the fifth place with a score of 0.3204.

Table 4.5: Result of the task 2 of ECML PKDD Discovery Challenge 2009

Rank	Method	F1-value
1	PITF	0.35594
-	**FasTag + *blsC***	**0.34791**
2	Relational Classification	0.33185
3	Content-based	0.32461
4	Content-based	0.32230
-	**FasTag**	**0.32044**
5	Content-based	0.32039
6	Personomy translation	0.31396
⋮	⋮	⋮

Furthermore, to get a more fair comparison with the first two winners, we adjust our recommendations using an optimization algorithm that we proposed in [52]. We called it *blsC*. The latter is able to optimize the size of recommendation lists in order to get more accuracy. It improves noticeably our score, placing us at the second position. We measure an improvement of 8.57%, which demonstrates the efficiency of *blsC*. We present this algorithm in our next section.
Let us emphasize that FasTag targets online applications where recommendations must be fast and good. This discards many model-based approaches that require prohibitive computation to learn the model parameters like those from non-distributed matrix factorization [8]. For comparison, in our experimentation the tensor factorization model, which is non-distributed, and which won the first place takes 89 seconds for each learning step. Thus it needs around 12 hours to achieve 500 iterations while FasTag make all its recommendations in real time and in 120 seconds. Furthermore FasTag uses less parameters to tune and the use of the *blsC* method improves a lot its efficiency.
Let us notice that FasTag outstrips the other graph-based approaches proposed during this challenge.

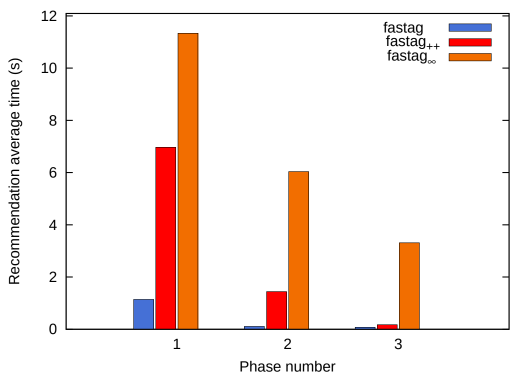
7. They respectively focus on web content quality (2010), recommendation of video lectures (2011), hierarchical text classification (2012) and recommendation of given names (2013)

4.4.5 Scalability of FasTag

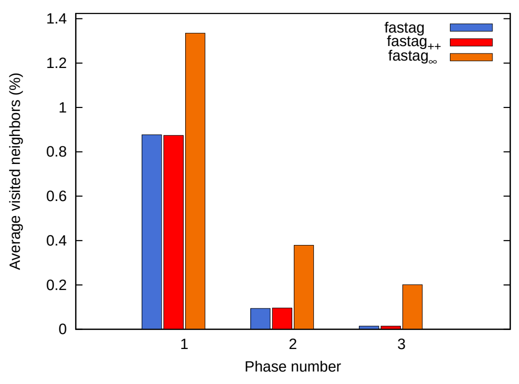
The objectives of these experiments is to demonstrate the scalability of FasTag in case of large user networks. We used the large dataset *delicious*(LS). We aim to show that our approach is tractable at large scale. In other words, we check if accessing only a small number of neighbors (wrt. to the size of the network) is sufficient to deliver recommendations. To measure the impact of our new edge addition strategy, we build a scenario where a user will request up to 3 recommendations during the test. We expect the second and third recommendation to benefit from the new edges. To this end, we build a test set containing 50,000 posts from three groups of users, denoted G_1, G_2 and G_3. The number of posts per user is respectively 1, 2, and 3 for the groups G_1, G_2, and G_3. We schedule the posts in the test set into three phases such as the first posts of all the users come first (phase 1), followed by the second posts of the users in G_2 and G_3 (phase 2), followed by the third posts of the G_3 users (phase 3).

For each recommendation process, we measure the percentage of visited neighbors, the traversal depth and the computation time. We report the respective average values (for each phase 1, 2, and 3 on the x-axis) on Figures 4.2b, 4.2c and 4.2a. In Figure 4.2a, we observe that the recommendation average computation time is decreasing from phase 1 to phase 3. FasTag speeds up when it recommends the same user (for different items). Moreover, the computation time of all the strategies is decreasing. This is because the overhead of detecting partitions only impacts recommendations made at the beginning of phase 1. The recommendations made during phase 2 and phase 3 are aware of the partitions, thus they can stop earlier. Figure 4.2b confirms this fact. The percentage of neighbors visited during a recommendation process is also decreasing from phase 1 to phase 3. Furthermore, the $FasTag_{++}$ strategy still remains better than $FasTag_{\infty}$. That validates the interest of the approximation of tag scores to bound the network traversal.
However at the beginning $FasTag_{++}$ may be more costly than $FasTag$ (due to the cost of new edges addition) but we observe that it seems to catch up with it, as it goes along. Besides, Figure 4.2c confirms that. It shows that the average depth of network traversal of $FasTag_{++}$ is lesser thanks to the new edges that directly connect most of the relevant users. These three experiments demonstrate the contribution of the approximation of tag scores, the detection of partitions and new edge addition. Although the latter brings an additional cost, it becomes beneficial when enough users are connected. Moreover we use it in a lesser extent as we make recommendations.

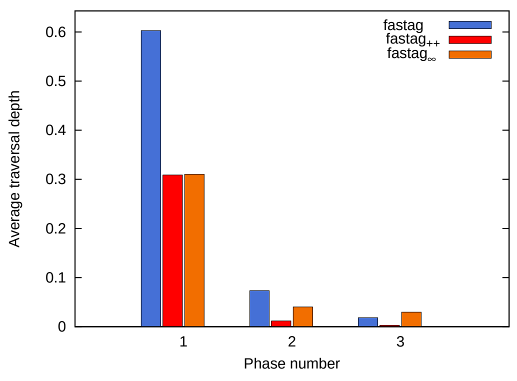
Finally, we validate FasTag scalability when it is facing an increasing demand. FasTag is able to handle several simultaneous recommendation requests, and process them in parallel. In 4.2d, we show that FasTag recommendation throughput is linearly increasing with respect to the degree of parallelism (i.e. the number of parallel instances). This makes FasTag a sound candidate for large scale use cases.



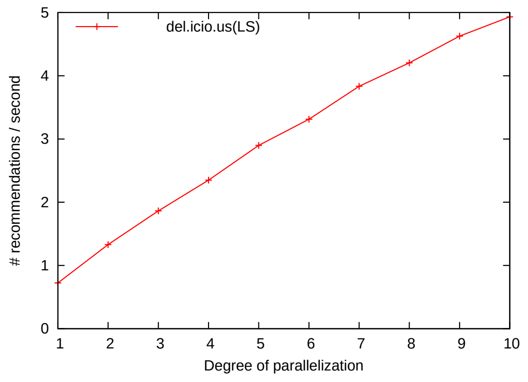
(a) Recommendation average time



(b) Percentage of involved neighbors



(c) Depths of network traversals



(d) Parallelization degree vs throughput evolution

Figure 4.2: Results about the scalability of FasTag

4.5 Conclusion

In this chapter, we presented FasTag, a tag recommender which combines the popularity of tags and the similarity between users. It considers a user's neighbors as a circle of trusted experts, which allows to better understand the user's tagging behavior. Our implementation on different datasets showed the efficiency of this approach.
Moreover, an important aspect of FasTag is its ability to bound the network traversal while considering the whole network. This reduces significantly the computation time of the recommendations as our experimentation pointed out.

5 | Optimizing Tag Recommendation List Size

As we showed in the previous chapter, tag recommenders aim to suggest the most suitable tags to a user when tagging an item. Therefore, one of their main challenges is the effectiveness of their recommendations. People generally focus on techniques that enable retrieving the best suitable tags to give beforehand, with a fixed number of tags at each recommendation.

In this chapter, we follow another direction in order to improve tag recommendation accuracy. We aim to dynamically adjust the number of tags to recommend. In other words, consider $L_N = \{t_1, t_2, \dots, t_N\}$ the list of N tags to be recommended to a user, the goal is to substitute L_N by one of its sublists that is more accurate, and provide a better quality of recommendation.

We consider all the sublists of L_N in increasing size (i.e., the prefixes : L_1, L_2, $\dots L_{N-1}$ and L_N) so as to keep the tag order as illustrated below[1].

$$\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{t_1}_{L_1} \quad t_2}_{L_2} \quad t_3}_{\dots} \quad \dots}_{L_{N-2}} \quad t_{N-1}}_{L_{N-1}} \quad t_N}_{L_N}$$

We introduce a relevance measure for the recommended lists $Rel(L_N|u,i)$, which estimates the probability that a user u will use all the recommended tags in L_N for the tagging of an item i. Based on this measure, we compute the best list (the one having the optimal size, *bls*) that will be finally recommended to the user. We define the optimal size as follows:

$$bls = \max\left(s \mid s \in \mathcal{S}\right) \tag{5.1}$$

with

$$\mathcal{S} = \Big\{ s \;\Big|\; s \leq N \wedge \forall n \leq N,\ Rel(L_n|u,i) \leq Rel(L_s|u,i) \Big\}$$

1. $L_N = L_{N-1} \cup \{t_N\} = L_{N-2} \cup \{t_{N-1}, t_N\} \dots$

Existing approaches use linear combinations to compute the global average number of tags per post, the one related to a user and/or the one specific to an item [90, 113, 114]. These combinations are then used to infer a fixed list size. Such approaches need some calibrations which are generally difficult to set, and they lack of dynamicity which limits their accuracy.
The algorithm we propose here enables adjusting dynamically the size of the recommended list of tags, and then increases the accuracy of the recommendations. It is a parameter-free algorithm that adjusts the list of recommended tags by discarding those which are estimated irrelevant to the user for the tagging of the item. Our method looks like an add-on filter on top of a tag recommender. It estimates the sublist which gives the best accuracy. We present in this chapter two relevance measures and the algorithm that we use to retrieve the optimal sublist from a given tag recommendation list.

To evaluate the efficiency of our approach we implement it on top of four tag recommenders, from different approaches. One of our candidates is the pairwise interaction tensor factorization model (PITF) of Rendle and Schmidt-Thieme which won the task 2 of the ECML PKDD Discovery Challenge 2009 [114]. It is still considered in the literature as one of the best tag recommenders. We took also the well-known tripartite graph-based algorithm FolkRank [65] and the "Most Popular Tag"recommendation approach [69]. As last candidate we chose FasTag, network-based tag recommender we proposed in Chapter 4. It computes the list of tags based on the opinions of the users' neighborhood and their tagging posts.
The experiments we did on the same five datasets that in the previous chapter demonstrate the efficiency of our optimization technique.

The remainder of this chapter is organized as follows. In Section 5.1 we present some preliminaries and describe briefly the four tag recommender candidates. Section 5.2 details the related work and presents our approach for finding the best size of a tag recommendation list to keep. In Section 5.3, we present our experiments and conclude in Section 5.4.

5.1 Preliminaries

We rely on the same notations and definitions as in Section 4.1 of Chapter 4. Thus we consider a folksonomy $\mathbb{F} := (U, I, T, S)$ and its collection of a set of users U, set of tags T, set of items I, and the ternary relation between them $S \subseteq U \times I \times T$. We still assume that a user can tag an item with a given tag at most once.

The interest of a tag t for a given user u to annotate an item i is estimated by its score $score(t|u, i)$. Hence, the purpose of a tag recommender is to compute the top-K highest scoring tags for a post (u, i) which represents its recommendations.

$$Top(u, i, K) = \underset{t \in T}{\overset{K}{argmax}}\ score(t|u, i) \tag{5.2}$$

In the following we describe how our four tag recommender candidates model the scores associated with the tags.

5.1.1 Factor Models for Tag Recommendation

Factorization models are known to be among the best performing models. They are a very successful class of models for recommender systems where they outperform the other approaches.
We chose the pairwise interaction tensor factorization model (PITF) of Rendle and Schmidt-Thieme in our experimentation due to its efficiency [114]. Indeed it took the first place of the ECML PKDD Discovery Challenge 2009 for graph-based tag recommendation.

PITF proposes to infer pairwise ranking constraints from the set of tagging triples S. It captures the interactions between users and tags as well as between items and tags. The equation of PITF's model is given by:

$$score(t|u,i) = \sum_f \hat{U}_{u,f} \cdot \hat{T}^U_{t,f} + \sum_f \hat{I}_{i,f} \cdot \hat{T}^I_{t,f} \tag{5.3}$$

Where $\hat{U}$, $\hat{I}$, $\hat{T}^U$ and $\hat{T}^I$ are feature matrices which capture the latent interactions.

The main assumption of PITF is that within a post $T(u,i)$, a tag t can be preferred over another tag t' iff the tagging triple $(u,i,t) \in S$ (i.e., has been observed) and not (u,i,t'). PITF models these preferences in Equation 5.3 such that the score of a tag which is more preferred than another one is greater.

5.1.2 FolkRank - A Topic-Specific Ranking

FolkRank is a tripartite graph-based tag recommender designed in the spirit of PageRank [65, 68]. Its assumes that a tag becomes important when it is used by important users or for tagging important items. It also takes the same principle for users and items. Therefore FolkRank represents a folksonomy $\mathbb{F}$ as a graph where the vertices are mutually reinforcing each other by spreading their weights.

Let $G_{\mathbb{F}} = (V,E)$ be this graph. Its vertices are the users, items and tags (i.e. $V = U \cup I \cup T$) and the edges defined between them such as if a tagging triple $(u,i,t) \in S$ then $\{\{u,i\},\{u,t\},\{i,t\}\} \subset E$.
Let v_i be a vertex of this graph, i.e. $v_i \in V$. We denote by $\mathcal{N}(v_i)$ the set of neighbors of v_i and by $w(v_i,v_j)$ the weight of the edge between the vertices v_i and v_j. The weight of an edge is here the number of times its two vertices appear together in the tagging triples in S. From that, the degree of a vertex v_i is defined as follows:

$$w(v_i) = \sum_{v_j \in \mathcal{N}(v_i)} w(v_i,v_j) \tag{5.4}$$

FolkRank ranks the vertices according to their importance computed as follows:

$$PR(v_i) = \lambda \sum_{v_j \in \mathcal{N}(v_i)} \frac{w(v_i,v_j)}{w(v_j)} \cdot PR(v_j) + (1-\lambda) \cdot p(v_i) \tag{5.5}$$

where $PR(v_i)$ is the PageRank value and $p(v_i)$ the preference value of the vertex v_i. Hence a straightforward idea for tag recommendation is to set more preference to the user and item to be suggested for, and then compute ranking values using PageRank as in Equation 5.5. The parameter λ determines the influence of $p(v_i)$. Its value is between 0 and 1.

To recommend some tags for a user u and an item i, FolkRank uses two random surfer models on the graph, $s^{(0)}$ and $s^{(1)}$, to infer the importance of each vertex of the graph.
The first surfer, $s^{(0)}$, set the same preference value to all the vertices ($p(v) = 1, \forall v \in V$) while the second, $s^{(1)}$, set their preference values to 0 except for the user u and the item i for which $p(u) = 1$ and $p(i) = 1$.

After the execution of the two surfers, the difference $s := s^{(1)} - s^{(0)}$ is computed. Then the tags are ranked according to their importance values in s (what represent their scores) and the first recommended to user u for tagging the item i.

5.1.3 Recommending the Most Popular Tags

The rational of Most Popular Tags' Recommenders (MPTR) is that when a tag is popular (i.e., frequently used) for an item and/or by a user, it may be relevant to recommend to the user when tagging the item. Hence MPTR models these popularities in their scoring models. A well-known MPTR model consists in adding the tag popularities (i.e. the ones of the user with those of the item) after normalizing and weighting them as follows:

$$score(t|u,i) = \alpha \cdot \frac{|U(i,t)|}{|U(i)|} + (1-\alpha) \cdot \frac{|I(u,t)|}{|I(u)|} \tag{5.6}$$

The parameter α in Equation 5.6 allows to tune the relative importance of the item or user-related tag popularity with respect to the second. When $\alpha = 1$ we keep only the tags which are most specific to the item. On the other hand, when we set it to 0, we consider only the user's popular tags. By default we fix α to 0.5 in our experimentation. Thus, we consider the user tags as much important as those associated to the item. The advantage of MPTR is that they are fast to compute, while giving good predictions [69].

5.1.4 Social and popularity-based Recommender

One weakness of MPTR (which entirely relies on popularity measures) is that it is not able to decide between tags with close popularities. Furthermore, some particular users can have their own vocabulary (i.e., tags) and the popular tags of the item may not be relevant for them. Thus, having reliable opinions about the tags from some trusted neighbors, in addition to the popularities of tags, may be a great asset to make better recommendations.
FasTag [53], we presented in Chapter 4, uses such an approach. It models the relevance score of a tag t for a user u and an item i (i.e. $score(t|u,i)$) as a popularity-dependent component, based on a user's proximity with her neighbors in the network. Let us denote by $score^{MPTR}(t|u,i)$ the scoring model of MPTR in Equation 5.6, the scoring one of

FasTag is defined as

$$score(t|u,i) = score^{MPTR}(t|u,i) \cdot \Big(1 + \eta(t|u,i)\Big) \tag{5.7}$$

where $\eta(t|u,i)$ represents the opinion of the user's neighbors about tag t. It is a normalized sum of the user's proximity values associated to her neighbors who already tagged this item with t. The rationale of this *score* function is to estimate a relative popularity of a tag depending in the vicinity of a user: i.e., the more a user is similar to a neighbor, the more this neighbor's opinion contributes in the user recommendations. The sum $1 + \eta(t|u,i)$ enables taking into account the isolated users (when $\eta(t|u,i) = 0$).

We chose FasTag as a candidate for network-based tag recommenders, since it is fast and efficient as shown in [53]. Its scoring model is not only based on the proximity associated to the direct neighbors of the user, it also considers proximity propagation, following a natural interpretation that it is, at some extent, transitive. See [130, 158] for more details on propagation models.

5.2 Adjusted Recommendation list size

In this section, we present two ways to choose the best recommendation list size. The first one is based on existing works employing linear combination techniques, and the second one is our proposal to optimize dynamically the size of the recommended list.

5.2.1 Linear combination models

To choose the best list size (bls) of tags to recommend, usual approaches use some linear combinations of the global average number of tags per post, the one related to a user and/or the one specific to an item [114, 113, 90]. Therefore, we take as baseline the following general linear combination model

$$bls = \min\left(K, \lfloor \lambda + (\beta_G \cdot \mu_G) + (\beta_u \cdot \mu_u) + (\beta_i \cdot \mu_i) \rfloor\right) \tag{5.8}$$

where K stands for the maximum number of tags to recommend, i.e. the maximal list size; μ_G the global average number of tags per post; μ_u the average number of tags per user and per post, and μ_i the average number of tags per item and per post. The rest stands for parameters which allow us to make a lot of possible combinations. For instance, if we set the parameter β_u to 1 and all the other parameters to zero, we obtain as a list size the average number of tags per user and post. We denote in the following the linear combination method by *LC_bls*.
For our experiments, we apply a grid search in order to find optimal parameters to keep. We test $K \times 1,000$ combinations of these parameters each time a top-K query is asked (i.e., each time we look for a list of at maximum K tags). For instance, we make 10,000 combinations for the top-10 query and 5,000 for the top-5 one. We vary the parameter λ from 0 to $K-1$, each time by a step of 1. And using nested loops, we vary each of the other parameters from 0 to 1 by a step of 0.1. Thus, we test enough combinations with

different values of parameters. At the end, we keep the combination that gives the best result (in terms of reached F1-measure). For our experiments, we apply a grid search in order to find optimal parameters to keep. We test $K \times 1,000$ combinations of these parameters each time a top-K query is asked (i.e., each time we look for a list of at maximum K tags). For instance, we make 10,000 combinations for the top-10 query and 5,000 for the top-5 one. We vary the parameter λ from 0 to $K-1$, each time by a step of 1. And using nested loops, we vary each of the other parameters from 0 to 1 by a step of 0.1. Thus, we test enough combinations with different values of parameters. At the end, we keep the combination that gives the best result (in terms of reached F1-measure).

5.2.2 The blsC algorithm

blsC denotes the algorithm we propose to find the best list size. Let $Rel(t|u,i)$ be the relevance of a tag t, according to a user u, for the tagging of an item i. And $L_N = \{t_1, t_2, \ldots, t_N\}$ is an ordered list of N tags. We define the relevance of the list L_N, for both a user u and an item i, as follows:

$$Rel(L_N|u,i) = \omega(L_N|u,i) \cdot \frac{\sum_{t \in L_N} Rel(t|u,i)}{N} \tag{5.9}$$

In this formula, $\omega(L_N|u,i)$ stands as a weight for the adjustment of the list relevance. It allows us to promote longer lists than the others. We will give its definition shortly in Subsection 5.2.2.2. $Rel(t|u,i)$ is the relevance of a tag t for a user u and an item i. Intuitively, it measures the probability that user u will tag the item i with the tag t.

Let Max be a maximal list size and $Rel(L_{Max}|u,i)$ the relevance of this list (L_{Max}). We look for the best list size starting from Max down to 1. At each step, we compute the relevance of the current list and update the best list size as shown in Algorithm 4. In case we obtain the same relevance for two different lists, we choose the longest one.

Algorithm 4: *blsC*: Best list size Computation

```
Input: $Max$, $L_{Max}$ /* Initial recommended tags list*/
Output: $bls$    /* Suggested number of tags to keep */
1 $bls \leftarrow Max$                /* bls : Best list size */
2 $blR \leftarrow Rel(L_{Max}|u,i)$   /* blR : Best list relevance */
3 for $N = (Max - 1)$ to 1 do
4     if $Rel(L_N|u,i) > blR$ then
5         $bls \leftarrow N$
6         $blR \leftarrow Rel(L_N|u,i)$
7     end
8 end
9 return $bls$
```

To compute the relevance of a tag, we propose two solutions. In the first one, we distinguish the known tags from the others and assign them different relevance values.

In the second solution, we link the relevance values of the tags to some statistics we obtain from the available data (our training sets).

5.2.2.1 Simple relevance measure

Making a distinction between the known tags (those already used by the user and already associated to the item) from the others may be a great factor to determine the relevance of a recommended list of tags. Our intuition is that the tags already linked to the user u and also the item i are more relevant to be recommended than the others.

Let $P_N = L_N \cap T(u) \cap T(i)$ be the sublist of L_N containing the known tags. We assign a unique high relevance value, $R_{elev_{max}}$, to the tags in P_N and an unique low one, $R_{elev_{min}}$, to the other tags. By setting the weight of each list to one, we can rewrite the Equation 5.9 as follows:

$$Rel(L_N|u,i) = \frac{\sum\limits_{t_j \in P_N} R_{elev_{max}} + \sum\limits_{t_j \in \{L_N \setminus P_N\}} R_{elev_{min}}}{N} \tag{5.10}$$

After simplification it becomes:

$$Rel(L_N|u,i) = \frac{\big(|P_N| \cdot R_{elev_{max}} + (N - |P_N|) \cdot R_{elev_{min}}\big)}{N} \tag{5.11}$$

From Equation 5.11, one can see that the relevance of a recommendation list depends mainly on the ratio between $|P_N|$ and N. The relevance is an increasing function, having its input from the interval $[0, N]$. It reaches its maximum when $|P_N| = N$. Thus we can simply consider the relevance of a recommendation list as follows:

$$Rel(L_N|u,i) = \frac{|P_N|}{N} \tag{5.12}$$

With this formula, we can easily adjust the list size in order to obtain the best relevance value. One can see $Rel(L_N|u,i)$ as a density measure of known tags in the recommended list. Then, the *blsC* is just seeking for the best density.

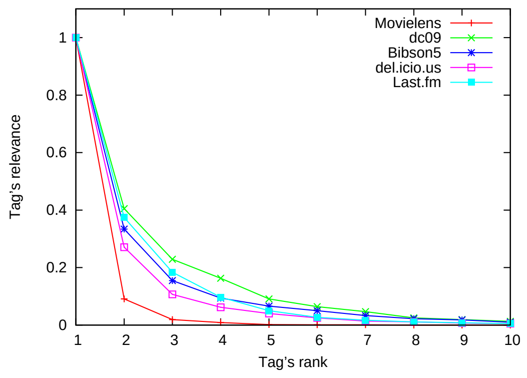
5.2.2.2 Refining the relevance measure

Among the weaknesses of the relevance measure given in Equation 5.12 we can mention the fact that it fails to give a sublist when all the tags in the recommendation list are known (in P_N). We propose here a second relevance measure which faces the drawbacks of the previous solution by taking into account the popularity of the tags for both the user and the item. The latter is our very proposal for optimizing the size of recommendation lists.

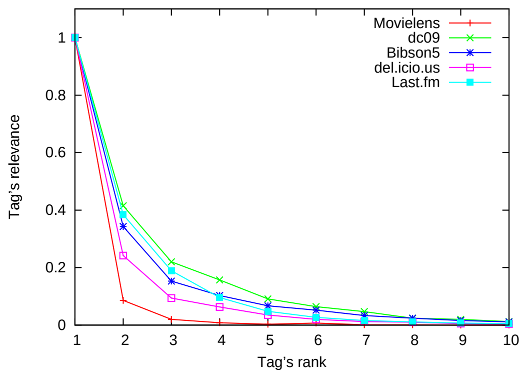
$$Rel(t|u,i) = \frac{|U(i,t)|}{|U(i)|} \times \frac{|I(u,t)|}{|I(u)|} \tag{5.13}$$

Naturally, we expect that the relevance of a tag decreases according to its rank in a recommended list (high relevance for the first tag and low relevances for the last ones). This intuition is confirmed by our experimentations, on all the datasets we used and for

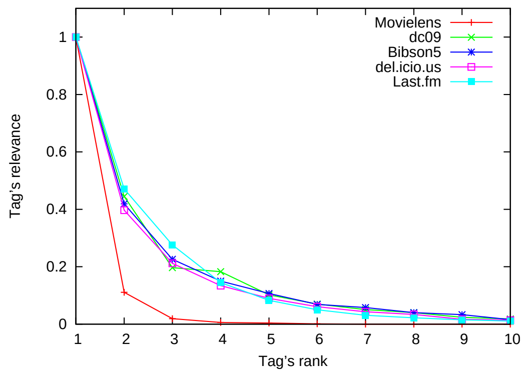
all the tag recommenders we tested, as shown in Figure 5.1. We draw the evolution of the relative relevances of tags according to both their positions in a recommendation list and the relevance of the first tag of the list.
We limit our tests to the top-10 lists of recommended tags.



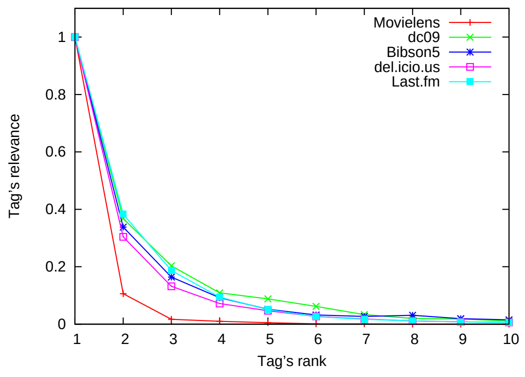
(a) MPTR



(b) FasTag



(c) PITF



(d) FolkRank

Figure 5.1: Relative relevance vs tag position in a recommendation list

From these observation, one can see the necessity to use weighted means when evaluating the relevance of recommended lists. Indeed, since a recommendation list is ordered, the first tag is generally more relevant than the rest, then the second one and so on. Thus, using the average of the relevances of the tags contained in a list may still lead to a singleton, i.e. L_1 as the best list.
Thus to take into account this natural decrease of the relevance of tags according to their positions, we model the weight $\omega(L_N|u,i)$ of a recommendation list L_N with a

Zipf-Mandelbrot law [2] and define it as follows:

$$\omega(L_N|u,i) = \frac{N}{\sqrt{\frac{1}{2}\left(N + Max\right)}} \tag{5.14}$$

Our weight function estimates the importance of a list size compared to the maximal possible list size. It allows us to penalize short recommendation lists while promoting the long lists. Therefore, from Equation 5.9 we introduce a new list relevance measure:

$$Rel(L_N|u,i) = \frac{\sum_{t \in L_N} Rel(t|u,i)}{\sqrt{\frac{1}{2}\left(N + Max\right)}} \tag{5.15}$$

As we can see, we do not compute the mean of the relevances of tags but a relative list relevance according to the greatest possible list size. The comparison of the relevance of all the sublists is done as described in Algorithm 4. We just compute the most relevant list size from Max down to 1 by using this new relevance measure given in Equation 5.15. We denote this second proposal *blsC_v2*. The latter can overcome the weak points of *blsC*. Indeed, even if all the tags in a recommendation list are in P_N, *blsC_v2* does not rely on their presence or not but only on their relevance. Thus, it is able to decide when *blsC* fails.

5.3 Experimentation

We demonstrate in this section the effectiveness of our proposal. We led a set of experiments with four tag recommender candidates on five publicly available datasets. In the next two subsections, we describe shortly these datasets and the evaluation measures and methodology we used. Then we present the results we obtained.

5.3.1 Datasets

We chose the same five datasets as in Chapter 4. Table 5.1 presents some details of these datasets.

5.3.2 Evaluation Measures and Methodology

To evaluate our proposal, we used a variant of the leave-one-out hold-out estimation called LeavePostOut [69]. In all datasets except $dc09$, we picked randomly and for each user u, one item i, which he had tagged before. Thus we create a test set and a training one. The task of our recommender was then to predict the tags the user assigned to the item.

2. The Zipf-Mandelbrot law is a power-law distribution on ranked data. It is well-known for its statement that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table

Table 5.1: Characteristics of the datasets

dataset	$\|U\|$	$\|I\|$	$\|T\|$	$\|T(u,i)\|$
dc09	1,185	22,389	13,276	64,406
Last.fm	1,892	17,632	11,946	71,065
delicious	1,867	69,226	53,388	104,799
Movielens	2,113	10,197	13,222	27,713
Bibson5	116	361	412	2,526

On each dataset, we run the four tag recommenders. We asked them to give successively a top-*1*, then a top-*2* and so on up to a top-*10*. Then we apply *LC_bls*, *blsC* and *blsC_v2* on their tag recommendation lists in order to get a better sublist. We use the F1-measure as performance measure.

Let us notice that for all the experiments, we set the parameter α of MPTR and FasTag to 0.5 (see Equation 5.6). Similarly we set the parameter λ of FolkRank to 0.7 (see Equation 5.5) as in [68]. For PITF we use the software[3] and the parameters given by the winners of the task 2 of the ECML PKDD Discovery Challenge 2009 but we do not rely on ensembling factor models as they did in [113]. We only compute one model with 64 factors as the dimensionality and a regularization of $5 \cdot 10^{-5}$. We stop learning after $2,000$ iterations.

5.3.3 Experimental Results

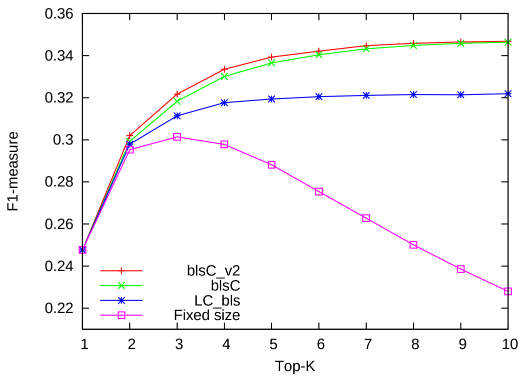
We present in this part the results of our experimentation. We aim to point out that our proposal outstrips the methods based on linear combinations.

5.3.3.1 Effectiveness of our proposal

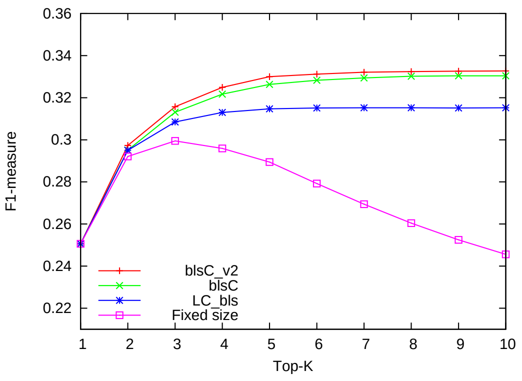
We show here the effectiveness of our proposal. We implemented them on top of the four tag recommenders. Figure 5.2 shows the average F1's values over the five datasets of the original recommendation lists given by each tag recommender and the ones of each optimization method (i.e., *blsC*, *blsC_v2* and *LC_bls*). On each of these figures, the x-axis gives the original number of tags to recommend before length optimization. In almost all these figures we see that the quality of the size-adjusted recommendation lists is increasing while the one of those with fixed sizes decreases when they exceed a certain size. This demonstrates the importance of giving optimal recommendation list size.

Second, in the most cases *blsC_v2* outperforms *blsC* and linear combinations. For instance, it outstrips the results of the task 2 of the ECML PKDD Discovery Challenge 2009. Indeed with the same tag recommender than the winners, we reach an F1 measure of 0.366 while they got 0.356 despite they used linear combinations to adjust the sizes of

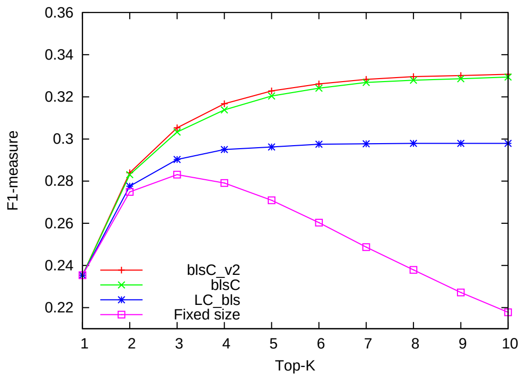
3. `http://bit.ly/1qL6NeF`



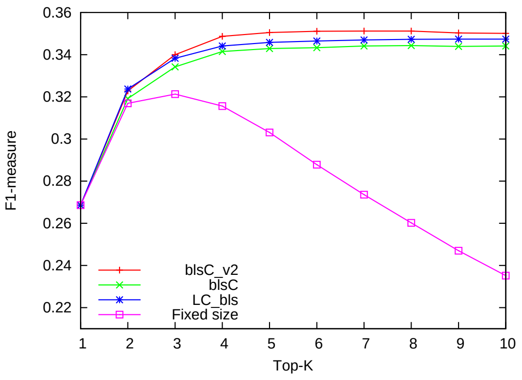
(a) MPTR



(b) FasTag



(c) FolkRank



(d) PITF

Figure 5.2: Quality increases vs recommendation list sizes

the recommendation lists [113]. *blsC_v2* yields 3% of improvement over their F1 score on this dataset.

Let us notice that linear combinations surpass *blsC* and *blsC_v2* on only 9.19% of cases. On all the rest *blsC_v2* dominates with more than 60% of cases, then *blsC* follows with 30%. Tables 5.2 and 5.3 give their comparison in term of percentage of cases where each of them give the best contribution. In our experimentations, linear combinations are specifically better on the Movielens' dataset which is the worst among our five datasets for computing sound tag relevances. In this dataset, around 65% of the users have in average one tag in their posts indeed. What is very small and this is also the case of 41,76% of its items [52].

5.3.3.2 Giving up some recall for more precision

The underlying idea of *blsC* is to make a tradeoff between the precision and recall of the recommendation. We measured their evolution with and without size-fixed lists. Figure 5.3 points out the results that we obtained. One can see in this figure that size-fixed lists can gain more in recall with longer list, but they lose so more in precision.

Table 5.2: Comparison of the three methods

Method	% of cases with the best size
blsC_v2	**60.34**
blsC	30.45
LC_bls	9.19

Table 5.3: Comparison of the three methods in twos

Method	% of cases with the best size
blsC_v2	**90.85**
LC_bls	9.14

Method	% of cases with the best size
blsC	**76.96**
LC_bls	23.03

Method	% of cases with the best size
blsC_v2	**65.34**
blsC	34.09

Our proposal allows a better tradeoff. *blsC_v2* gives particulary the best ratio in terms of precision and recall.

5.3.3.3 Distribution of optimal list sizes

Table 5.4 shows the average optimal list size given by each of the methods *blsC_v2*, *blsC* and *LC_bls*. From these results, we can say that *blsC_v2* proposes longer lists than *LC_bls* in 80% of the cases, in addition to giving the best size for 90.85% of the cases compared to *LC_bls* (see Table 5.3).
In Figure 5.4, we drawed the distribution of the proposed list sizes of *blsC_v2* and *LC_bls*. One can see over it that linear combination tend to remain near to the mean size, while, in a lot of cases, *blsC_v2* can detect that the list size is already optimal and leaves it unchanged. All these experimentations show the ability of our proposal to optimize the size of recommendation lists.

5.4 Conclusion

In this chapter, we presented a new proposal that improves the accuracy of the recommendations delivered by a tag recommender system. Our solution optimizes the size of the recommended list in order to obtain a better recommendation quality. The ex-

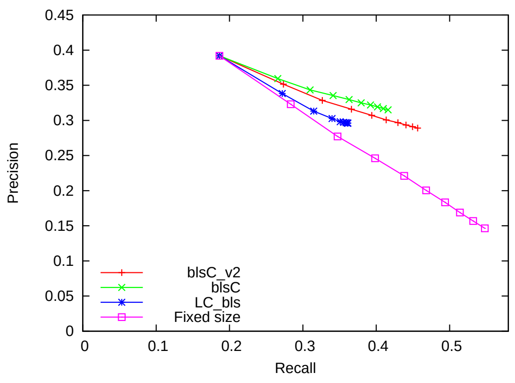
Table 5.4: Average optimal list length with 10 tags at maximum

Dataset	FasTag		
	LC_bls	*blsC*	*blsC_v2*
bibsonomy	3.28	3.73	**4.91**
movielens	1.96	6.51	**6.56**
delicious	**7.17**	2.92	3.08
lastfm	**5.94**	4.22	4.72
dc09	4.56	3.93	**4.86**

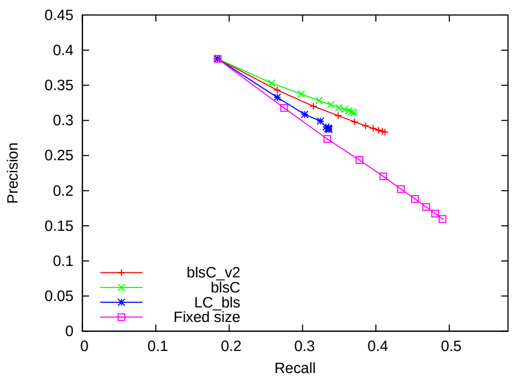
Dataset	MPTR		
	LC_bls	*blsC*	*blsC_v2*
bibsonomy	3.21	3.85	**5.00**
movielens	1.90	8.23	**8.28**
delicious	7.23	7.12	**7.35**
lastfm	**5.53**	4.81	5.34
dc09	4.56	4.57	**5.51**

Dataset	FolkRank		
	LC_bls	*blsC*	*blsC_v2*
bibsonomy	3.49	3.70	**4.94**
movielens	2.37	8.23	**8.29**
delicious	6.75	7.18	**7.51**
lastfm	**5.91**	4.76	5.28
dc09	4.41	4.44	**5.46**

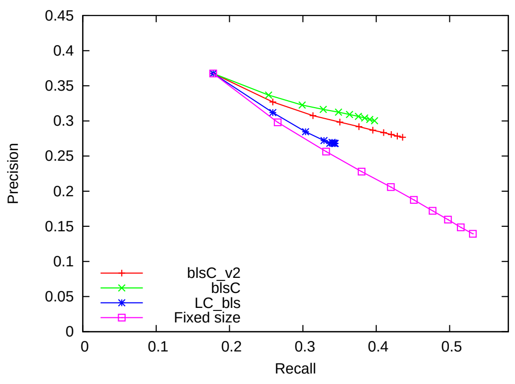
Dataset	PITF		
	LC_bls	*blsC*	*blsC_v2*
bibsonomy	3.50	6.89	**7.70**
movielens	1.67	9.37	**9.41**
delicious	5.25	9.36	**9.47**
lastfm	6.08	7.51	**7.97**
dc09	4.53	6.35	**7.38**



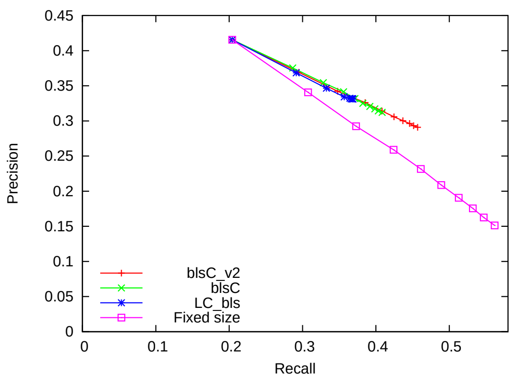
(a) MPTR



(b) FasTag

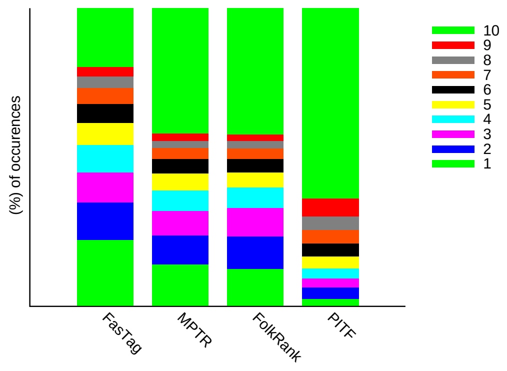


(c) FolkRank

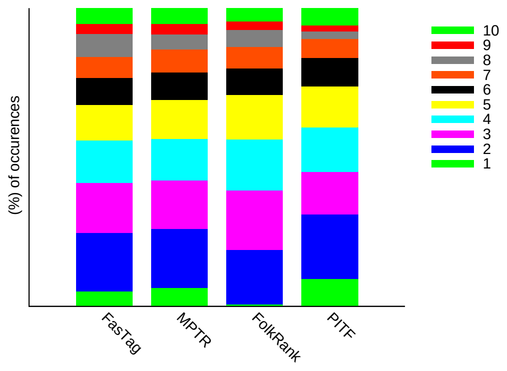


(d) PITF

Figure 5.3: Recall vs Precision



(a) blsC_v2



(b) LC_bls

Figure 5.4: Distribution of the proposed optimal list sizes

perimentations we did show the effectiveness of our approach.

Our proposal, blsC can also be implemented for item context-aware recommendation where the user and some other parameters of the recommendation's context have to be taken into account. Furthermore, our approach suits well for recommendation diversity. Since it shortens the number of relevant tags to recommend, it frees some space to include extra tags in the response, while keeping constant the total number of tags presented to the user. This allows for the selection of extra tags that maximize another score function. For instance, a diversity function would bring diversity within the recommendation process.

More generally, our solution brings new opportunities to aggregate recommendations from various recommenders while keeping recommendation list size above an given limit.

Research Perspectives

In this final chapter, we discuss some open issues and promising extensions of our work that it would be interesting to address in the future.

Making dynamic recommendation with local biases adjustment

In Chapter 3, we tackled the collaborative filtering problem of accurately recommending items to users in dynamic contexts, where new ratings are continuously produced. We proposed a matrix factorization model which incorporates cluster-based biases. We demonstrated also its efficiency to fasten the integration of new ratings without recomputing the recommendation model. It just updates some local biases. However, the need of recomputing the model will arise sooner or later. This will depend on the tradeoff between the loss in recommendation quality and the cost of learning a new model. Having the ability to determine the ideal moment to recompute the model would be a great asset. It is an interesting problem that can complement our proposal.

Using proximities propogation for better recommendations

FasTag as we presented it in Chapter 4, uses only *positive* relationships between users, we referred by their proximities. In practice, these proximities correspond to similarities deducted from users' behavior patterns or trust they have to each other. Trust and similarity are close in a certain way. [134, 17, 157, 44] assert that there is a strong correlation between both trust and users' profile similarity.
In the literature, some researchers have investigated the use of dissimilarity between users and distrust to improve the recommendations [36, 139, 87, 55]. They showed that user *negative* relation (e.g., distrust) information can be beneficial to recommender systems. We think that *FasTag* can be extended by taking account negative relationships between users. That could improve its accuracy. Furthermore, as we use heuristics to bound the network traversal, the incorporation of negative relationships may allows us to stop earlier the network traversal and then optimise the time needed to compute the recommendation.

Parameter-free methods for optimizing tag recommendation list size

In Chapter 5, we presented a new proposal that improves the accuracy of the recommendations delivered by a tag recommender system. It optimizes the sizes of recommended lists in order to increase the likelihood that all the tags are relevant for the user. In fact, tag recommendation relates back to context-/item-aware tag recommendation. Therefore, by transposition, *blsC* can also be implemented for context-aware item recommendation where both the user and some other dimensions (i.e., the context) have to be taken into account. Context-/time-aware movie recommendations is a common case study.
In this context, movies occupy more space in a web page than tags. Therefore, with the use of *blsC*, we have several possibilities to occupy the freed space. For instance, some ads may be added instead, or we can integrate a second recommmender system to fill this freed space while diversifying our suggesting. But, which impact these decisions will have on the users? What will be their reactions compared to the case where *blsC* is not used?
One may wonder to know if the conversion rate is increased, users' satisfaction and fidelity improved or is it now possible to sell more diverse items with *blsC*. In short, it seems that it still remains a lot of interesting things to do with this method.

List of Publications

In Proceedings of International Conferences and Workshops

- [49] Modou Gueye, Talel Abdessalem and Hubert Naacke. FoldCons: A Simple Way To Improve Tag Recommendation. In Recsys@RSWeb '13.

- [50] Modou Gueye, Talel Abdessalem and Hubert Naacke. STRec: An Improved Graph-based Tag Recommender. In Recsys@RSWeb '13.

- [52] Modou Gueye, Talel Abdessalem and Hubert Naacke. A Parameter-free Algorithm for an Optimized Tag Recommendation List Size. In Recsys '14.

- [53] Modou Gueye, Talel Abdessalem and Hubert Naacke. A Social and Popularity-based Tag Recommender. In SocialCom '14.

In Proceedings of National Conferences

- [47] Modou Gueye, Talel Abdessalem and Hubert Naacke. A cluster-based matrix-factorization for online integration of new ratings. In BDA '11.

- [48] Modou Gueye, Talel Abdessalem and Hubert Naacke. Factorisation multi-biais pour de meilleures recommandations. In CNRIA '13.

- [51] Modou Gueye, Talel Abdessalem and Hubert Naacke. Technique de factorisation multi-biais pour des recommandations dynamiques. In EGC '13.

Book Chapters

- [54] Modou Gueye, Talel Abdessalem and Hubert Naacke. Dynamic recommender system : using cluster-based biases to improve the accuracy of the predictions. In AKDM '15.

Bibliography

[1] M. R. Abbasifard, B. Ghahremani, and H. Naderi. A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 95(25):39–52, June 2014. Published by Foundation of Computer Science, New York, USA.

[2] G. Adomavicius and Y. Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Trans. on Knowl. and Data Eng.*, 24(5):896–911, May 2012.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17:734–749, 2005.

[4] G. Adomavicius and J. Zhang. Impact of data characteristics on recommender systems performance. *ACM Trans. Manage. Inf. Syst.*, 3(1):3:1–3:17, 2012.

[5] A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. In Shawe-Taylor et al. [125], pages 873–881.

[6] D. Agarwal, B.-C. Chen, and P. Elango. Fast online learning through offline initialization for time-sensitive recommendation. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 703–712, New York, NY, USA, 2010. ACM.

[7] S. Alag. *Collective intelligence in action.* Manning, Greenwich, Conn., 2008.

[8] X. Amatriain et al. Netflix recommendations: Beyond the 5 stars. `bitly.com/Hu482Q`, 2012.

[9] X. Amatriain, A. Jaimes*, N. Oliver, and J. Pujol. Data mining methods for recommender systems. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 39–71. Springer US, 2011.

[10] S. Amer-Yahia et al. Efficient network aware search in collaborative tagging sites. *PVLDB*, 1(1):710–721, 2008.

[11] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, June 2007.

[12] R. Barros, M. Basgalupp, A. C. P. L. F. De Carvalho, and A. Freitas. A survey of evolutionary algorithms for decision-tree induction. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(3):291–312, May 2012.

[13] R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 95–104, New York, NY, USA, 2007. ACM.

[14] R. M. Bell, J. Bennett, Y. Koren, and C. Volinsky. The million dollar programming prize. *IEEE Spectr.*, 46:28–33, 2009.

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

[16] D. Benz et al. The social bookmark and publication management system BibSonomy. *The VLDB Journal*, 19(6):849–875, 2010.

[17] T. Bhuiyan. A survey on the relationship between trust and interest similarity in online social networks. *Journal of Emerging Technologies in Web Intelligence*, 2010.

[18] J. Bobadilla, F. Ortega, A. Hernando, and A. GutiéRrez. Recommender systems survey. *Know.-Based Syst.*, 46:109–132, July 2013.

[19] A. Bouza, G. Reif, A. Bernstein, and H. Gall. Semtree: Ontology-based decision tree algorithm for recommender systems. In C. Bizer and A. Joshi, editors, *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[20] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[21] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, Apr. 1998.

[22] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12:331–370, 2002.

[23] R. D. Burke. Hybrid web recommender systems. In *The Adaptive Web, Methods and Strategies of Web Personalization*, pages 377–408, 2007.

[24] I. Cantador et al. Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011). In *ACM RecSys*, 2011.

[25] B. Cao, D. Shen, J.-T. Sun, X. Wang, Q. Yang, and Z. Chen. Detect and track latent factors with online nonnegative matrix factorization. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2689–2694, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[26] J. Cao, Z. Wu, B. Mao, and Y. Zhang. Shilling attack detection utilizing semi-supervised learning method for collaborative recommender system. *World Wide Web*, 16(5-6):729–748, Nov. 2013.

[27] A. Dattolo, F. Ferrara, and C. Tasso. The role of tags for recommendation: a survey. In *Proc. of the 3rd International Conference on Human System Interaction - HSI'2010*, pages 548–555, Rzeszow, Poland, May 2010. IEEE press.

[28] L. M. de Campos, J. M. Fernández-Luna, J. F. Huete, and M. A. Rueda-Morales. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *Int. J. Approx. Reasoning*, 51(7):785–799, Sept. 2010.

[29] J. de Wit. Evaluating recommender systems. Master's thesis, University of Twente, May 2008.

[30] M. B. Dias, D. Locher, M. Li, W. El-Deredy, and P. J. Lisboa. The value of personalised recommender systems to e-business: a case study. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 291–294, New York, NY, USA, 2008. ACM.

[31] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The yahoo! music dataset and kdd-cup'11. In *Proceedings of KDDCup 2011*, 2011.

[32] T. DuBois, J. Golbeck, and A. Srinivasan. Predicting trust and distrust in social networks. In *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), Boston, MA, USA, 9-11 Oct., 2011*, pages 418–424, 2011.

[33] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM.

[34] A. Felfernig, R. Burke, and P. Pu. Preface to the special issue on user interfaces for recommender systems. *User Modeling and User-Adapted Interaction*, 22(4-5):313–316, 2012.

[35] D. M. Fleder and K. Hosanagar. Recommender systems and their impact on sales diversity. In *Proceedings of the 8th ACM conference on Electronic commerce*, EC '07, pages 192–199, New York, NY, USA, 2007. ACM.

[36] R. Forsati, M. Mahdavi, M. Shamsfard, and M. Sarwat. Matrix factorization with explicit trust and distrust relationships. *CoRR*, abs/1408.0325, 2014.

[37] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[38] M. Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In *Proceedings of the Fourth*

ACM Conference on Recommender Systems, RecSys '10, pages 257–260, New York, NY, USA, 2010. ACM.

[39] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(6):721–741, Nov. 1984.

[40] J. Gemmell, T. Schimoler, B. Mobasher, and R. Burke. Hybrid tag recommendation for social annotation systems. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 829–838, New York, NY, USA, 2010. ACM.

[41] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 69–77, New York, NY, USA, 2011. ACM.

[42] A. Gershman, A. Meisels, K.-H. L uke, L. Rokach, A. Schclar, and A. Sturm. A decision tree based recommender system. In G. Eichler, P. G. Kropf, U. Lechner, P. Meesad, and H. Unger, editors, *IICS*, volume 165 of *LNI*, pages 170–179. GI, 2010.

[43] M. Ghazanfar and A. Prugel-Bennett. An improved switching hybrid recommender system using naive bayes classifier and collaborative filtering. In *The 2010 IAENG International Conference on Data Mining and Applications*, April 2010. Event Dates: 17-19 March, 2010.

[44] J. Golbeck. Trust and nuanced profile similarity in online social networks. *ACM Transactions on the Web (TWEB)*, 2009.

[45] N. Good, J. B. Schafer, J. A. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[46] O. Görlitz et al. PINTS: Peer-to-Peer Infrastructure for Tagging Systems. In *Intl Conf. on Peer-to-Peer Systems (IPTPS)*, 2008.

[47] M. Gueye, T. Abdessalem, and H. Naacke. A cluster-based matrix-factorization for online integration of new ratings. In *27-èmes journées Bases de Données Avancées (BDA'11)*, Rabat, Maroc, 2011.

[48] M. Gueye, T. Abdessalem, and H. Naacke. Factorisation multi-biais pour de meilleures recommandations. In *Actes du 5ème Colloque National sur la Recherche en Informatique et ses Applications (CNRIA'13)*, Ziguinchor, Sénégal, 2013.

[49] M. Gueye, T. Abdessalem, and H. Naacke. Foldcons: A simple way to improve tag recommendation. In *Proceedings of the Fifth ACM RecSys Workshop on Recommender Systems and the Social Web co-located with the 7th ACM Conference on Recommender Systems (RecSys 2013), Hong Kong, China, October 13, 2013.*, 2013.

[50] M. Gueye, T. Abdessalem, and H. Naacke. Strec: An improved graph-based tag recommender. In *Proceedings of the Fifth ACM RecSys Workshop on Recommender Systems and the Social Web co-located with the 7th ACM Conference on Recommender Systems (RecSys 2013), Hong Kong, China, October 13, 2013.*, 2013.

[51] M. Gueye, T. Abdessalem, and H. Naacke. Technique de factorisation multi-biais pour des recommandations dynamiques. In C. Vrain, A. Péninou, and F. Sèdes, editors, *Extraction et gestion des connaissances (EGC'2013), Actes, 29 janvier - 01 février 2013, Toulouse, France*, volume RNTI-E-24 of *Revue des Nouvelles Technologies de l'Information*, pages 365–376. Hermann-Éditions, 2013.

[52] M. Gueye, T. Abdessalem, and H. Naacke. A parameter-free algorithm for an optimized tag recommendation list size. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14. ACM, 2014.

[53] M. Gueye, T. Abdessalem, and H. Naacke. A social and popularity-based tag recommender. In *Proceedings of the 7th IEEE International Conference on Social Computing and Networking (SocialCom 2014). December 3-5, Sydney, Australia.* IEEE, 12 2014.

[54] M. Gueye, T. Abdessalem, and H. Naacke. Dynamic recommender system : using cluster-based biases to improve the accuracy of the predictions. In F. Guillet, G. Ritschard, D. Zighed, and H. Briand, editors, *Advances in Knowledge Discovery and Management*, Studies in Computational Intelligence. Springer Berlin Heidelberg, 2015.

[55] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web*, 2004.

[56] A. Gunawardana and C. Meek. A unified approach to building hybrid recommender systems. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 117–124, New York, NY, USA, 2009. ACM.

[57] A. Gunawardana and G. Shani. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res.*, 10:2935–2962, Dec. 2009.

[58] M. Gupta et al. Survey on social tagging techniques. *SIGKDD Explorations*, 12(1), 2010.

[59] M. R. Gupta and Y. Chen. Theory and use of the em algorithm. *Found. Trends Signal Process.*, 4(3):223–296, Mar. 2011.

[60] I. Guy, A. Jaimes, P. Agulló, P. Moore, P. Nandy, C. Nastar, and H. Schinzel. Will recommenders kill search?: Recommender systems - an industry perspective. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 7–12, New York, NY, USA, 2010. ACM.

[61] S. Hamouda and N. M. Wanas. Put-tag: personalized user-centric tag recommendation for social bookmarking systems. *Social Netw. Analys. Mining*, 1(4):377–385, 2011.

[62] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, 2004.

[63] Y. Hijikata, K. Iwahama, and S. Nishida. Content-based music filtering system with editable user profile. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 1050–1057, New York, NY, USA, 2006. ACM.

[64] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, Jan. 2004.

[65] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Folkrank: A ranking algorithm for folksonomies. In *Fachgruppe Information Retrieval (FGIR)*, 2006.

[66] D. Jannach and K. Hegelich. A case study on the effectiveness of recommendations in the mobile internet. In L. D. Bergman, A. Tuzhilin, R. D. Burke, A. Felfernig, and L. Schmidt-Thieme, editors, *RecSys*, pages 205–208. ACM, 2009.

[67] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, Oct. 2002.

[68] R. Jäschke et al. Tag recommendations in folksonomies. In *PKDD*, pages 506–514, 2007.

[69] R. Jäschke et al. Tag recommendations in social bookmarking systems. *AI Commun.*, 21(4):231–247, 2008.

[70] N. Koenigstein, G. Dror, and Y. Koren. Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy. In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys '11, pages 165–172, New York, NY, USA, 2011. ACM.

[71] J. Kogan. *Introduction to Clustering Large and High-Dimensional Data.* Cambridge University Press, New York, NY, USA, 2007.

[72] J. Kogan, C. Nicholas, and M. Teboulle. *Grouping Multidimensional Data: Recent Advances in Clustering.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[73] R. Kohavi, R. Longbotham, D. Sommerfield, and R. Henne. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 18(1):140–181, 2009.

[74] Y. Koren. How useful is a lower rmse?, 2007. Netflix Prize Forum.

[75] Y. Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, 2010.

[76] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, 2009.

[77] M. Kubatz, F. Gedikli, and D. Jannach. Localrank - neighborhood-based, fast computation of tag recommendations. In C. Huemer and T. Setzer, editors, *E-Commerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, pages 258–269. Springer Berlin Heidelberg, 2011.

[78] M. Kumar and N. Yadav. Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey. *Comput. Math. Appl.*, 62(10):3796–3811, Nov. 2011.

[79] L. G. Landau and J. G. Taylor. *Concepts for Neural Networks: A Survey.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1997.

[80] D. D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 4–15, London, UK, UK, 1998. Springer-Verlag.

[81] Y. Li and W. Ma. Applications of artificial neural networks in financial economics: A survey. In *Proceedings of the 2010 International Symposium on Computational Intelligence and Design - Volume 01*, ISCID '10, pages 211–214, Washington, DC, USA, 2010. IEEE Computer Society.

[82] G. Linden, B. Smith, and J. York. Industry report: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Distributed Systems Online*, 4(1), 2003.

[83] P. Lops, M. de Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In Ricci et al. [116], pages 73–105.

[84] L. Lü, M. Medo, C. H. Yeung, Y.-C. Zhang, Z.-K. Zhang, and T. Zhou. Recommender systems. *Physics Reports*, 519(1):1 – 49, 2012. Recommender Systems.

[85] X. Luo, Y. Xia, Q. Zhu, and Y. Li. Boosting the k-nearest-neighborhood based incremental collaborative filtering. *Knowledge-Based Systems*, 53(0):90 – 99, 2013.

[86] H. Ma et al. Recommender systems with social regularization. In *WSDM*, pages 287–296, 2011.

[87] H. Ma, M. R. Lyu, and I. King. Learning to recommend with trust and distrust relationships. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 189–196, New York, NY, USA, 2009. ACM.

[88] S. Maniu and B. Cautis. Network-aware search in collaborative tagging applications: Instance optimality versus efficiency. In *CIKM*, pages 939–948, 2013.

[89] S. Maniu, B. Cautis, and T. Abdessalem. Efficient top-k retrieval in online social tagging networks. *CoRR*, abs/1104.1605, 2012.

[90] L. Marinho et al. Relational classification for personalized tag recommendation. In *ECML PKDD Discovery Challenge*, 2009.

[91] L. Marinho et al. Social tagging recommender systems. In *Recommender Systems Handbook*. Springer, 2011.

[92] B. M. Marlin and R. S. Zemel. Collaborative prediction and ranking with non-random missing data. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 5–12, New York, NY, USA, 2009. ACM.

[93] P. Marx, T. Hennig-Thurau, and A. Marchand. Increasing consumers' understanding of recommender results: A preference-based hybrid algorithm with strong explanatory power. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 297–300, New York, NY, USA, 2010. ACM.

[94] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification, 1998.

[95] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM.

[96] M. McPherson, L. Smith-Lovin, and J. Cook. Birds of a feather: Homophily in social networks. *ANNUAL REVIEW OF SOCIOLOGY*, 27:415–444, 2001.

[97] A. K. Milicevic et al. Social tagging in recommender systems: a survey of the state-of-the-art and possible extensions. *Artif. Intell. Rev.*, 33(3):187–209, 2010.

[98] J. Mrosek et al. Content- and graph-based tag recommendation: Two variations. In *ECML PKDD Discovery Challenge*, 2009.

[99] K. Niemann and M. Wolpers. A new collaborative filtering approach for increasing the aggregate diversity of recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 955–963, New York, NY, USA, 2013. ACM.

[100] J. H. Paik, D. Pal, and S. K. Parui. A novel corpus-based stemming algorithm using co-occurrence statistics. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 863–872, New York, NY, USA, 2011. ACM.

[101] J. H. Paik, S. K. Parui, D. Pal, and S. E. Robertson. Effective and robust query-based stemming. *ACM Trans. Inf. Syst.*, 31(4):18:1–18:29, Nov. 2013.

[102] M. Papagelis, D. Plexousakis, and T. Kutsuras. Alleviating the sparsity problem of collaborative filtering using trust inferences. In *iTrust*. Springer, 2005.

[103] M. Papagelis, I. Rousidis, D. Plexousakis, and E. Theoharopoulos. Incremental collaborative filtering for highly-scalable recommendation algorithms. In M.-S. Hacid, N. Murray, Z. Raś, and S. Tsumoto, editors, *Foundations of Intelligent Systems*,

volume 3488 of *Lecture Notes in Computer Science*, pages 553–561. Springer Berlin Heidelberg, 2005.

[104] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, 2007.

[105] M. Pazzani and D. Billsus. Content-based recommendation systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 325–341. Springer Berlin Heidelberg, 2007.

[106] M. J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.*, 13(5-6):393–408, Dec. 1999.

[107] M. F. Porter. Readings in information retrieval. In K. Sparck Jones and P. Willett, editors, *Readings in Information Retrieval*, chapter An Algorithm for Suffix Stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[108] M. N. Postorino and G. M. L. Sarne. A neural network hybrid recommender system. In *Proceedings of the 2011 Conference on Neural Nets WIRN10: Proceedings of the 20th Italian Workshop on Neural Nets*, pages 180–187, Amsterdam, The Netherlands, The Netherlands, 2011. IOS Press.

[109] S. Ray and A. Mahanti. Weighted class based hybrid algorithm for top-n recommender systems. In *Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications*, AIA '08, pages 245–251, Anaheim, CA, USA, 2008. ACTA Press.

[110] B. Recht, C. Re, S. J. Wright, and F. Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In Shawe-Taylor et al. [125], pages 693–701.

[111] B. Recht and C. Recht. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, 2013.

[112] S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In P. Pu, D. G. Bridge, B. Mobasher, and F. Ricci, editors, *RecSys*, pages 251–258. ACM, 2008.

[113] S. Rendle and L. Schmidt-Thieme. Factor models for tag recommendation in bibsonomy. In *ECML PKDD Discovery Challenge*, Bled, Slovenia, 2009.

[114] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM*, pages 81–90, 2010.

[115] F. Ricci. Mobile recommender systems. *Information Technology & Tourism*, 12(3):205–231, 2010.

[116] F. Ricci et al., editors. *Recommender Systems Handbook*. Springer, 2011.

[117] F. Rosenblatt. Neurocomputing: Foundations of research. In J. A. Anderson and E. Rosenfeld, editors, *Neurocomputing: Foundations of Research*, chapter The Perception: A Probabilistic Model for Information Storage and Organization in the Brain, pages 89–114. MIT Press, Cambridge, MA, USA, 1988.

[118] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[119] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[120] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proceedings of the 5th International Conference in Computers and Information Technology*, 2002.

[121] J. B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[122] C. E. Seminario. Accuracy and robustness impacts of power user attacks on collaborative recommender systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 447–450, New York, NY, USA, 2013. ACM.

[123] G. Shani and A. Gunawardana. Evaluating recommendation systems. In Ricci et al. [116], pages 257–297.

[124] A. Sharma and D. Cosley. Do social explanations work?: Studying and modeling the effects of social explanations in recommender systems. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1133–1144, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[125] J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, editors. *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, 2011.

[126] R. Sinha and K. Swearingen. The role of transparency in recommender systems. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '02, pages 830–831, New York, NY, USA, 2002. ACM.

[127] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, 2009.

[128] Y. Sun, G. Liu, and K. Xu. A k-means-based projected clustering algorithm. In *Proceedings of the 2010 Third International Joint Conference on Computational*

Science and Optimization - Volume 01, CSO '10, pages 466–470, Washington, DC, USA, 2010. IEEE Computer Society.

[129] N. Sundaresan. Recommender systems at the long tail. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 1–6, New York, NY, USA, 2011. ACM.

[130] M. Tahajod et al. Trust management for semantic web. In *ICCEE*, pages 3–6, 2009.

[131] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Major components of the gravity recommendation system. *SIGKDD Explor. Newsl.*, 9:80–83, 2007.

[132] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, NETFLIX '08, pages 6:1–6:8, New York, NY, USA, 2008. ACM.

[133] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, 2009.

[134] M. Tavakolifard. Similarity-based techniques for trust management. *Web Intelligence and Intelligent Agents*, 2010.

[135] N. Tintarev. Explanations of recommendations. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, pages 203–206, New York, NY, USA, 2007. ACM.

[136] TPC-Council. Tpc benchmark c, rev 5.11. Technical report, Transaction Processing Performance Council, 2010.

[137] S. Vargas and P. Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 109–116, New York, NY, USA, 2011. ACM.

[138] S. Vembu and S. Baumann. A self-organizing map based knowledge discovery for music recommendation systems. In U. Wiil, editor, *Computer Music Modeling and Retrieval*, volume 3310 of *Lecture Notes in Computer Science*, pages 119–129. Springer Berlin Heidelberg, 2005.

[139] P. Victor, N. Verbiest, C. Cornelis, and M. D. Cock. Enhancing the trust-based recommendation process with explicit distrust. *ACM Trans. Web*, 7(2):6:1–6:19, May 2013.

[140] J. Vig, S. Sen, and J. Riedl. Tagsplanations: Explaining recommendations using tags. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, pages 47–56, New York, NY, USA, 2009. ACM.

[141] D. C. Wilson and C. E. Seminario. When power users attack: Assessing impacts in collaborative recommender systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 427–430, New York, NY, USA, 2013. ACM.

[142] H. Wu and H. Fang. Relation based term weighting regularization. In *Proceedings of the 34th European Conference on Advances in Information Retrieval*, ECIR'12, pages 109–120, Berlin, Heidelberg, 2012. Springer-Verlag.

[143] H. Wu, Y. Wang, and X. Cheng. Incremental probabilistic latent semantic analysis for automatic question recommendation. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 99–106, New York, NY, USA, 2008. ACM.

[144] X. Yang, Y. Guo, Y. Liu, and H. Steck. A survey of collaborative filtering based social recommender systems. *Comput. Commun.*, 41:1–10, Mar. 2014.

[145] X. Yang, Z. Zhang, and K. Wang. Scalable collaborative filtering using incremental update and local link prediction. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 2371–2374, New York, NY, USA, 2012. ACM.

[146] H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*, pages 1–27, 2013.

[147] M. Zanker. The influence of knowledgeable explanations on users' perception of a recommender system. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 269–272, New York, NY, USA, 2012. ACM.

[148] F.-G. Zhang. Preventing recommendation attack in trust-based recommender systems. *J. Comput. Sci. Technol.*, 26(5):823–828, Sept. 2011.

[149] G. P. Zhang. Neural networks for classification: A survey. *Trans. Sys. Man Cyber Part C*, 30(4):451–462, Nov. 2000.

[150] Y. Zhang, N. Zhang, and J. Tang. A collaborative filtering tag recommendation system based on graph. In *ECML PKDD Discovery Challenge*, Bled, Slovenia, 2009.

[151] Y.-C. Zhang, M. Blattner, and Y.-K. Yu. Heat conduction process on community networks as a recommendation model. *Physical Review Letters*, 99(15):154301, 2007.

[152] Y.-C. Zhang, M. Medo, J. Ren, T. Zhou, T. Li, and F. Yang. Recommendation model based on opinion diffusion. *EPL (Europhysics Letters)*, 80(6):68003, 2007.

[153] T. Zhou, Z. Kuscsik, J.-G. Liu, M. Medo, J. R. Wakeling, and Y.-C. Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

[154] T. Zhou, J. Ren, M. c. v. Medo, and Y.-C. Zhang. Bipartite network projection and personal recommendation. *Phys. Rev. E*, 76:046115, Oct 2007.

[155] W. Zhou, Y. S. Koh, J. Wen, S. Alam, and G. Dobbie. Detection of abnormal profiles on group attacks in recommender systems. In *Proceedings of the 37th International ACM SIGIR Conference on Research &#38; Development in Information Retrieval*, SIGIR '14, pages 955–958, New York, NY, USA, 2014. ACM.

[156] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 249–256, New York, NY, USA, 2013. ACM.

[157] C.-N. Ziegler and J. Golbeck. Investigating interactions of trust and interest similarity. *Decision Support Systems*, 2007.

[158] C.-N. Ziegler and G. Lausen. Propagation models for trust and distrust in social networks. *Information Systems Frontiers*, 7(4-5):337–358, 2005.

[159] C.-N. Ziegler, G. Lausen, and J. A. Konstan. On exploiting classification taxonomies in recommender systems. *AI Commun.*, 21(2-3):97–125, Apr. 2008.

[160] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM.

[161] M. Zinkevich, A. J. Smola, and J. Langford. Slow learners are fast. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 2331–2339. Curran Associates, Inc., 2009.

[162] J. Zou and F. Fekri. A belief propagation approach for detecting shilling attacks in collaborative filtering. In *Proceedings of the 22Nd ACM International Conference on Conference on Information &#38; Knowledge Management*, CIKM '13, pages 1837–1840, New York, NY, USA, 2013. ACM.

Appendices

A | Résumé en Français

La recherche d'information dans le contexte de l'augmentation du nombre des ressources sur Internet demeure un défi à relever. Les utilisateurs doivent souvent choisir parmi un très grand nombre de ressources. De ce fait, ils rencontrent beaucoup de difficultés pour prendre une décision appropriée. Par exemple, dans les moteurs de recherche comme Google, Yahoo et autre, un utilisateur formule son besoin par une requête, en utilisant des mots-clés qui seront comparés avec le contenu des ressources (c-à-d., les documents web, dans ce cas). Le résultat retourné à l'utilisateur contient souvent un grand nombre de documents dont l'ordre de tri n'est pas forcément pertinent pour lui qui est ainsi obligé de sélectionner manuellement les documents l'intéressant le plus. Ce qui peut être perçu comme une tâche pénible et ennuyeuse pour l'utilisateur.

Les utilisateurs étant souvent submergés par le nombre d'objetsqu'ils peuvent choisir, ils peinent à trouver les objets qui les intéressent et finissent souvent désorientés et déconcertés. C'est le phénomène de surcharge d'information ! Ce problème de surcharge d'information peut être pallié par la personnalisation de l'accès aux informations [60] : c'est tout l'intérêt de la recommandation de contenu.

La recommandation, dans ses principes élémentaires, se base sur les antécédents des utilisateurs (lectures, achats, consultations, etc.). Elle peut utiliser des profils représentatifs des intérêts, relativement stables, des utilisateurs (filtrage de contenu [105, 83]) [105, 63] ou les notes d'un groupe d'utilisateurs pour faire une suggestion à un autre (filtrage collaboratif [127, 76, 70, 86]) [127]. Ce dernier est le plus utilisé. Cependant on note assez souvent des approches mixtes [22, 56, 43, 28, 109].

L'objectif des systèmes de recommandation (*recommander Systems* en anglais, *RS* en abrégé) est de déterminer, parmi une grande quantité de contenu lesquels intéresseront le plus un utilisateur donné (les objets, livres, films, etc. qui sont susceptibles de l'intéresser, les utilisateurs d'un réseau social avec qui il pourrait tisser des liens, etc.). Ils ont une valeur commerciale capitale pour tout type de e-commerce [121, 35, 66]. Tout en subvenant au problème de surcharge d'informations et donc en assurant d'une certaine manière la satisfaction et fidélité des clients, un RS vise à assurer plus de profit à tout commerce ou fournisseur de données l'utilisant. D'après le directeur marketing de la société Strands recommander [1], un leader dans la mise en place de systèmes de recommandation, la recommandation participerait en moyenne à hauteur de 8 à 12%

1. `http://bit.ly/1rpbM5L`

dans les ventes d'un site commercial. Et d'après certains dires, 35% des ventes du leader du e-commerce qu'est Amazon sont dues à son système de recommandation[2] [3]. La recommandation représente des dizaines de milliards de dollars de chiffre d'affaire. Tous ces chiffres expliquent l'engouement actuel autour de ces systèmes.

Un RS peut être évalué sur la qualité de ses recommandations. Plus ces dernières sont intéressantes pour les utilisateurs, plus elles sont de qualité. Cependant, la qualité n'est pas le seul critère pour évaluer un système de recommandation. Le temps de traitement nécessaire pour fournir des suggestions aux utilisateurs est un second critère à prendre en compte. En effet, un RS doit traiter des volumes de données très importants et qui augmentent continuellement, ce qui peut poser des problèmes de passage à l'échelle.

Dans cette thèse, nous nous sommes consacrés au développement d'algorithmes de recommandation offrant de bonne qualité de recommandation, tout en étant capables de s'adapter au volume de données à prendre en charge. Pour cette tâche, principalement deux directions peuvent être suivies : d'une part, l'optimisation d'algorithmes existants, et d'autre part, la réduction du temps de calcul en adaptant ces algorithmes à des infrastructures distribuées (ex : de type Cloud) offrant une capacité de traitement extensible.

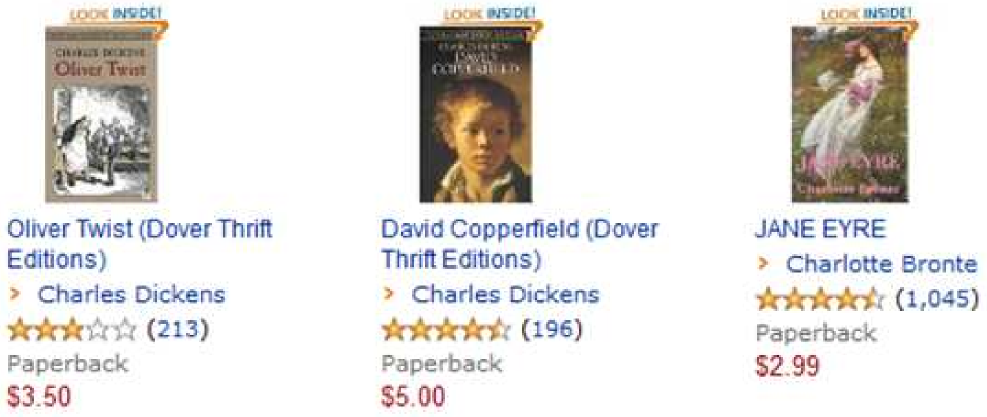
Nous avons étudié et fait des propositions dans deux contextes de recommandation, à savoir la recommandation de contenu noté par les utilisateurs et la recommandation de tags (c-à-d., la suggestion de termes pour annoter un objet donné). Les sections qui suivent décrivent chacun de ces deux contextes et présentent les approches que nous avons proposées afin d'optimiser le temps de calcul tout en assurant une bonne qualité de recommandation.

Factorisation multi-biaisée pour des recommandations dynamiques

Dans le cadre de la recommandation, un RS a besoin d'estimer l'intérêt qu'un utilisateur aurait sur un objet afin de lui proposer les objets les plus susceptibles de l'intéresser. Pour ce faire, les systèmes de recommandations ont besoin de savoir les notes que les utilisateurs ont données aux objets qu'ils connaissent déjà. La plupart du temps, ces notes (ou intérets) des utilisateurs sur les objets sont récupérés sous forme de valeurs numériques (ex : 1 à 5 étoiles comme sur beaucoup de sites web). La figure A.1 montre le *widget* d'Amazon qui permet à ses utilisateurs de donner leur notes sur ses objets. Plus la valeur donnée est élevée, plus l'utilisateur s'intéresse à l'objet.

Considèrons un ensemble d'utilisateurs U, un ensemble de objets I et une liste de notes (u, i, r_{ui}, t_{ui}) où chaque valeur r_{ui} représente l'intérêt de l'utilisateur u pour le objet i, t_{ui} étant le moment où la note a été soumise, la recommandation repose sur la prédiction des futures notes des utilisateurs telles que l'écart entre une note prédite $f(u, i)$

2. http://bit.ly/1tFcThk
3. http://glinden.blogspot.fr/2006/12/35-of-sales-from-recommendations.html



Oliver Twist (Dover Thrift Editions)
› Charles Dickens
★★★☆☆ (213)
Paperback
$3.50

David Copperfield (Dover Thrift Editions)
› Charles Dickens
★★★★½ (196)
Paperback
$5.00

JANE EYRE
› Charlotte Bronte
★★★★½ (1,045)
Paperback
$2.99

FIGURE A.1 – Widget à 5 étoiles sur Amazon.com

et celle réellement donnée ultérieurement r_{ui}, soit le plus petit possible. Cela permet de proposer à l'utilisateur les objets présentant les plus grandes valeurs de prédiction. Ainsi, la qualité d'un système de recommandation peut-être rattachée à la précision de ses prédictions. En pratique, pour estimer cette précision (c-à-d les écarts), l'ensemble des notes existantes est subdivisé en deux parties : la plus grande pour l'apprentissage et la seconde pour l'évaluation. La mesure appelée RMSE est l'une des plus utilisées pour l'évaluation. RMSE est la racine carré de la moyenne des carrés des écarts [62, 127]. Nous l'avons utilisée dans nos experiences.

$$RMSE = \sqrt{\frac{1}{n}\sum_{u,i}(r_{ui} - f(u,i))^2} \tag{A.1}$$

n représente le nombre total de notes à prédire. Plus petit est le RMSE, meilleures sont les prédictions.

La factorisation de matrices (FM) est une technique de filtrage collaboratif apportant une qualité très satisfaisante [127, 76, 104, 133, 76]. Elle consiste à construire des profils caractérisant les utilisateurs et les objets, au moyen de vecteurs de facteurs. Ces profils sont déduits des notes que les utilisateurs attribuent aux objets. Ainsi, il est possible d'estimer l'intérêt d'un utilisateur pour un objet en combinant le profil de l'utilisateur avec celui de l'objet. Le produit scalaire est généralement utilisé. Puis, les objets avec les estimations les plus grandes sont recommandés.

Les systèmes de recommandation utilisant la factorisation de matrices représentent, le plus souvent, les notes des utilisateurs dans une matrice R creuse. Les colonnes représentent les utilisateurs et les lignes les objets. Ainsi la note $r_{ui} \in R$ est celle donnée par l'utilisateur u à l'objet i. R est généralement très creuse.
L'objectif de la factorisation est de prédire les valeurs manquantes dans R. Dans sa forme

basique (FM basique), elle cherche à approximer R comme le produit de deux autres matrices

$$R = P \cdot Q \tag{A.2}$$

Les deux matrices P et Q contiennent respectivement les vecteurs de facteurs représentatifs des utilisateurs et ceux des objets. Ce sont les matrices de facteurs. Pour prédire la note $f(u,i)$ que l'utilisateur u donnerait à l'objet i, il suffit simplement d'appliquer la formule

$$f(u,i) = p_u \cdot q_i^T \tag{A.3}$$

p_u et q_i étant respectivement les vecteurs de facteurs de l'utilisateur u et de l'objet i dans P et Q.

Le processus d'apprentissage qu'effectue la factorisation détermine les valeurs dans P et Q telles qu'on s'approche le plus des notes r_{ui} existantes dans R. Il utilise une descente de gradient stochastique (DGS) qui calcule un minimum local tel que la somme des erreurs (c-à-d des écarts), $e_{ui} \overset{def}{=} r_{ui} - f(u,i)$ entre les notes prédites $f(u,i)$ et celles réelles r_{ui} données par les utilisateurs, soit la plus faible possible. DGS minimise la somme des erreurs quadratiques $\sum\limits_{ui} e_{ui}^2$ en ajustant les facteurs dans P et Q jusqu'à ce que cette somme ne diminue plus :

$$\begin{aligned} p_{uk} &\leftarrow p_{uk} + \lambda \cdot (2 \cdot e_{ui} \cdot q_{ki} - \beta \cdot p_{uk}) \\ q_{ki} &\leftarrow q_{ki} + \lambda \cdot (2 \cdot e_{ui} \cdot p_{uk} - \beta \cdot q_{ki}) \end{aligned} \tag{A.4}$$

Ceci permet de diminuer les erreurs et par conséquent d'avoir une meilleure approximation des notes réelles. Le paramètre λ introduit dans l'ajustement des facteurs représente un taux d'apprentissage. β est un paramètre de régularisation.

Après cette phase, les prédictions $f(u,i)$ sont calculées comme les objets $p_u \cdot q_i^T$. Un tri est effectué par la suite pour trouver les objets les plus intéressants (ceux avec les plus grandes notes de prédiction) et les recommander à l'utilisateur concerné.

L'une des améliorations en termes de qualité de la FM basique suppose que la plupart des variations observées sur les notes des utilisateurs sont dues principalement à des effets associés soit aux utilisateurs, soit aux objets [133, 76, 104]. Autrement dit, certains utilisateurs ont tendance à donner des notes plus élevées ou plus faibles que les autres utilisateurs. Et certains objets aussi sont plus ou moins appréciés que les autres. La factorisation basique (FM basique) présentée précédemment ne prend pas en compte ces tendances. La factorisation biaisée de matrices (FBM) introduit des biais pour tenir en compte ces variations de notation. Les biais reflètent les tendances des utilisateurs et des objets. On a la formule de prédiction suivante :

$$f(u,i) = p_u \cdot q_i^T + \mu + b_u + b_i \tag{A.5}$$

où μ dénote la moyenne de toutes les notes confondues, b_u et b_i sont respectivement le biais de l'utilisateur et celui de l'objet (i.e, la tendance de l'utilisateur et la perception de l'objet par rapport à la moyenne). Une bonne approximation de ces biais est cruciale pour avoir des prédictions de bonne qualité [104, 76]. Ainsi, ils doivent être ajustés durant la phase d'apprentissage en utilisant

$$\begin{aligned} b_i &\leftarrow b_i + \lambda \cdot (2 \cdot e_{ui} - \gamma \cdot b_i) \\ b_u &\leftarrow b_u + \lambda \cdot (2 \cdot e_{ui} - \gamma \cdot b_u) \end{aligned} \tag{A.6}$$

γ est un paramètre de régularisation. Il joue un rôle comparable à β dans l'équation A.4.

Bien que très utilisée, la factorisation présente des limites. Un inconvénient majeur est que le modèle résultant de la factorisation, reste statique. Le modèle ne tient pas compte des nouvelles notes que les utilisateurs produisent continuellement. Ces nouvelles notes ne seront prises en compte qu'à une prochaine factorisation. Ainsi, le modèle a besoin d'être régénéré fréquemment. Bien qu'il existe des approches pour paralléliser le calcul [110, 41, 111], ceci n'est pas toujours possible à cause du coût prohibitif de la factorisation. De ce fait, la qualité des recommandations décroît graduellement entre deux générations du modèle.

Nos travaux considèrent les contextes dynamiques où de nouvelles notes sont continuellement soumises. Dans de tels contextes, il n'est pas possible d'avoir un modèle à jour à cause du temps nécessaire pour le calculer. Au minimum, les notes soumises durant la génération d'un modèle ne sont pas prises en compte. Après la génération d'un modèle, la situation peut se dégrader assez rapidement puisque le nombre de notes non prises en compte augmente rapidement. De ce fait, une perte de qualité grandissante peut être observée dans les recommandations aussi longtemps qu'un nouveau modèle n'est pas généré (ce que nous démontrons dans nos expérimentations). Pour y faire face, nous proposons un modèle combinant des biais globaux à des biais locaux.

Nous nous basons sur l'observation que beaucoup d'utilisateurs tendent à sur-apprécier ou sous-apprécier les objets qu'ils notent. Une manière simple de quantifier cette tendance est d'assigner un biais global à chaque utilisateur comme avec la FBM [76, 104, 132]. Cependant, la tendance d'un utilisateur n'est généralement pas uniforme : elle peut changer d'un groupe de objets à un autre. Pour certains groupes de objets, un utilisateur peut avoir tendance à noter comme tout le monde alors qu'il surestime ou sous-estime d'autres groupes par manque d'objectivité. Cette tendance devient uniforme pour un ensemble de objets similaires ; ce que nous avons formalisé dans nos travaux [51, 54] à travers l'équation bornant la variance de notation des utilisateurs par la dissimilarité des objets notés :

$$0 \leq \sum_{u \in U} Var_u \leq \left(\sum_{(i,j) \in I^2} dissim_{ij} \right)^2 \tag{A.7}$$

Pour prendre en compte cette diversité de notation, nous attribuons un biais local $\delta_u^{c(i)}$ à chaque utilisateur u pour chaque groupe C de objets similaires ou proches. Cette multitude de biais par utilisateur permet d'avoir un modèle plus raffiné et vise à une meilleure qualité de recommandation. Nous appelons ce modèle par CBMF et sa formule de prédiction est la suivante

$$f(u,i) = p_u \cdot q_i^T + \mu + \left(b_u + \frac{1}{|\varsigma_i|} \sum_{C \in \varsigma_i} b_u^C \right) + b_i \tag{A.8}$$

où ς_i représente la liste des groupes auxquels appartient l'objet i. Ce modèle permet de prendre les cas où objet pourrait convenir à plusieurs groupes ou catégories. C'est l'exemple d'un film catalogué sous plusieurs genres. Nous affinons les biais locaux b_u^C durant la phase d'apprentissage.

Les biais locaux étant calculés sur des groupes d'objets similaires, leur coût de calcul faible permet de les ajuster à la volée lorsque de nouvelles notes arrivent. Ils assurent ainsi la robustesse du modèle dans un contexte dynamique en maintenant une meilleure qualité dans le temps.

Nos expérimentations sont faites sur les jeux de données de Netflix [15] et MovieLens [4] dont les tailles respectives sont de 100 et 10 millions de notes . Ces jeux de données sont très utilisés dans la littérature [127].

Nos résultats montrent que notre modèle est plus performant que la FBM et la FM basique. CBMF donne de plus petites erreurs dans ses prédictions. La table A.1 présente les différents résultats que nous avons obtenus sur les deux jeux de données. Nous démontrons aussi la perte de qualité encourue dans un contexte où le modèle n'est

Jeu de données	FM basique	FBM	CBMF
Movielens	0,7743	0,7608	**0,7578**
Netflix	0,9599	0,9312	**0,9208**

Table A.1 – RMSE des trois modèles (FM basique, FBM et CBMF).

pas mis à jour et que de nouvelles notes arrivent en continue. Pour cela, nous simulons un cadre réel où les utilisateurs soumettent une quantité importante de notes (i.e. plusieurs millions de notes par jour). La compagnie Netflix, par exemple, reçoit jusqu'à 4M de notes par jour [8]. Pour mettre en place cette situation, l'utilisation d'un jeu de données de grande taille s'impose et seul celui de Netflix contient suffisamment de notes. Le jeu de données de MovieLens étant petit.

Nous observons l'impact global des nouvelles notes provenant de tous les utilisateurs sans exception. Ainsi, nous construisons le jeu de test de telle sorte que chaque utilisateur y soit présent. Le jeu de test contient 10% des notes les plus récentes de chaque utilisateur, le reste (c-à-d., les 90%) servant à l'apprentissage. Plus précisément, nous alignons dans l'échantillon de test la séquence d'arrivée des notes de sorte qu'à la date D_i, i notes sont déjà arrivées pour chaque utilisateur.

Nous mesurons ensuite l'évolution de la qualité des prédictions pour des dates D_i successives lorsqu'on progresse à travers le jeu de test. Nous utilisons une fenêtre glissante de 200.000 notes (dont la moitié est partagée avec la fenêtre précédente afin de lisser les résultats). La figure A.2 montre l'évolution de la qualité pour les trois modèles : MF basique, FBM et CBMF. L'erreur de prédiction (i.e., le RMSE) augmente, ce qui confirme la perte de qualité au fil du temps. Nous observons une augmentation de 5% du RMSE lorsque 5M à 7M de nouvelles notes n'ont pas été prises en compte. En pratique, cela signifie que les trois modèles deviennent rapidement obsolètes.

4. `http://www.grouplens.org/node/73`

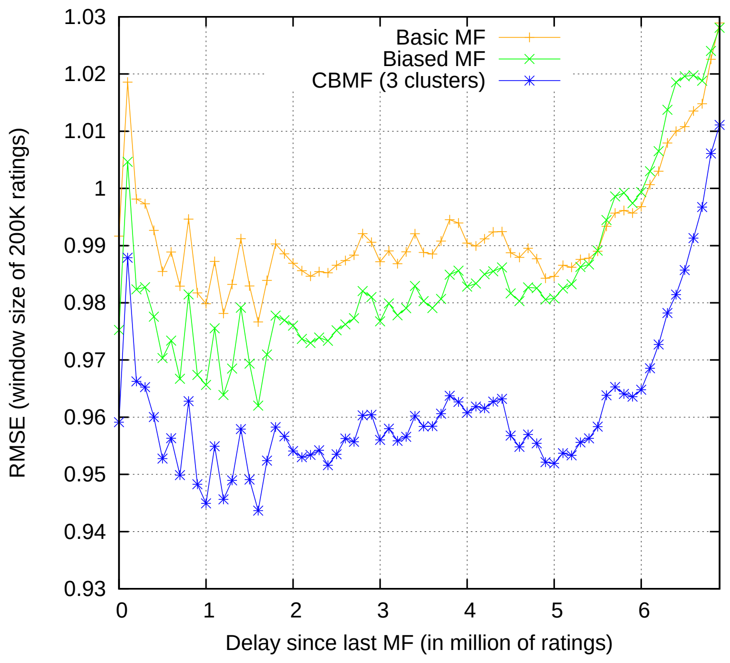


FIGURE A.2 – Croissance de l'erreur RMSE lorsque le nombre de nouvelles notes non prises en compte augmente.

Nous montrons aussi la robustesse de notre modèle dans le temps. Autrement dit, qu'il maintient une bonne qualité dans le temps. Utilisant toujours les jeux d'apprentissage et de test de l'expérimentation précédente, nous prenons en compte l'intégration en ligne des nouvelles notes des utilisateurs en ajustant leurs biais locaux.

Nous parcourons une à une les nouvelles notes dans le jeu de test. Chaque nouvelle note est comparée par rapport à la prédiction qui aurait été faite (on calcul l'écart), puis elle est automatiquement intégrée afin d'améliorer les futures prédictions. Le temps moyen d'intégration est de 1,24 millisecondes. L'intégration est rapide et ne constitue qu'un léger calcul.

La figure A.3 présente la nouvelle évolution de la qualité des prédictions pour le modèle FMBM lorsqu'on intègre les nouvelles notes des utilisateurs. On y voit l'importance de la prise en compte de ces nouvelles notes avec un bénéfice de 13,97% lorsqu'on atteint 7M de nouvelles notes intégrées. C'est une amélioration significative pour des recommandations. Ce qui prouve que notre solution est robuste. Nous avons effectué des expériences supplémentaires démontrant le bénéfice de mettre à jour les biais plutôt que les facteurs. Nous renvoyons le lecteur au chapitre 3.

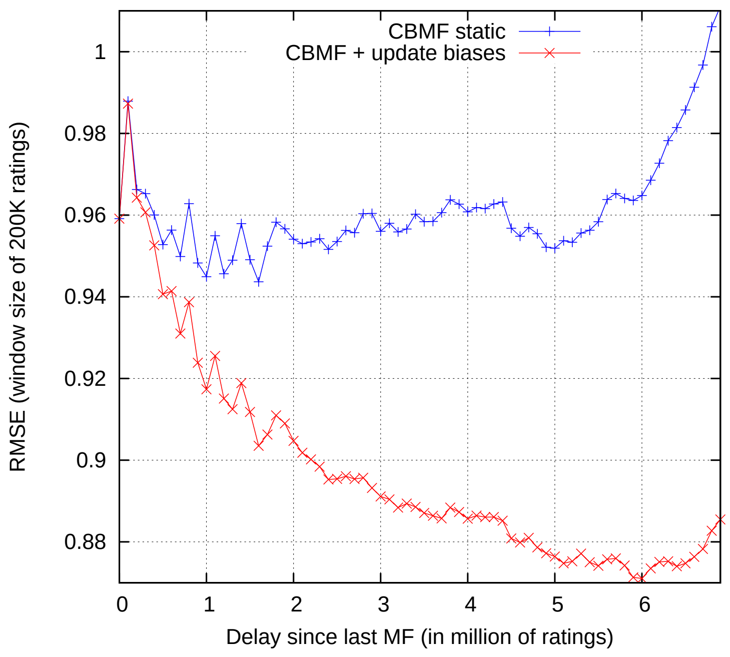


Figure A.3 – Évolution du RMSE avec l'intégration en ligne

Une méthode de voisinage étendu et optimisé pour la recommandation de tags

En deuxième partie de nos travaux, nous nous sommes intéressés à la recommandation de tags. Il s'agit de la suggestion aux utilisateurs de mots-clés à utiliser lorsqu'ils veulent annoter une ressource tel que cela est proposé dans Delicious[5]ou sur Bibsonomy[6].

Nous nous sommes concentré sur le célèbre algorithme des plus proches voisins (*K-Nearest neighbors* en anglais, *KNN* en abrégé) dans le cadre de l'utilisation des réseaux sociaux pour améliorer la qualité des recommandations. Un défaut de cet algorithme tel qu'utilisé est que le nombre de voisins à considérer est fixé à priori, ou tout au plus limité au voisinage direct. Ceci à pour conséquence que la qualité des recommandation n'est pas constante car elle est dépendante du nombre de voisins utilisé.

Nous proposons une adaptation de la méthode d'optimisation de Fagin et al [33] à l'algorithme KNN, dans le cadre de la recommandation de tags. A travers cette adaptation, il nous est possible de ne plus fixer le nombre de voisins à consulter en vue de faire des suggestions. Nous choisissons dynamiquement le nombre optimal de voisins à considérer. De plus, notre approche prend en compte les voisins éloignés (c-à-d., indirects) en

5. `http://www.delicious.com`
6. `http://www.bibsonomy.com`

étendant la relation de proximité entre les utilisateurs. Si un utilisateur u est connecté à un utilisateur v avec une valeur de proximité $\theta^{+}(u,v)$ et que ce dernier à une proximité $\theta^{+}(v,z)$ avec un autre utilisateur z, nous pouvons inférer, par transition, une proximité $\theta^{+}(u,z)$ entre u et z en utilisant la multiplication des deux précédentes proximités. Nous nous basons sur les études effectuées dans ce sens et accréditant cette possibilité [130, 32].

Pour recommander des tags à un utilisateur u pour annoter une ressource i, notre algorithme combine les popularités des tags avec les avis du voisinage de u. Il assigne à chaque tag t deux scores :

– un score courant, $score(t|u,i)$, tenu à jour durant la navigation dans le graphe d'utilisateurs (ex. un réseau social). Nous définissons ce score comme suit :

$$score(t|u,i) = \Big(\alpha \times \rho(t,i) + (1-\alpha) \times \rho(t,u)\Big) \times \Big(1 + \eta(t|u,i)\Big)$$

avec $\rho(t,i)$ et $\rho(t,u)$ étant les pourcentages d'utilisation du tag t respectivement pour la ressource i et par l'utilisateur u. $\eta(t|u,i)$ représente l'avis du voisinage de u sur l'annotation de la ressource i avec le tag t. Nous le prenons comme le rapport du nombre de ses voisins qui ont annoté i par t et le nombre total de personnes ayant annoté i.

– et un score maximal, $MaxScore(t|u,i)$, que le tag ne peut dépasser. Le score maximale prend en compte la probable contribution des voisins ayant annoté la ressource et qui ne sont pas encore visités durant le parcours du graphe d'utilisateurs.

En plus de ces deux valeurs de score, un score maximal et global $MaxScoreUnseen$ est attribué à tous les tags non encore rencontrés durant le parcours du graphe.
Durant toute l'exécution, une liste D des k tags avec les plus grands scores est maintenue ordonnée et mise à jour. Nous arrêtons le parcours lorsque le plus faible des scores courants des tags de la liste D est supérieur aux scores maximaux $MaxScore(t|u,i)$ de tous les autres tags qui n'y sont pas encore et aussi celui des tags qu'on n'a pas encore rencontrés, $MaxScoreUnseen$. Ceci nous permet de borner le temps d'exécution tout en optimisant la qualité finale des recommandations.

Nous avons expérimenté notre proposition sur 6 jeux de données disponibles et largement connus de la communauté. Nous avons mené une série d'expériences montrant les gains en qualité et l'adaptation de notre proposition à de larges graphes d'utilisateurs. Plus de détails sont disponible au chapitre 4.

Optimisation du nombre d'objets à recommander

En troisième partie de nos travaux, nous nous sommes exclusivement focalisés sur l'amélioration de la qualité des recommandations, spécialement dans le cadre de la recommandation de tags. La recommandation de liste d'objets est courante sur le web, elle consiste à proposer une liste classée d'objets comme nous l'avons précédemment vu.

Généralement la taille des listes proposées est constante quel que soit l'utilisateur à qui ont fait la recommandation. Même si certains objets peuvent ne pas être pertinents pour l'utilisateur, ils sont inclus dans la liste afin de respecter une taille fixée à priori. Ceci peut avoir un mauvais impact sur la qualité des listes recommandées.

Dans nos expériences, nous montrons l'effet négatif qui peut découler de la fixation de la taille des listes de recommandations. Nous avons utilisé quatre algorithmes de recommandation de tags sur cinq jeux de données différents. La figure A.4 rapporte l'évolution moyenne de la qualité des listes recommandées, en termes de valeurs *F1*, pour des tailles de liste allant de 1 à 10. Elle démontre l'existence d'une taille de liste optimale dépendant des jeux de données. Nous voyons aussi que de longues listes n'apportent pas forcément de meilleures recommandations. D'où l'intérêt de pouvoir personnaliser la taille des listes de recommandations selon le contexte de la recommandation, c-à-d., en prenant en compte l'utilisateur à qui on souhaiterait faire la recommandation et la ressource qu'il voudrait annoter.

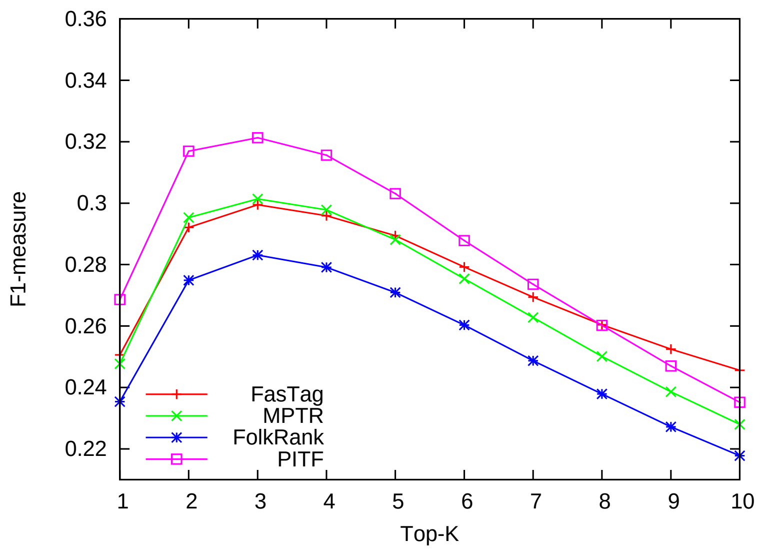


FIGURE A.4 – Pertinence relative des tags en fonction de leurs rangs et du premier

Nous proposons (voir le chapitre 5) une méthode permettant de déterminer une taille de liste optimale à utiliser lors d'une recommandation. Autrement dit, étant donnée une liste L_N de tags recommandés, nous souhaitons trouver parmi toutes ses sous-listes, celle qui optimiserait la plus la qualité globale de la recommandation comme illustré ci-dessous :

$$\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{t_1}_{L_1}\quad t_2}_{L_2}\quad t_3}_{\dots}\quad \dots}_{L_{N-2}}\quad t_{N-1}}_{L_{N-1}}\quad t_N}_{L_N}$$

Plus formellement, nous introduisons une mesure de pertinence $Rel(L_N|u,i)$ d'une liste L_N de recommandations qui estime la probabilité que l'utilisateur u prenne tous les tags de la liste pour annoter la ressource i. Nous calculons la sous-liste offrant la meilleure pertinence et considérons sa taille – que nous notons par *bls*[7] – comme celle optimale à prendre.

$$bls = \max\left(s \mid s \in \mathcal{S}\right) \tag{A.9}$$

avec

$$\mathcal{S} = \Big\{ s \;\Big|\; s \leq N \wedge \forall n \leq N,\ Rel(L_n|u,i) \leq Rel(L_s|u,i) \Big\}$$

Notre approche fonctionne comme un module indépendant qu'on peut utiliser avec n'importe quel algorithme de recommandation de tags. Il reçoit en entrée une liste de recommandations dont il optimise la taille en enlevant les tags qu'il juge impertinents. Ceci à un double avantage. Premièrement, on peut espérer plus de confiance chez l'utilisateur à travers sa satisfaction tout en ne le submergeant pas de propositions inutiles. Deuxièment, l'espace des tags enlévés peut être occupé par d'autres suggérés par un autre système de recommandation ayant un fonctionnement différent, afin d'avoir une plus grande diversité dans les recommandations.

Nous définissons d'abord la pertinence $Rel(t|u,i)$ d'un tag t du point de vue d'un utilisateur u qui voudrait annoter une ressource i comme suit :

$$Rel(t|u,i) = \rho(t,u) \times \rho(t,i) \tag{A.10}$$

De là, notre proposition se base sur le fait que, dans les annotations des utilisateurs, la pertinence des tags diminue en fonction de leurs rangs. Nous observons cette tendence sur l'ensemble de nos cinq jeux de données. La figure A.5 montre l'évolution décroissante de la pertinence des tags en fonction de leurs rangs et relativement à la pertinence du premier tag de la liste.

Nous proposons deux modèles de mesure de pertinence pour les listes de recommandations :

1. Le premier modèle que nous appelons *blsC* définit la pertinence d'une liste à travers son pourcentage de tags déjà utilisés par l'utilisateur et aussi pour annoté la ressource. Quoique assez simpliste, il donne de bonnes performances.

7. *bls* pour « best list size »

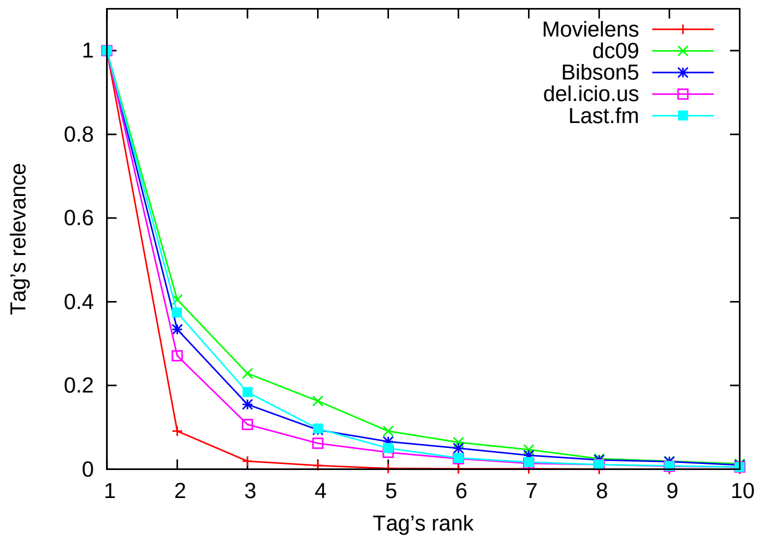


FIGURE A.5 – Pertinence relative des tags en fonction de leurs rangs et du premier

2. Comme second modèle de pertinence, nous traduisons la décroissance de la pertinence des tags en fonction de leurs rangs dans la liste à travers une loi de Zipf-Mandelbrot[8]. Ainsi, nous établissons la pertinence d'une liste ou sous-liste de recommandations comme suit :

$$Rel(L_N|u,i) = \frac{\sum_{t \in L_N} Rel(t|u,i)}{\sqrt{\frac{1}{2}(N + Max)}} \tag{A.11}$$

   A travers cette définition, nous sommes capables de trouver une sous-liste de taille optimale à conserver tout en promouvant celles qui sont de longues tailles. En outre, lorsque plusieurs sous-listes sont toutes optimales, nous prenons la plus longue d'entre elles. Nous appelons ce deuxième modèle par *blsC_v2*.

Nos expériences confirment la supériorité de *blsC_v2* par rapport aux autres approches que sont *blsC* et les combinaisons linéaires généralement utilisées dans la littérature. La figure A.6 met en évidence le maintient de la qualité des recommandations apporté par l'optimisation de la taille des listes. Sur l'ensemble des expériences que nous avons menées, les combinaisons linéaires (*LC_bls* sur la figure) ne surpassent nos deux modèles que sur 9,19% des cas. Pour tout le reste, *blsC_v2* propose les meilleures tailles à garder pour 60% des cas. Il s'en suit *blsC* avec 30% des cas. Ceci confirme l'efficacité de notre approche qui, en plus, ne dépend d'aucun paramètre.

8. C'est une loi de puissance sur des données ordonnées

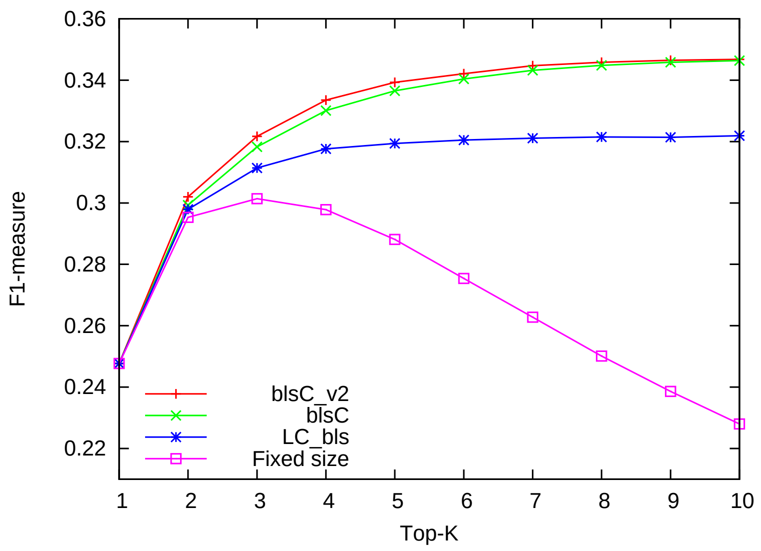


FIGURE A.6 – Evolution de la qualité selon la taille des listes de recommandations

Conclusion

Durant notre thèse, nous avons étudié différents cadres de recommandation : de la prédiction de notes d'utilisateurs à la recommandation de tags.

Toutes nos contributions prennent en compte à la fois la passage à l'échelle de la méthode proposée et la qualité de ses recommandations. Pour ce faire, et à défaut de proposer une nouvelle approche, nous avons à chaque fois étudié l'état de l'art et choisi un candidat, parmi les meilleures solutions du moment et reconnu comme tel, que nous avons essayé d'améliorer. Ainsi, pour chacun de cadres cités ci-dessus, nous avons proposé et évalué des approches distribuées ou au moins optimisant le temps d'exécution.

Dans toutes nos expérimentations, nous nous sommes comparés à des techniques reconnues de recommandation. En dernière étape, nous avons proposé une méthode permettant d'épurer une liste d'objets recommandés et de n'en conserver que les éléments pouvant maximiser la satisfaction de l'utilisateur. Cette nouvelle méthode est adaptable à beaucoup de systèmes de recommandation.